インテル® C++ Composer XE 2013 Windows* 版インストール・ガイドおよびリリースノート

資料番号: 321414-004JA
2013 年 3 月 5 日

目次

1 概要

このドキュメントでは、製品のインストール方法、新機能、変更された機能、注意事項、および製品ドキュメントに記述されていない既知の問題について説明します。

1.1 変更履歴

このセクションでは製品アップデートにおける重要な変更内容を説明します。各コンポーネントの新機能の詳細は、各コンポーネントのリリースノートを参照してください。

1.1.1 Update 3 (2013.3)

- インテル® C++ コンパイラー XE 13.1.1
- インテル® マス・カーネル・ライブラリー 11.0 Update 3
- インテル® スレッディング・ビルディング・ブロック 4.1 Update 3
- 10 進浮動小数点をチェックする関数のドキュメント
- 報告された問題の修正

1.1.2 Update 2 (2013.2)

- インテル® C++ コンパイラー XE 13.1.0
- インテル® マス・カーネル・ライブラリー 11.0 Update 2
- インテル® スレッディング・ビルディング・ブロック 4.1 Update 2
- OpenMP* SIMD 機能のサポート
- 新しい KMP_PLACE_THREADS 環境変数
- 新しい INTEL_PRE_CFLAGS 環境変数と_INTEL_POST_CFLAGS 環境変数
- 新しい /Qvec-report7 ベクトル化レポートレベル
- プリコンパイル済みヘッダーの診断を有効/無効にする /W[no-]pch-messages オプション
- 報告された問題の修正

1.1.3 Update 1 (2013.1)

- インテル® C++ コンパイラー XE 13.0.1
- インテル® マス・カーネル・ライブラリー 11.0 Update 1
- インテル® インテグレーテッド・パフォーマンス・プリミティブ 7.1 Update 1
- インテル® スレッディング・ビルディング・ブロック 4.1 Update 1
- マニュアル CPU ディスパッチに core_4th_gen_avx を追加
- インライン・アセンブリーと組込み関数での新しいインテル® アーキテクチャー (開発コード名: Broadwell) のサポート
- Microsoft* ループプラグマのサポート
- /Qcheck-pointers:w
- 報告された問題の修正

1.1.4 インテル® C++ Composer XE 2011 からの変更点

- インテル® C++ コンパイラーがバージョン 13.0 にアップデート
- インテル® Parallel Debugger Extension の削除
- インテル® マス・カーネル・ライブラリーがバージョン 11.0 にアップデート
    - インテル® Pentium® III プロセッサーのサポートが終了。詳細は、削除された機能に関するナレッジベースの記事 (英語) を参照してください。
- インテル® インテグレーテッド・パフォーマンス・プリミティブがバージョン 7.1 にアップデート

    - インテル® IPP スタティック・スレッド・ライブラリーを別のパッケージで提供
- インテル® スレッディング・ビルディング・ブロックがバージョン 4.1 にアップデート
- Microsoft* Windows Vista* および Windows Server* 2003 のサポートが終了
- Microsoft* Visual Studio* 2005* のサポートが終了
- 製品のアップデートとライセンスのアクティベーションを管理するインテル® Software Manager の追加
- 新しい C++11 機能
- 将来のインテル® プロセッサーに対するサポートの強化
- 新しいインテル® パフォーマンス・ウィザード
- 範囲外のメモリーチェック
- スタティック解析の改良

1.2 製品の内容

インテル® C++ Composer XE 2013 Update 3 Windows 版*には、次のコンポーネントが含まれています。

- インテル® C++ コンパイラー XE 13.1.1。Windows* オペレーティング・システムを実行する IA-32 およびインテル® 64 アーキテクチャー・システムで動作するアプリケーションをビルドします。
- インテル® インテグレーテッド・パフォーマンス・プリミティブ 7.1 Update 1
- インテル® マス・カーネル・ライブラリー 11.0 Update 3
- インテル® スレッディング・ビルディング・ブロック 4.1 Update 3
- Microsoft* 開発環境への統合
- サンプルプログラム
- 各種ドキュメント

1.3 動作環境

アーキテクチャー名についての説明は、http://intel.ly/q9JVjE (英語) を参照してください。

- インテル® ストリーミング SIMD 拡張命令 2 (インテル® SSE2) 対応の IA-32 またはインテル® 64 アーキテクチャー・プロセッサーをベースとするコンピューター (インテル® Pentium® 4 プロセッサー以降、または互換性のあるインテル以外のプロセッサー)
    - 機能を最大限に活用できるよう、マルチコアまたはマルチプロセッサー・システムの使用を推奨します。
- RAM 1GB (2GB 推奨)
- 4GB のディスク空き容量 (すべての機能およびすべてのアーキテクチャー)
- Microsoft* Windows* XP、Microsoft* Windows* 7、Microsoft* Windows* 8、Microsoft* Windows Server* 2008、Microsoft* Windows HPC Server* 2008、Microsoft* Windows Server* 2012 (エンベデッド・エディションはサポートされていません)
    - Microsoft* Windows Server* 2008 または Windows HPC Server* 2008 では Microsoft* Visual Studio* 2010 または Microsoft* Visual Studio* 2008 SP1 が必要です。
    - Microsoft* Windows* 8 および Microsoft* Windows Server* 2012 では、製品は「デスクトップ」環境にインストールされます。「Windows* 8 UI」アプリケーションの開発はサポートされていません。[4]
- IA-32 対応アプリケーションまたはインテル® 64 対応アプリケーションのビルドに、Microsoft* Visual Studio* 開発環境あるいはコマンドライン・ツールを使用する場合は、次のいずれか:
    - Microsoft* Visual Studio* 2012 Standard Edition 以上 (C++ コンポーネントがインストールされていること)

    - Microsoft* Visual Studio* 2010 Standard Edition 以上 (C++ と [x64 コンパイラおよびツール] コンポーネントがインストールされていること) [1]
    - Microsoft* Visual Studio* 2008 Standard Edition 以上 (C++ と [x64 コンパイラおよびツール] コンポーネントがインストールされていること) [1]
- IA-32 アーキテクチャー・アプリケーションのビルドに、コマンドライン・ツールのみを使用する場合は、次のいずれか:
    - Microsoft* Visual C++* Express 2012 for Windows Desktop
    - Microsoft* Visual C++* 2010 Express Edition
    - Microsoft* Visual C++* 2008 Express Edition
- インテル® 64 対応アプリケーションのビルドに、コマンドライン・ツールのみを使用する場合は、次のいずれか:
    - Microsoft* Visual C++* Express 2012 for Windows Desktop
    - Microsoft* Windows* Software Development Kit (SDK) for Windows* 7 and .NET Framework 4
- ドキュメントの参照用に Adobe* Reader* 7.0 以降

注:

1. Microsoft* Visual Studio* 2008 Standard Edition では、[x64 コンパイラおよびツール] コンポーネントがデフォルトでインストールされます。Professional 以上のエディションでは、[カスタム] インストールが必要です。Microsoft* Visual Studio* 2010 では、このコンポーネントがデフォルトで含まれています。
2. インテル® コンパイラーは、デフォルトで、インテル® SSE2 命令対応のプロセッサー (例: インテル® Pentium® 4 プロセッサー) が必要な IA-32 アーキテクチャー・アプリケーションをビルドします。コンパイラー・オプションを使用して任意の IA-32 アーキテクチャー・プロセッサー上で動作するコードを生成できます。ただし、アプリケーションでインテル® インテグレーテッド・パフォーマンス・プリミティブまたはインテル® スレッディング・ビルディング・ブロックを使用している場合、そのアプリケーションの実行には、インテル® SSE2 命令対応のプロセッサーが必要です。
3. アプリケーションは、上記の開発用と同じ Windows* バージョンで実行できます。また、Windows XP よりも前の非エンベデッドの Microsoft Windows 32 ビット・バージョンでも実行できますが、インテルではこれらの互換性テストは行われていません。開発アプリケーションが、古いバージョンの Windows* にはない Win32* API ルーチンを使用している可能性があります。アプリケーションの互換性テストをご自身の責任で行ってください。アプリケーションを実行するには、特定のランタイム DLL をターゲットシステムにコピーしなければならないことがあります。
4. インテル® C++ Composer XE は、Windows 8* UI アプリケーションの開発をサポートしていません。インテルでは、ユーザーの皆様のご意見を常に参考にしています。例えば、Windows* 8 UI アプリケーションにインテル® C++ Compiler XE またはその他のインテル® ソフトウェア開発製品の機能を利用したい方は、インテル® プレミアサポート (https://premier.intel.com/) からご意見をお送りください。
   Windows* 8 UI アプリケーションの開発で、このサポートされていないインテル® ソフトウェア開発製品の機能をテストすることにご興味のある方は、「Experimenting with Intel C++ Composer XE for Windows and Windows 8 Store Apps」(http://intel.ly/WLeXRo) (英語) をお読みください。

1.3.1 IA-64 アーキテクチャー (インテル® Itanium®) 開発のサポートを終了

本バージョンでは、IA-64 アーキテクチャー (インテル® Itanium®) システム上、または IA-64 アーキテクチャー・システム向けの開発をサポートしていません。インテル® コンパイラー 11.1 ではまだサポートされています。

1.3.2 Windows Server* 2003 および Windows Vista* のサポートを終了

Windows Server* 2003 および Windows Vista* のサポートを終了しました。これらのオペレーティング・システムを使用している場合は、新しいバージョンへの移行を推奨します。

1.3.3 Visual Studio* 2005 のサポートを終了

Visual Studio* 2005 のサポートを終了しました。Visual Studio* 2005 を使用している場合は、新しいバージョンへの移行を推奨します。

1.4 ドキュメント

製品ドキュメントは、「インストール先フォルダー」で示されているように、`Documentation` フォルダーに保存されています。

最適化に関する注意事項

インテル® コンパイラーは、互換マイクロプロセッサー向けには、インテル製マイクロプロセッサー向けと同等レベルの最適化が行われない可能性があります。これには、インテル® ストリーミング SIMD 拡張命令 2 (インテル® SSE2)、インテル® ストリーミング SIMD 拡張命令 3 (インテル® SSE3)、ストリーミング SIMD 拡張命令 3 補足命令 (SSSE3) 命令セットに関連する最適化およびその他の最適化が含まれます。インテルでは、インテル製ではないマイクロプロセッサーに対して、最適化の提供、機能、効果を保証していません。本製品のマイクロプロセッサー固有の最適化は、インテル製マイクロプロセッサーでの使用を目的としています。インテル® マイクロアーキテクチャーに非固有の特定の最適化は、インテル製マイクロプロセッサー向けに予約されています。この注意事項の適用対象である特定の命令セットの詳細は、該当する製品のユーザー・リファレンス・ガイドを参照してください。

改訂 #20110804

1.5 サンプル

製品コンポーネントのサンプルは、「インストール先フォルダー」の説明にある `Samples` フォルダーに用意されています。

1.6 日本語サポート

インテル® コンパイラーは、日本語ユーザー向けのサポートを提供しています。エラーメッセージ、ビジュアル開発環境ダイアログ、ドキュメントの一部が英語のほかに日本語でも提供されています。エラーメッセージやダイアログの言語は、システムの言語設定に依存します。日本語版ドキュメントは、Documentation および Samples ディレクトリー以下の `ja_JP` サブディレクトリーにあります。

日本語サポート版は、インテル® C++ Composer XE 2013 のアップデートとして提供されます。

日本語サポート版を英語のオペレーティング・システムで使用する場合や日本語のオペレーティング・システムで英語サポート版を使用する場合は、http://intel.ly/oZjpZs (英語) の説明を参照してください。

1.7 テクニカルサポート

インストール時に製品の登録を行わなかった場合は、インテル® ソフトウェア開発製品レジストレーション・センター (http://registrationcenter.intel.com) で登録してください。登録を

行うことで、サポートサービス期間中 (通常は 1 年間)、製品アップデートと新しいバージョンの入手を含む無償テクニカルサポートが提供されます。

テクニカルサポート、製品のアップデート、ユーザーフォーラム、FAQ、ヒント、およびその他のサポート情報は、http://www.intel.com/software/products/support/ (英語) を参照してください。

注: 代理店がテクニカルサポートを提供している場合は、インテルではなく代理店にお問い合わせください。

2 インストール

2.1 インテル® Software Manager

インテル® Software Manager は、製品アップデートの配信方法を簡素化し、現在インストールされているすべてのインテル® ソフトウェア製品のライセンス情報とステータスを表示します。

将来の製品設計の参考のため、製品使用状況に関する匿名情報をインテルに提供する、インテル® ソフトウェア向上プログラムに参加できます。このプログラムは、デフォルトで無効になっていますが、インストール中または後から有効にして参加できます。参加はいつでも取りやめることができます。詳細は、http://intel.ly/SoftwareImprovementProgram (英語) を参照してください。

2.2 インストール前の準備

2.2.1 64 ビット・アプリケーション用の Visual Studio* の設定

Microsoft* Visual Studio* 2008 を使用し、64 ビット・アプリケーション (インテル® 64 アーキテクチャー向け) を開発する場合は、Visual Studio* の構成を変更して、64 ビット・サポートを追加します。

Visual Studio* 2008 Standard Edition または Visual Studio* 2010 Professional Edition 以上を使用する場合は、インテル® 64 対応アプリケーションのビルド用に構成を変更する必要はありません。その他のエディションの場合は、次の操作を行ってください。

1. [コントロール パネル] の [プログラムの追加と削除] から [Microsoft Visual Studio 2008] を選択し、[変更と削除] をクリックします。[Visual Studio メンテナンス モード] ウィンドウが表示されます。[次へ] をクリックします。
2. [機能の追加と削除] をクリックします。
3. [選択した機能をインストールします] で [言語ツール] の [Visual C++] を展開します。
4. [x64 コンパイラおよびツール] ボックスがオンになっていない場合は、オンにし、[更新] をクリックします。ボックスがオンの場合は、[キャンセル] をクリックします。

Visual C++* Express Edition では 64 ビットの開発はサポートされていません。

2.3 インストール

本製品のインストールには、有効なライセンスファイルまたはシリアル番号が必要です。本製品を評価する場合には、インストール時に [製品を評価する (シリアル番号不要)] オプションを選択してください。

DVD で製品を受け取った場合、製品 DVD を DVD ドライブに挿入します。自動でインストールが開始されます。自動で開始されない場合は、Windows* エクスプローラで DVD ドライブのトップレベル・ディレクトリーを開き、setup.exe をダブルクリックします。

製品のダウンロード版を購入した場合は、ダウンロードしたファイル (.EXE) をダブルクリックして、インストールを開始します。利用可能なダウンロード・ファイルには各種あり、それぞれ異なるコンポーネントの組み合わせを提供していることに注意してください。ダウンロード・ページを注意深くお読みになり、適切なファイルを選択してください。

新しいバージョンをインストールする前に古いバージョンをアンインストールする必要はありません。新しいバージョンは古いバージョンと共存可能です。

2.3.1 PATH 環境変数の変更によるコマンドシェル (cmd.exe) への一時的な影響

Windows* 7 または Windows* 8 では、インストーラーが PATH 環境変数に項目を追加すると、PATH の長さが非常に長くなり (2000-4000 文字)、システムを再起動するまで Windows* コマンドプロンプト (cmd.exe) が動作しなくなることがあります。システムを再起動しても同じ問題が発生する場合は、テクニカルサポート までお問い合わせください。

2.3.2 サイレント・インストール

自動インストール、「サイレント」インストール機能についての詳細は、
http://intel.ly/nKrzhv (英語) を参照してください。

2.3.3 クラスターでのインストール

インストールするマシンに Microsoft* Compute Cluster Pack のライセンスがあり、クラスターメンバーの場合、「フル・インストール」を選択すると、そのクラスターのアクセス可能なすべてのノードに製品がインストールされます。「カスタム・インストール」を選択すると、現在のノードのみにインストールするオプションを選択できます。

2.3.4 ライセンスサーバーの使用

「フローティング・ライセンス」を購入された場合は、ライセンスファイルまたはライセンスサーバーを使用したインストール方法について、http://intel.ly/pjGfwC (英語) を参照してください。この記事には、多様なシステムにインストールすることができるインテル・ライセンス・サーバーに関する情報も記述されています。

2.4 製品の変更、更新、削除

Windows* のコントロールパネルの [プログラムの追加と削除] でインストールまたは削除する製品コンポーネントを変更します。

製品のアップデート・バージョンをインストールする際、古いバージョンを最初にアンインストールする必要はありません。複数のバージョンのコンパイラーをインストールし、その中から選択して使用することができます。新しいバージョンのコンパイラーを削除した場合、以前のバージョンの Microsoft* Visual Studio* への統合を再インストールする必要があります。

2.5 インストール先フォルダー

インストール・フォルダーの構成を以下に示します。一部含まれていないフォルダーもあります。

- `C:\Program Files\Intel\Composer XE 2013`
  - `bin`
    - `ia32`

- ia32_intel64
- intel64
- sourcechecker

- compiler
  - include
    - cilk
    - ia32
  - lib
    - ia32
    - intel64
  - perf_headers
- Documentation
  - en_US
    - compiler_c
      - cl
    - gs_resources
    - ipp
      - get_started_files
      - ipp_userguide
      - tutorials
    - mkl
      - get_started_files
      - mkl_userguide
      - tutorials
    - ssadiag_docs
    - tbb
      - get_started_files
      - html
      - tutorial
    - tutorials
      - cmp_gap_c
      - cmp_thd_c
      - cmp_vec_c
  - msvhelp
    - 1033
      - compiler_c
      - ipp
      - mkl
      - ssadiag
      - tbb
  - vshelp
    - intel.cppprodocs
    - intel.cprocompilerdocs
    - intel.ippdocs
    - intel.mkldocs
    - intel.sssadiag
    - intel.tbbdocs
- Help
- ipp
  - bin
  - demo
  - include
  - interfaces

- lib
- tools

- mkl
  - benchmarks
  - bin
  - examples
  - include
  - interfaces
  - lib
  - tests
  - tools
- redist
  - ia32
    - compiler
      - 1033
      - irml
      - irml_c
    - ipp
      - 1033
    - mkl
      - 1033
    - tbb
      - vc_mt
      - vc8
      - vc9
      - vc10
  - intel64
    - compiler
      - 1033
      - irml
      - irml_c
    - ipp
      - 1033
    - mkl
      - 1033
    - tbb
      - vc_mt
      - vc8
      - vc9
      - vc10
- Samples
  - en_US
    - C++
    - ipp
- tbb
  - bin
  - examples
  - include
    - serial
      - tbb
    - tbb
      - compat
      - internal

```
                o  machine
        ▪  lib
            •  ia32
                o  vc_mt
                o  vc8
                o  vc9
                o  vc10
            •  intel64
                o  vc_mt
                o  vc8
                o  vc9
                o  vc10
    o  VS Integration
        ▪  C++
            •  VS2008
```

bin、include および lib 配下のフォルダーは次のとおりです。

- ia32: IA-32 上で動作するアプリケーションのビルドに使用するファイル
- intel64: インテル® 64 上で動作するアプリケーションのビルドに使用するファイル
- ia32_intel64: IA-32 上での実行用のコンパイラー。インテル®64 上で動作するアプリケーションをビルドします。

英語以外の Windows* システムにインストールする場合、Program Files フォルダー名が異なる場合があります。インテル® 64 アーキテクチャー・システムでは、フォルダー名は Program Files (X86) またはそれに相当する名前です。

デフォルトでは、アップデートによって既存のディレクトリーの内容が置換されます。最初のアップデートをインストールするときに、以前のインストールとは別に新しいアップデートをインストールして、システムに両方のファイルを残すオプションを選択できます。両方を残すオプションを選択した場合、古いアップデートのトップレベルのフォルダー名は Composer XE 2013.*nnn* (*nnn* はアップデート番号) に変更されます。

3 インテル® C++ コンパイラー

このセクションでは、インテル® C++ コンパイラーの変更点、新機能、および最新情報をまとめています。

3.1 互換性

バージョン 11 では、IA-32 システムのデフォルトでのコード生成において、アプリケーションを実行するシステムでインテル® ストリーミング SIMD 拡張命令 2 (インテル® SSE2) がサポートされていると仮定するように変更されました。詳細は、下記を参照してください。

3.2 新機能と変更された機能

インテル® C++ Composer XE 2013 には、インテル® C++ コンパイラー XE 13.0 が含まれています。このバージョンでは、次の機能が新たに追加または大幅に拡張されています。これらの機能に関する詳細は、ドキュメントを参照してください。

- 第 3 世代インテル® Core™ プロセッサー・ファミリー (/QxCORE-AVX-I および /QaxCORE-AVX-I) とインテル® アドバンスト・ベクトル・エクステンション (インテル® AVX) 対応の将来のインテル® プロセッサー (/QxCORE-AVX2 および /QaxCORE-AVX2) のサポートを強化

- 最適なコンパイラー・オプション設定の識別を支援する新しいインテル® パフォーマンス・ウィザード (Microsoft* Visual Studio* で、[Intel Composer XE 2013 (インテル(R) Composer XE 2013)] -> [Start Performance Wizard (パフォーマンス・ウィザードを開始] を選択)
- C++11 の機能 (/Qstd:c++0x)
    - 追加の型特性
    - 統一的な初期化構文
    - 汎用化された定数式 (一部サポート)
    - noexcept
    - 範囲に基づく for ループ
    - ラムダから関数ポインターへの変換
    - 暗黙の移動コンストラクターと移動代入演算子
- 範囲外のメモリーチェック (/Qcheck-pointers)
- スタティック解析の改良:
    - スタティック解析の解析を制御する追加オプション (/Qdiag-enable:sc-{full|concise|precise})
    - Windows* ワンクリック・インターフェイスの拡張
    - 新しいプログレスバー
    - Microsoft* SAL 言語拡張のサポート

3.2.1 Microsoft* Visual Studio* 2012 のサポート

Microsoft* Visual Studio* 2012 でのデスクトップ・アプリケーションの開発がサポートされました。「Windows* 8 UI」アプリケーションの開発はサポートされていません。

3.2.2 新しい OpenMP* SIMD 機能のサポート (インテル® Composer XE 2013 Update 2)

インテル® Composer XE 2013 Update 2 では、策定前の OpenMP* 機能のサポートが追加されました。http://openmp.org (英語) の「OpenMP* 4.0 Public Review Release Candidate 1」仕様で定義されている SIMD プラグマ、宣言子、節がサポートされています。詳細は、http://intel.ly/W7CHjb (英語) を参照してください。

3.2.3 KMP_PLACE_THREADS 環境変数の追加 (インテル® Composer XE 2013 Update 2)

この環境変数を使用すると、ユーザーは明示的なアフィニティー設定やプロセス・アフィニティー・マスクを記述する代わりに、OpenMP* アプリケーションで使用するコア数およびコアごとのスレッド数を簡単に指定することができます。

構文

```
value = ( int [ "C" | "T" ] [ delim ] | delim ) [ int [ "T" ]
[ delim ] ] [ int [ "O" ] ];
delim = "," | "x";
```

効果

使用するコア数、オフセット値 (オプション)、コアごとのスレッド数を指定します。"C" はコア数、"T" はスレッド数、"O" はオフセットを示します。コア数またはスレッド数のいずれかを指定する必要があります。省略した場合、デフォルト値は利用可能なコア数 (スレッド数) です。

例

5C,3T,10 - オフセット 1 で 5 コア、コアごとに 3 スレッド使用します。
5,3,1 - 上記と同じです。

24 - 最初の 24 コア、コアごとに利用可能なすべてのスレッドを使用します。
2T - すべてのコア、コアごとに 2 スレッド使用します。
,2 - 上記と同じです。
3x2 - 3 コア、コアごとに 2 スレッド使用します。
4C120 - オフセット 12 で 4 コア、コアごとに利用可能なすべてのスレッドを使用します。

3.2.4 新しい __INTEL_PRE_CFLAGS 環境変数と __INTEL_POST_CFLAGS 環境変数の追加 (インテル® Composer XE 2013 Update 2)

インテル® Composer XE 2013 Update 2 では、ホストシステムの環境変数からインテル® コンパイラーのコマンドライン・オプションを指定する機能が追加されました。この機能は、コンパイラー設定ファイル icl.cfg ですでに提供されている機能を拡張したものです。コマンドライン・オプションは、INTEL_PRE_CFLAGS 環境変数を使用してプリフィックスの位置に、または INTEL_POST_CFLAGS 環境変数を使用してポストフィックスの位置に挿入することができます。コマンドラインは次のようになります。

```
icl <PRE フラグ> <設定ファイルのフラグ> <コンパイラー呼び出しのフラグ> <POST フラグ>
```

3.2.5 マニュアル CPU ディスパッチに core_4th_gen_avx を追加 (インテル® Composer XE 2013 Update 1)

cpu_dispatch および cpu_specific のマニュアル CPU ディスパッチ・メカニズムに cpuid core_4th_gen_avx のサポートが追加されました。この cpuid は、インテル® アドバンスト・ベクトル・エクステンション 2 (インテル® AVX 2) をサポートするプロセッサーをターゲットにします。

3.2.6 Microsoft* のループプラグマ構文のサポート (インテル® Composer XE 2013 Update 1)

Update 1 では、Microsoft* Visual C++* 2012 コンパイラーの `#pragma loop [hint_parallel(n),no_vector,ivdep]` のサポートが追加されました。

3.2.7 インライン・アセンブリーと組込み関数での新しいインテル® アーキテクチャー (開発コード名: Broadwell) のサポート (インテル® Composer XE 2013 Update 1)

新しいインテル® アーキテクチャー (開発コード名: Broadwell) の一部の新しい命令がサポートされました。インテル® Composer XE 2013 Update 1 では、インライン・アセンブリーと組込み関数でこれらの命令をサポートします。組込み関数は、immintrin.h に定義されています。

```
extern int _rdseed16_step(unsigned short *random_val);
extern int _rdseed32_step(unsigned int *random_val);
extern int _rdseed64_step(unsigned __int64 *random_val);
```

これらの組込み関数は、16/32/64 ビットの乱数整数を生成します。これらの組込み関数は、ハードウェア命令 RDSEED にマップされます。生成された乱数は指定されたメモリーの場所に書き込まれ、ステータス (ハードウェアによって有効な乱数が返された場合は -1、そうでない場合は 0) が返されます。

rdseed と rdrand 組込み関数の違いは、rdseed が NIST SP 800-90B および NIST SP 800-90C 標準に準拠し、rdrand が NIST SP 800-90A 標準に準拠することです。

```
extern unsigned char _addcarry_u32(unsigned char c_in, unsigned int
src1, unsigned int src2, unsigned int *sum_out);
```

```
extern unsigned char _addcarry_u64(unsigned char c_in, unsigned
__int64 src1, unsigned __int64 src2, unsigned __int64 *sum_out);
```

これらの組込み関数は、2 つの 32/64 ビット整数値 (src1, src2) とキャリーイン値の合計を計算します。キャリーイン値は、c_in 入力値が非ゼロの場合は 1、その他の場合は 0 と見なされます。計算の結果は、sum_out 引数で指定したメモリー位置に格納されます。

```
*sum_out = src1 + src2 + (c_in !=0 ? 1 : 0)
```

この組込み関数は sum_out で指定されたメモリーアドレスの有効性チェックは行わないため、合計の結果を格納しないでキャリーアウトが発生したかどうか確認するために使用できません。組込み関数の戻り値は、合計により生成されるキャリーアウト値です。計算の結果は、sum_out 引数で指定したメモリー位置に格納されます。

```
extern unsigned char _subborrow_u32(unsigned char b_in, unsigned int
src1, unsigned int src2, unsigned int *diff_out);

extern unsigned char _subborrow_u64(unsigned char b_in, unsigned
__int64 src1, unsigned __int64 src2, unsigned __int64 *diff_out);
```

これらの組込み関数は、32/64 ビットの符号なし整数値 src2 とボローイン値の合計を計算して、合計の結果を 32/64 ビットの符号なし整数値 src1 から引きます。ボローイン値は、c_in 入力値が非ゼロの場合は 1、その他の場合は 0 と見なされます。計算の結果は、diff_out 引数で指定したメモリー位置に格納されます。

```
*diff_out = src1 + (src2 + (b_in !=0 ? 1 : 0))
```

この組込み関数は diff_out で指定されたメモリーアドレスの有効性チェックは行わないため、減算の結果を格納しないでボローアウトが発生したかどうか確認するために使用できません。組込み関数の戻り値は、減算により生成されるボローアウト値です。減算の結果は、diff_out 引数で指定したメモリー位置に格納されます。

```
extern unsigned char _addcarryx_u32(unsigned char c_in, unsigned int
src1,   unsigned int src2, unsigned int *sum_out);

extern unsigned char _addcarryx_u64(unsigned char c_in, unsigned
__int64 src1,   unsigned __int64 src2, unsigned __int64 *sum_out);
```

これらの組込み関数は、2 つの 32/64 ビット整数値 (src1, src2) とキャリーイン値の合計を計算します。キャリーイン値は、c_in 入力値が非ゼロの場合は 1、その他の場合は 0 と見なされます。計算の結果は、sum_out 引数で指定したメモリー位置に格納されます。

```
*sum_out = src1 + src2 + (c_in !=0 ? 1 : 0)
```

この組込み関数は sum_out で指定されたメモリーアドレスの有効性チェックは行わないため、合計の結果を格納しないでキャリーアウトが発生したかどうか確認するために使用できません。
この組込み関数は、コンパイラーの決定に応じて ADCX または ADOX 組込み関数に変換されます。これらの組込み関数は、2 つのインターリーブされたキャリーあり加算命令シーケンスを、ADCX および ADOX 組込み関数を使用して並列実行できるようにします。組込み関数の戻り値は、合計により生成されるキャリーアウト値です。合計の結果は、sum_out 引数で指定したメモリー位置に格納されます。

_mm_prefetch 組込み関数の新しい _MM_HINT_ET0 ヒント

_MM_HINT_ET0 ヒントは、組込み関数を新しいインテル® アーキテクチャー (開発コード名: Broadwell) でサポートされる PREFETCHW 命令に変更します。_MM_HINT_ET0 を使用する前に、ターゲット CPU が PREFETCHW 命令をサポートしているかどうか確認してください。

3.2.8 インテル® Cilk™ Plus の “scalar” 節の削除

本リリースで、インテル® Cilk™ Plus の要素関数のオプションとして使用されていた “scalar” 節が削除されました。代わりに、機能的に同じ “uniform” 節を使用してください。

3.2.9 インテル® アドバンスト・ベクトル・エクステンション 2 (インテル® AVX 2) 命令のサポート (Update 7)

インテル® C++ Composer XE 2011 Update 7 のコンパイラーは、インライン・アセンブリーと 組込み関数 (immintrin.h) でインテル® AVX 2 命令をサポートします。

3.2.10 インテル® Cilk™ Plus の配列表記 (アレイ・ノーテーション) セマンティクスの変更 (2011 Update 6)

インテル® C++ Composer XE 2011 では、次のようなインテル® Cilk™ Plus の部分配列の代入は、結果の一時コピーが生成されるためパフォーマンスに影響します。

```
a[:]= b[:]+ c[:];
```

インテル® C++ Composer XE 2011 Update 6 から、代入式の右辺の部分配列 (上記の例では b[:] や c[:]) の一部が左辺の部分配列 (上記の例では a[:]) とメモリー上でオーバーラップする場合、そのような代入の結果は不定となります。意図する動作が得られるように、代入式でメモリー上の部分的なオーバーラップが発生しないようにするのはプログラマーの責任です。

ただし、次のように、部分配列が完全にオーバーラップする場合は例外です。

```
a[:]= a[:]+ 3;
```

この場合、配列が完全にオーバーラップするため、意図したとおりに動作し、期待どおりの結果が得られます。

3.2.11 /Qsox オプションの追加キーワード、デフォルトの変更 (2011 Update 3)

オブジェクト・ファイルおよび実行ファイルに使用されたコンパイラー・オプションとプロシージャーのプロファイル情報を追加するための /Qsox オプションは、インライン展開された関数のリストを含めたり、プロシージャーのプロファイル情報を除外できるようになりました。

/Qsox の構文は次のように変更されました。

```
/Qsox[-]
/Qsox=keyword[,keyword]
```

keyword には、inline または profile のいずれかを指定できます。キーワードなしで /Qsox を指定すると、以前のリリースとは異なり、コマンドライン・オプションの情報のみが追加されます。以前のリリースと同じ動作にするには、/Qsox=profile を使用してください。/Qsox オプションはコマンドラインで複数回指定することができますが、その場合は左から右の順に解釈されます。

この情報は、オブジェクト・ファイルにコメントとして追加されます。Visual Studio* 2005 以降の Microsoft* のリンカーではこれらのコメントは無視されるため、実行ファイルにはこの情報は含まれません。

3.2.12 3 つの組込み関数の変更 (2011 Update 2)

3 つの組込み関数 (_rdrand16_step()、_rdrand32_step()、_rdrand64_step()) は、Update 2 で変更されました。この変更は、ドキュメントにはまだ反映されていません。これらの組込み関数は、"immintrin.h" ヘッダーファイルで宣言されており、ハードウェアにより生成される乱数値を返します。

3 つとも同じ RDRAND 命令にマップされ、16/32/64 ビットの乱数整数を生成します。

構文

1. extern int _rdrand16_step(unsigned short *random_val);
2. extern int _rdrand32_step(unsigned int *random_val);
3. extern int _rdrand64_step(unsigned __int64 *random_val);

説明

これらの組込み関数は、RDRAND 命令を使用して、ハードウェアにより生成される乱数値の取得を 1 回試みます。生成された乱数値は指定されたメモリー位置に書き込まれ、成功ステータスが返されます。ハードウェアにより有効な乱数値が返された場合は 1 を返し、そうでない場合は 0 を返します。

戻り値

ハードウェアにより生成された 16/32/64 乱数値。

制限事項

_rdrand64_step() 組込み関数は、64 ビット・レジスター対応のシステムでのみ使用できます。

3.2.13 スタティック解析機能 (旧:「スタティック・セキュリティー解析」または「ソースチェッカー」) にはインテル® Inspector XE が必要

バージョン 11.1 の「ソースチェッカー」機能が拡張され、「スタティック解析」に名称が変更されました。スタティック解析を有効にするコンパイラー・オプションはバージョン 11.1 と同じですが (例: `/Qdiag-enable:sc`)、解析結果がコンパイラー診断結果ではなく、インテル® Inspector XE で表示可能なファイルに出力されるようになりました。

3.2.13.1 2011 Update 2 からの "inspxe-runsc.exe" コマンドライン・ユーティリティーの変更

インテル® Composer XE 2011 に含まれるこのユーティリティーは、2011 Update 2 から変更されています。この変更は、インテル® Composer XE 2011 を使用してスタティック解析を実行する場合にのみ影響します。スタティック解析を使用しない場合や、このユーティリティーを使用せずにスタティック解析を実行する場合には影響ありません。スタティック解析はインテル® Parallel Studio XE 2011/2013、インテル® Fortran Studio XE 2011/2013 またはインテル® C++ Studio XE 2011/2013 でのみ利用できます。そのため、これらの製品をお使いでない場合は影響ありません。

inspxe-runsc は、アプリケーションのビルド方法を示す**ビルド仕様**を実行します。通常、ビルド仕様ファイルは、ビルドを実行して、実際に行われたコンパイルとリンクを記録することにより生成されます。inspxe-runsc は、インテル® コンパイラーをスタティック解析モードで使用して、再度この処理を行います。スタティック解析結果はリンクステップで生成されるため、inspxe-runsc で複数のリンクステップを持つビルドが含まれるビルド仕様を実行すると、複数のスタティック解析結果が生成されます。

インテル® Composer XE 2011 およびインテル® Composer XE 2011 Update 1 の inspxe-runsc は、すべてのスタティック解析結果を同じディレクトリーに生成します。リンクが複数ある場合、これは、1 つのプロジェクトのスタティック解析結果は同じディレクトリーに 1 つだけでなければならないという規則に違反します。新しいバージョンの inspxe-runsc は、リンクステップごとの結果を個別のディレクトリーに生成することで、この規則に従っています。ディレクトリー名は、リンクされるファイルの名前を基に付けられます。2 つの実行ファイル file1.exe と file2.exe をビルドするプロジェクトのビルド仕様の場合、以前のバージョンの inspxe-runsc では、file1 の結果と file2 の結果 (例えば r000sc と r001sc) が同じディレクトリーに作成されます。新しいバージョンの inspxe-runsc でも結果は 2 つ作成されますが、file1 の結果は “My Inspector XE results – file1/r000sc”、file2 の結果は “My Inspector XE results – file2/r000sc” というように別々のディレクトリーに作成されます。2 つの結果のディレクトリーは同じ親ディレクトリーの下に作成されます。

inspxe-runsc には、結果ファイルの場所を指定する -result-dir (-r) コマンドライン・オプションがあります。このオプションの動作が変更されました。以前は、r000sc のように結果が作成されるディレクトリーの名前を指定していましたが、現在は、“My Inspector XE Results - name” のように結果が作成されるディレクトリーの親ディレクトリーを指定します。つまり、-r オプションのディレクトリー名は、結果の生成される場所から 2 つ上のディレクトリーのものになります。

inspxe-runsc のこの変更により、結果ディレクトリーが効率良く移動します。この変更に伴い、ユーザーによる対応が必要になります。-r オプションを指定して inspxe-runsc を呼び出すスクリプトを使用している場合は、新しい動作に合わせて、-r オプションの引数を変更してください。また、新しいバージョンの inspxe-runsc によって生成されるスタティック解析結果が、以前のバージョンの inspxe-runsc によって生成された結果と同じディレクトリーに保存されることがないように、以前の結果ファイルを新しいディレクトリーに移動する必要があります。以前のバージョンの inspxe-runsc でリンクステップが 1 つのみのビルド仕様を実行した結果は、“My Inspector XE results – name” という形式のディレクトリーに移動します。この操作を行わないと、新しく作成される結果ですべての問題が “新規” として表示されます。以前のバージョンの inspxe-runsc で複数のリンクステップを含むビルド仕様を実行した場合、スタティック解析ではさまざまな問題がありましたが、これらの問題は新しいバージョンを使用することで解決されます。この場合、以前の結果のうち最も新しいものを “My Inspector XE results – name” という形式の新しいディレクトリーに (1 つのディレクトリーに 1 つの結果が含まれるように) コピーします。これにより、新しいバージョンで作成される結果に以前の問題ステート情報が正しく適用される可能性が高くなります。

3.2.14 インテル® C++ プロジェクト・ファイルの互換性

インテル® C++ プロジェクト・ファイル (`.icproj`) の形式がバージョン 13.0 (インテル® Composer XE 2013) で変更されました。インテル® C++ の古いバージョンで作成されたプロジェクトを開くと、プロジェクトの変換が必要である旨のメッセージが表示されます。バージョン 13.0 のプロジェクトを古いバージョンのインテル® C++ 統合で使用することはできません (ただし、古いバージョンのコンパイラーは、[ツール] > [オプション] > [Intel C++ (インテル(R) C++)] > [Compilers (コンパイラー)] から使用できます)。

3.3 新規および変更されたコンパイラー・オプション

コンパイラー・オプションの詳細に関しては、ドキュメントのコンパイラー・オプションのセクションを参照してください。

- /Qvec-report6、/Qvec-report7
- /W[no-]pch-messages
- /Qextend-arguments:[32|64]
- /Qguide-profile:<[file|dir]>[,[file|dir],...]
- /Qcheck-pointers:<arg>

- /Qcheck-pointers-dangling:<arg>
- /Qcheck-pointers-undimensioned[-]
- /Qstd=c++11 (same as /Qstd=c++0x)
- /check:<keyword>[,<keyword>,...]
- /Qdiag-enable:sc-{full|concise|precise}
- /Qdiag-enable:sc-single-file
- /Qdiag-enable:sc-enums
- /watch:<keyword>
- /nowatch
- /Qvc11
- /Qgcc-dialect:<version>
- /Qipp-link:{static|dynamic|static_thread}

廃止予定のコンパイラー・オプションのリストは、ドキュメントのコンパイラー・オプションのセクションを参照してください。

3.3.1 /Qvec-report7 オプションの追加 (インテル® Composer XE 2013 Update 2)

ループのベクトル化についてより詳細で高度な情報を提供する新しいベクトル化レポートレベルが追加されました。詳細は、http://intel.ly/XeSkW6 (英語) の記事を参照してください。

3.3.2 /W[no-]pch-messages オプションの追加 (インテル® Composer XE 2013 Update 2)

プリコンパイル済みヘッダーに関連する診断を有効または無効にする機能が追加されました。

3.3.3 /Qcheck-pointers:w オプションの追加 (インテル® Composer XE 2013 Update 1)

Update 1 では、ポインターチェッカーに書き込みのみのエラーチェックを実行する機能が追加されました。

3.3.4 /Qipp-link オプション

このオプションは、使用するインテル® インテグレーテッド・パフォーマンス・プリミティブのライブラリーを指定します。/Qipp オプションとともに使用されます。static (スタティック・シングルスレッド・ライブラリーにリンク)、dynamic (ダイナミック・ライブラリーにリンク)、static-thread (スタティック・マルチスレッド・ライブラリーにリンク) の 3 つのオプションがあります。スタティック・マルチスレッド・ライブラリーは 別のパッケージで提供されることに注意してください。

3.3.5 廃止予定のオプション

次の方法で廃止予定のすべてのコンパイラー・オプションを確認できます。

1) [スタート] メニューからコマンドプロンプトを開きます: [スタート] > [すべてのプログラム] > [Intel Parallel Studio XE 2013] > [Command Prompt (コマンドプロンプト)] > [Parallel Studio XE with Intel Compiler XE v13.0 [Update xx] (インテル(R) コンパイラー XE 13.0 [Update xx])] > [IA-32/Intel(R) 64 Visual Studio xxx mode (IA-32/インテル(R) 64 Visual Studio xxx モード)] を選択します。
2) 次のコマンドを実行します。

```
>> icl /? deprecate
```

3.4 その他の変更

3.4.1 ビルド環境コマンドスクリプトの変更

ビルド環境を構築するコマンド・ウィンドウ・スクリプトが使用する Microsoft* Visual Studio* バージョンを任意で指定できるよう変更されました。ビルド環境ウィンドウを開く

のに、定義済みのスタート・メニュー・ショートカットを使用していない場合は、次のコマンドを使用して適切な環境を構築してください。

```
"<install-dir>\bin\compilervars.bat" arch [vs]
```

arch はビルドする対象アーキテクチャーを指定します。次のいずれかの値を指定できます。

- `ia32`
- `ia32_intel64`
- `intel64`

vs は任意で指定します。次のいずれかの値を指定できます。*vs* が指定されていない場合は、コマンドライン統合用にインストール時に指定された Visual Studio* のバージョンがデフォルトで使用されます。

- `vs2012`
- `vs2010`
- `vs2008`

また、インテル® Visual Fortran Composer XE 2013 もインストールされている場合、このコマンドによりインテル® Visual Fortran Composer XE 2011 を使用する環境も構築されます。

スクリプトファイル名 iclvars.bat および ifortvars.bat は、以前のリリースとの互換性のために保持されています。

3.4.2 OpenMP* レガシー・ライブラリーの削除

本リリースでは、OpenMP* のレガシー・ライブラリーが削除されました。"互換性がある" ライブラリーのみ提供されます。

3.4.3 OpenMP* スタティック・ライブラリーの削除

本リリースでは、OpenMP* ランタイム・ライブラリーが削除されました。これらのランタイム・ライブラリーは動的にリンクしてください。

3.4.4 バージョン管理システムでのインテル® C++ プロジェクトの使用

プロジェクトがバージョン管理システム (例: Microsoft* Visual SourceSafe* や Microsoft* Visual Studio* Team Foundation Server など) で管理されている場合、プロジェクトでインテル® C++ プロジェクト・システムを使用するには追加のステップが必要です。このトピックについての詳細な記事は、http://intel.ly/plmnpo (英語) を参照してください。

3.5 既知の問題

3.5.1 コンパイラーの既知の問題

3.5.1.1 ドキュメントに含まれていない 10 進浮動小数点のステータスをチェックする関数

10 進浮動小数点演算中に発生する例外を検出するには、次の浮動小数点例外関数を使用します。

関数	説明
fe_dec_feclearexcept	サポートされている浮動小数点例外をクリアします。
fe_dec_fegetexceptflag	実装で定義された浮動小数点ステータスフラグのステートを格納します。
fe_dec_feraiseexcept	サポートされている浮動小数点例外をセットします。
fe_dec_fesetexceptflag	浮動小数点ステータスフラグをセットします。
fe_dec_fetestexcept	現在セットされている浮動小数点例外フラグのサブセットを判断します。

10 進浮動小数点例外関数は、fenv.h ヘッダーファイルで定義されています。

同様の 2 進浮動小数点例外関数は、ISO C99 で説明されています。

DFP を使用してソースをコンパイルするには、プリプロセッサー・マクロ __STDC_WANT_DEC_FP__ を使用します。

3.5.1.2日本語ファイル名に関するコマンドライン診断表示の問題

コンパイル診断で日本語が含まれているファイル名は、ネイティブのインテル® 64 対応アプリケーション用コンパイラーを使用して、Windows* コマンドでコンパイルした場合に正しく表示されません。Visual Studio* を使用する場合やインテル® 64 対応アプリケーション用クロスコンパイラーまたは IA-32 対応アプリケーション用コンパイラーを使用する場合は、この問題は発生しません。

3.5.2 Visual Studio* の既知の問題

3.5.3 Windows Server* 2012 で Visual Studio* 2012 のドキュメントを表示できない場合

Windows Server* 2012 で Visual Studio* 2012 のヘルプまたはドキュメントを表示できない場合、Microsoft* Internet Explorer* のセキュリティー設定を変更すると表示されるようになります。[ツール] > [インターネット オプション] > [セキュリティ] を選択して、[インターネット] ゾーンで [MIME スニッフィングを有効にする] および [アクティブ スクリプト] を有効にします。

3.5.3.1MSVCP90D.dll (またはその他の Microsoft ランタイム DLL) が見つからない*

サンプルプロジェクト (および Microsoft* Visual C++* プロジェクト) を実行するときに Microsoft* Visual Studio* のランタイム DLL. が見つからない場合、ランタイムエラーが発生します。これは、マニフェスト・ファイルや SXS アセンブリーが見つからないことが原因です。この問題を解決するには、使用しているバージョンの Microsoft* Visual Studio* の redist フォルダー (デフォルトの場所は c:\program files[(x86)]\Microsoft Visual Studio X.X\VC\redist) に移動します。amd64、x86、Debug_NonRedist サブフォルダーで、必要なランタイムが含まれているフォルダーを探します (デバッグ・ライブラリーを探す場合は、ファイル名の最後が D のファイルが含まれているフォルダーを探します)。必要なランタイムが含まれているフォルダーが見つかったら、そのフォルダーの (.manifest ファイルを含む) すべての内容を、実行する .exe ファイルのあるフォルダーにコピーします。

3.5.3.2 Visual Studio 2010 では /fp:precise がデフォルトでオン*

Visual Studio* 2010 で作成または変換されたプロジェクトでは、デフォルトで /fp:precise コマンドライン・オプションがオンになります。このオプションは、パフォーマンスを低下させるいくつかの最適化を無効にして、浮動小数点演算の一貫性を向上させる「浮動小数点モデル」を設定します。インテルのデフォルトである /fp:fast に戻すには、プロジェクトのプロパティー・ページで [C/C++] > [Code Generation (コード生成)] > [Floating Point Model (浮動小数点モデル)] を Fast に変更します。

3.5.3.3 Visual Studio 2010 の言語パック*

インテル® C++ Composer XE 2013 をインストールした後に Visual Studio* 2010 の新しい言語パックをインストールすると、[プロジェクト プロパティ] ダイアログにインテル® C++ コンパイラー固有のオプションが表示されなくなることがあります。その場合には、以下の手順を試してみてください。

1) “<program files>\MSBuild\Microsoft.Cpp\v4.0\Platforms\[Win32|x64]\PlatformToolsets\Intel C++ Compiler XE 13.0\1033" ディレクトリーが存在する場合は、すべてのファイルを “<program files>\MSBuild\Microsoft.Cpp\v4.0\Platforms\[Win32|x64]\PlatformToolsets\Intel C++ Compiler XE 13.0\<locale-ID>” にコピーします。

2) "<program files>\MSBuild\Microsoft.Cpp\v4.0\Platforms\[Win32|x64]\PlatformToolsets\v100\1033\" が存在する場合は、すべてのファイルを “<program files>\MSBuild\Microsoft.Cpp\v4.0\Platforms\[Win32|x64]\PlatformToolsets\v100\<locale-ID>” にコピーします。

* <locale-ID> は言語パックを表します。

別の方法として、インテル® C++ Composer XE 2013 をアンインストールして、再インストールすることもできます。

3.5.4 インテル® Cilk™ Plus の既知の問題

- スチールが行われた後、対応する _Cilk_sync の前に SEH 例外がスローされると、

  Microsoft* C++ 構造化例外処理 (SEH) は失敗します。

3.5.5 ガイド付き自動並列化の既知の問題

プログラム全体のプロシージャー間の最適化 (/Qipo) が有効な場合、単一ファイル、関数名、ソースコードの指定範囲に対してガイド付き自動並列化 (GAP) 解析は行われません。この問題を回避するには、/Qipo を無効にします。Visual Studio* では、[プロジェクト] > [プロパティ ページ] > [C/C++] > [Optimization (最適化)] > [Interprocedural Optimization (プロシージャー間の最適化)] を「No (いいえ)」に設定します。

3.5.6 スタティック解析の既知の問題

3.5.6.1 仮想関数を含む C++ クラスに対する正しくないメッセージ

スタティック解析機能を使用するためには、インテル® Inspector XE も必要です。

プログラムで仮想関数を含む C++ クラスが使用されている場合に、スタティック解析は正しくない診断を多数出力します。場合によっては、診断結果の数が多すぎて結果ファイルが使用できないこともあります。

このような C++ ソース構造を使用しているアプリケーションでは、次のコマンドライン・オプションを追加することで不要なメッセージを表示しないようにできます: `/Qdiag-disable:12020,12040` (Windows*) または `-diag-disable 12020,12040` (Linux*)。**このオプションは、スタティック解析の結果が作成されるリンク時に追加する必要があります。**コンパイル時に追加しただけでは十分な効果が得られません。Microsoft* Visual Studio* では、このオプションを [プロパティ ページ] > [Linker (リンカー)] > [Command Line (コマンドライン)] に追加します。

ビルド仕様ファイルを使用してスタティック解析を行う場合は、`-disable-id 12020,12040` オプションを inspxe-runsc の呼び出しに追加します。
例:

```
inspxe-runsc -spec-file mybuildspec.spec -disable-id 12020,12040
```

この問題を含む作成済みのスタティック解析結果がある場合は、インテル® Inspector XE の GUI でそのファイルを開いて、次の手順に従って不要なメッセージを非表示にすることができます。

- 不要なメッセージは “`Arg count mismatch (引数の数の不一致)`” と “`Arg type mismatch (引数の型の不一致)`” です。それぞれの問題に対して、次の手順を実行します。
- 問題フィルターで不要な問題の種類をクリックします。これにより、それ以外の問題が非表示になります。
- 問題セットの表で任意の問題をクリックします。
- Ctrl+A キーを押すとすべての問題を選択できます。
- 右クリックしてポップアップ・メニューから *[Change State (ステートの変更)]* > *[Not a problem (問題なし)]* を選択し、不要なすべての問題のステートを設定します。
- 問題の種類のフィルターを [All (すべて)] に戻します。
- 他の不要な問題の種類に対して、上記の手順を行います。
- [Investigated/Not investigated (調査済み/未調査)] フィルターを *[Not investigated (未調査)]* に設定します。このフィルターは最後の方にあるため、フィルターペインを下にスクロールしないと見えないことがあります。[Not a problem (問題なし)] ステートは [Not investigated (未調査)] と見なされるため、これで不要なメッセージが非表示になります。

4 インテル® インテグレーテッド・パフォーマンス・プリミティブ

このセクションでは、インテル® インテグレーテッド・パフォーマンス・プリミティブ (インテル® IPP) のこのバージョンでの変更点、新機能、および最新情報をまとめています。インテル® IPP についての詳細は、次のリンクを参照してください。

- **新機能**: インテル® IPP 製品ページ (http://intel.ly/OG5IF7 (英語)) およびインテル® IPP リリースノート (http://intel.ly/OmWI4d (英語)) を参照してください。
- **ドキュメント、ヘルプ、サンプル**: インテル® IPP 製品ページ (http://intel.ly/OG5IF7 (英語)) のドキュメントのリンクを参照してください。

4.1 別途ダウンロード可能なインテル® IPP スタティック・スレッド・ライブラリー

インテル® IPP ライブラリーのスタティック・スレッド・バージョンはインテル® Composer XE パッケージに含まれなくなりました。これらのライブラリーは、インテル® Composer XE パッケージをダウンロードした場所から別途ダウンロードできます。

4.2 別途ダウンロード可能なインテル® IPP 暗号化ライブラリー

インテル® IPP 暗号化ライブラリーは別途ダウンロード可能です。ダウンロードとインストールの手順については、http://intel.ly/ndrGnR (英語) を参照してください。

4.3 インテル® IPP コードサンプル

インテル® IPP コードサンプルは、以下の Web サイトから入手できます。
http://intel.ly/pnsHxc (英語)

サンプルには、オーディオ/ビデオコーデック、画像処理、メディア・プレーヤー・アプリケーション、C++/C#/Java* からの呼び出し関数のソースコードが含まれています。サンプルのビルド方法についての説明は、各サンプルのインストール・パッケージの readme ファイルをご覧ください。

5 インテル® マス・カーネル・ライブラリー

このセクションでは、インテル® マス・カーネル・ライブラリー (インテル® MKL) の変更点、新機能、および最新情報をまとめています。問題の修正については、http://intel.ly/OeHQqf (英語) を参照してください。

5.1 注意事項

注意事項についての詳細は、削除された機能に関するナレッジベースの記事 (英語) を参照してください。

- Windows* の PATH 環境変数をインストール中に設定しないように変更
- インテル® MKL GNU Multiple Precision* (GMP) 関数インターフェイスを削除
- タイミング関数 mkl_set_cpu_frequency() を無効化 - インテル® MKL リファレンス・マニュアルで説明されているように、mkl_get_max_cpu_frequency()、mkl_get_clocks_frequency()、mkl_get_cpu_frequency() を使用してください。
- MKL_PARDISO 定数を削除 - mkl_domain_set_num_threads() 関数で PARDISO ドメインを指定する場合は MKL_DOMAIN_PARDISO を使用してください。
- インテル® MKL 10.2 Update 4 の畳み込み関数と相関関数の特別な後方互換関数を削除
- Windows* 版のインテル® MKL およびインテル® コンパイラーから OpenMP* スタティック・ランタイム・ライブラリーを削除
- インテル® Pentium® III プロセッサーのサポートが終了。サポートされる最小の命令セットはインテル® ストリーミング SIMD 拡張命令 2 (インテル® SSE2) になります。
- ドキュメント:
  - HTML 形式のインテル® MKL リファレンス・マニュアルの提供を終了

5.2 本バージョンでの変更

5.2.1 インテル® MKL 11.0 Update 3 の新機能

- BLAS:
  - Intel® Optimized MP LINPACK Benchmark for Clusters パッケージを HPL 2.1 に更新
- スパース BLAS:
  - インテル® アドバンスト・ベクトル・エクステンション 2 (インテル® AVX2) における DCOOMM のパフォーマンスが向上
- LAPACK:
  - ?LASET、?LACPY、?LANGE、?LANSY を並列化

- FFT:
  - インテル® AVX2 における複素数-複素数の 2 の累乗 FFT のパフォーマンスが向上
- VSL:
  - インテル® AVX2 における SFMT19937 基本乱数ジェネレーター (BRNG) のパフォーマンスが向上
- クラスター FFT:
  - ハイブリッド・モードのクラスター FFT のパフォーマンスが向上
- データ・フィッティング:
  - インテル® Xeon® プロセッサー X5570 およびインテル® Xeon® プロセッサー E5-2690 における線形および 3 次 Hermite/Bessel/Akima スプライン用 df?construct1d 関数のパフォーマンスが向上

5.2.2 インテル® MKL 11.0 Update 2 の新機能

- インテル® MKL Extended Eigensolver (拡張固有値ソルバー) を追加:

  インテル® MKL Extended Eigensolver は、密行列、LAPACK 帯行列、疎行列 (CSR) 形式の標準定値固有値問題および一般化定値固有値問題を解くハイパフォーマンスなパッケージです。Feast という革新的な高速かつ安定した数値アルゴリズムをベースにしています (「権利の帰属」セクションを参照)。
- スパース BLAS:
  - 0 ベースの DCSRMM のパフォーマンスが大幅に向上
- LAPACK:
  - x(OR/UN)(M/G)(LQ/QL/QR/RQ) 関数の並列バージョンのパフォーマンスが大幅に向上
- ScaLAPACK:
  - バージョン 2.0.2 にアップデート。新しい関数を追加:
    - PxHSEQR: 非対称固有値問題
    - PxSYEVR/PxHEEVR: MRRR (Multiple Relatively Robust Representations) アルゴリズム
- VSL:
  - VSL 対数正規乱数ジェネレーター (RNG) で累積分布逆関数 (ICDF) メソッドをサポート
  - 要約統計量関数の宣言に const 指定子を追加
- データ・フィッティング:
  - インテル® Xeon® プロセッサー X5570 およびインテル® Xeon® プロセッサー E5-2600 におけるスカラーケース (補間位置の数が 1) とベクトルケースのデフォルト/準一様分割、ソートされた補間位置用の df[d/s]Interpolate1D、df[d/s]InterpolateEx1D、df[d/s]SearchCells1D、df[d/s]SearchCellsEx1D ルーチンのパフォーマンスが向上
  - データ・フィッティング・ルーチンのパラメーターの正当性確認を無効にすることにより、補間位置の数が少ない (12 個未満) 場合のパフォーマンスを向上させる DF_DISABLE_CHECK_FLAG パラメーターを dfiEditVal エディターでサポート
  - 関数の宣言に const 指定子を追加
- 転置:
  - アウトオブプレース行列転置 (mkl_?omatcopy2) を並列化。パフォーマンスが大幅に向上

- サービス関数:
  - インテル® MKL メモリー・アロケーターで使用されるピークメモリーの量に関する情報を提供する mkl_peak_mem_usage 関数を追加
  - インテル® MKL メモリー・アロケーターの機能を標準 C ライブラリーのメモリー割り当て API に拡張する mkl_calloc 関数と mkl_realloc 関数を追加
- SMP LINPACK を拡張 (残差チェック):

  失敗が検出されるとエラーコード 1 を返し、生成された残差が精度チェックをパスするかどうか結論を出力します。条件付き数値再現性モードがオフの場合、同じ行列で実行ごとに残差が多少異なることがあるので、注意してください。残差チェックは結果が信頼できることを保証します。

5.2.3 インテル® MKL 11.0 Update 1 の新機能

- BLAS:
  - AMD* CPU (開発コード名: Bulldozer) における DGEMM および倍精度レベル 3 BLAS のパフォーマンスが向上
- PARDISO: エルミート行列の対数値の虚部を無視
- クラスター FFT:
  - ハイブリッド・クラスター FFT (MPI + OpenMP*) のパフォーマンスが向上 (最大 2 倍)
  - よりコミュニケーションの少ない新しいクラスター FFT アルゴリズムを 1D FFT 用に実装 - 有効にするには環境変数 MKL_CFFT_SOI_ENABLE を YES または 1 に設定 (詳細はインテル® MKL のドキュメントを参照)
- VSL:
  - ユーザーが定義した平均で記述統計の推定値を計算する VSL_SS_METHOD_FAST_USER_MEAN メソッドのサポートを追加
  - インテル® Xeon® プロセッサー E5-2690 において記述統計の推定値を計算する VSL_SS_METHOD_FAST メソッドのパフォーマンスが向上
- 転置: 第 2 世代インテル® Core™ マイクロアーキテクチャーにおけるアウトオブプレース転置のパフォーマンスが向上 (最大 7 倍)
- サービス関数: 古い名前の 7 つのサービス関数を削除 (詳細は削除された古いサービス関数についての記事を参照)
- 不具合の修正

5.2.4 最初のリリースでの変更

- 条件付きビット単位演算の再現性 (CBWR):コード分岐の選択により柔軟性を持たせ、アルゴリズムに決定性があることを保証して、パフォーマンスと再現性のある結果のバランスを取るインテル® MKL の新しい機能です。詳細は、『インテル® MKL ユーザーガイド』を参照してください。また、CBWR ナレッジベースの記事 (英語) も参照してください。
- インテル® MKL は、新しい FMA3 命令を含む、新しいインテル®アドバンスト・ベクトル・エクステンション 2 (インテル® AVX2) を使用した最適化もサポートします。詳細は、インテル® AVX2 のサポートに関するナレッジベースの記事 (英語) を参照してください。
- BLAS:
  - インテル® アドバンスト・ベクトル・エクステンション (インテル® AVX) 対応の 64 ビット・プログラムで DSYRK/SSYRK のパフォーマンスが向上
- LAPACK:
  - LAPACK バージョン 3.4.1 のサポートを追加

- FFT:
  - 記述子あたりのスレッド数を制限する構成パラメーター DFTI_THREAD_LIMIT を追加
  - 64 ビット整数で指定されたサイズの 1D 実数-複素数変換のサポートを追加
- VML /VSL:
  - インテル® アドバンスト・ベクトル・エクステンション (インテル® AVX) における viRngGeometric のパフォーマンスが向上
  - データ適合 Integrate1d 関数にスレッド化を実装
- 転置: パフォーマンスを大幅に向上させるために単精度と倍精度のリーディング・ディメンジョンが行列サイズより大きな正方行列のインプレース転置を並列化
- スレッド制御の柔軟性を高めるローカルスレッド制御関数 (mkl_set_num_threads_local) を実装
- リンク・ライン・アドバイザー:
  - アーキテクチャー (IA-32/インテル® 64) とインターフェイス・レイヤー (LP64/ILP64) を選択するアドバイザー機能を追加

5.3 権利の帰属

エンド・ユーザー・ソフトウェア使用許諾契約書 (End User License Agreement) で言及されているように、製品のドキュメントおよび Web サイトの両方で完全なインテル製品名の表示 (例えば、"インテル® マス・カーネル・ライブラリー") とインテル® MKL ホームページ (http://www.intel.com/software/products/mkl (英語)) へのリンク/URL の提供を正確に行うことが最低限必要です。

インテル® MKL の一部の基となった BLAS の原版は http://www.netlib.org/blas/index.html (英語) から、LAPACK の原版は http://www.netlib.org/lapack/index.html (英語) から入手できます。LAPACK の開発は、E. Anderson、Z. Bai、C. Bischof、S. Blackford、J. Demmel、J. Dongarra、J. Du Croz、A. Greenbaum、S. Hammarling、A. McKenney、D. Sorensen らによって行われました。LAPACK 用 FORTRAN 90/95 インターフェイスは、http://www.netlib.org/lapack95/index.html (英語) にある LAPACK95 パッケージと類似しています。すべてのインターフェイスは、純粋なプロシージャー用に提供されています。

インテル® MKL クラスター・エディションの一部の基となった ScaLAPACK の原版は http://www.netlib.org/scalapack/index.html (英語) から入手できます。ScaLAPACK の開発 は、L. S. Blackford、J. Choi、A. Cleary、E. D'Azevedo、J. Demmel、I. Dhillon、J. Dongarra、S. Hammarling、G. Henry、A. Petitet、K. Stanley、D. Walker、R. C. Whaley らによって行われました。

インテル® MKL Extended Eigensolver の機能は、Feast Eigenvalue Solver 2.0 (http://www.ecs.umass.edu/~polizzi/feast/) をベースにしています。

インテル® MKL の PARDISO は、バーゼル大学 (University of Basel) から無償で提供されている PARDISO 3.2 (http://www.pardiso-project.org (英語)) と互換性があります。

本リリースのインテル® MKL の一部の FFT 関数は、カーネギーメロン大学からライセンスを受けて、SPIRAL ソフトウェア生成システム (http://www.spiral.net/ (英語)) によって生成されました。SPIRAL の開発は、Markus Püschel、José Moura、Jeremy Johnson、David Padua、Manuela Veloso、Bryan Singer、Jianxin Xiong、Franz Franchetti、Aca Gacic、Yevgen Voronenko、Kang Chen、Robert W. Johnson、Nick Rizzolo らによって行われました。

6 インテル® スレッディング・ビルディング・ブロック

本バージョンのインテル® スレッディング・ビルディング・ブロックの変更に関する詳細は、TBB ドキュメント・ディレクトリーの CHANGES というファイルを参照してください。

6.1 既知の問題

インテル® スレッディング・ビルディング・ブロックの本リリースに関する次の注意事項に留意してください。

6.1.1 ライブラリーの問題

- 同じプログラムで連続してインテル® TBB と OpenMP* コンストラクトをともに使用していて、OpenMP* コードにインテル® コンパイラーを使用している場合、KMP_BLOCKTIME に小さな値 (例えば、20 ミリ秒) を設定するとパフォーマンスが向上します。この設定は、kmp_set_blocktime() ライブラリー呼び出しを使用して OpenMP* コード内で行うこともできます。KMP_BLOCKTIME および kmp_set_blocktime() の詳細は、コンパイラーの OpenMP* に関するドキュメントを参照してください。
- 一般に、アプリケーションやサンプルの非デバッグ ("リリース") ビルドは、インテル® TBB ライブラリーの非デバッグバージョンとリンクし、デバッグビルドはインテル® TBB ライブラリーのデバッグバージョンとリンクします。Windows* システムでは、/MD オプションを使用してコンパイルした場合はインテル® TBB ライブラリーのリリース・ライブラリー、/MDd オプションを使用してコンパイルした場合はデバッグ・ライブラリーを使ってビルドしてください。他の組み合わせでは、ランタイムエラーが発生します。デバッグ・ライブラリーとリリース・ライブラリーの詳細については、製品の "Documentation" サブディレクトリーに含まれているチュートリアルを参照してください。

7 著作権と商標について

インテル・プロセッサー・ナンバーはパフォーマンスの指標ではありません。プロセッサー・ナンバーは同一プロセッサー・ファミリー内の製品の機能を区別します。異なるプロセッサー・ファミリー間の機能の区別には用いません。詳細については、http://www.intel.co.jp/jp/products/processor_number/ を参照してください。

Intel、インテル、Intel ロゴ、Intel Core、Itanium、Pentium、Xeon は、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。

* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。