インテル® C++ Composer XE 2011 Mac OS* X 版インストール・ガイドおよびリリースノート

資料番号: 321413-003JA
2010 年 11 月 8 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。

インテル® C++ Composer XE 2011 は、以前「インテル® C++ コンパイラー・プロフェッショナル・エディション」と呼ばれていた製品の最新バージョンです。

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。

これは最初の製品リリースです。

1.2 製品の内容

インテル® C++ Composer XE 2011 Mac OS X 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 12.0。Mac OS* X オペレーティング・システムを実行するインテル® プロセッサー・ベースの Mac 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® デバッガー
- 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 7.0 Update 1
- インテル® マス・カーネル・ライブラリー 10.3
- インテル® スレッディング・ビルディング・ブロック 3.0 Update 3
- Xcode* 開発環境への統合
- 各種ドキュメント

1.3 動作環境

- インテル® プロセッサー・ベースの Apple* Mac* システム
- 1GB RAM (最小)、2GB RAM (推奨)
- 3GB のディスク空き容量
- Mac OS* X、Xcode*、Xcode SDK の次のいずれかの組み合わせ:
  - OS X 10.6.3、Xcode 3.2.2、SDK 10.6 または 10.5
  - OS X 10.5.8、Xcode 3.1.4、SDK 10.6 または 10.5
- gcc* 4

注: 高度な最適化オプションを利用した場合、またはプログラムが非常に大きい場合、メモリーやディスク容量がさらに必要になることがあります。

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、関連ライブラリーおよび関連開発ツールには、インテル製マイクロプロセッサーおよび互換マイクロプロセッサーで利用可能な命令セット (SIMD 命令セットなど) 向けの最適化オプションが含まれているか、あるいはオプションを利用している可能性がありますが、両者では結果が異なります。また、インテル® コンパイラー用の特定のコンパイラー・オプション (インテル® マイクロアーキテクチャーに非固有のオプションを含む) 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。これらのコンパイラー・オプションと関連する命令セットおよび特定のマイクロプロセッサーの詳細は、『インテル® コンパイラー・ユーザー・リファレンス・ガイド』の「コンパイラー・オプション」を参照してください。インテル® コンパイラー製品のライブラリー・ルーチンの多くは、互換マイクロプロセッサーよりもインテル製マイクロプロセッサーでより高度に最適化されます。インテル® コンパイラー製品のコンパイラーとライブラリーは、選択されたオプション、コード、およびその他の要因に基づいてインテル製マイクロプロセッサーおよび互換マイクロプロセッサー向けに最適化されますが、インテル製マイクロプロセッサーにおいてより優れたパフォーマンスが得られる傾向にあります。

インテル® コンパイラー、関連ライブラリーおよび関連開発ツールは、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 (インテル® 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。

インテルでは、インテル® コンパイラーおよびライブラリーがインテル製マイクロプロセッサーおよび互換マイクロプロセッサーにおいて、優れたパフォーマンスを引き出すのに役立つ選択肢であると信じておりますが、お客様の要件に最適なコンパイラーを選択いただくよう、他のコンパイラーの評価を行うことを推奨しています。インテルでは、あらゆるコンパイラーやライブラリーで優れたパフォーマンスが引き出され、お客様のビジネスの成功のお役に立ちたいと願っております。お気づきの点がございましたら、お知らせください。

改訂 #20101101

1.5 テクニカルサポート

インストール時にコンパイラー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

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。

Xcode を使用する場合、サポートされているバージョンの Xcode がインストールされていることを確認してください。将来、新しいバージョンの Xcode をインストールする場合は、そのインストール後にインテル® C++ コンパイラーを再インストールする必要があります。

製品のインストール、変更、アンインストールには管理者権限または “sudo” 権限が必要です。

DVD 版を購入した場合は、DVD を挿入して、DVD のディスク・イメージ・ファイル (xxx.dmg) をダブルクリックします。ダウンロード版の場合は、ダウンロードしたファイルをダブルクリックします。

手順に従ってインストールを完了します。

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。

2.1 インテルのアクティベーション・ツールを使用した製品のアクティベーション

この製品リリースでは、新しいインテルのアクティベーション・ツール “ActivationTool” が /opt/intel/ActivationTool/Activation/ ディレクトリーにインストールされます。

インストール中に評価用ライセンスまたは評価用シリアル番号を使用したり、あるいは [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製品をインストールした場合、製品を購入した後にこのアクティベーション・ツール (/opt/intel/composerxe-2011.x.xxx/Activation/ActivationTool) を使用して製品をアクティベートできます。これにより、評価版から製品版へ移行することができます。このツールを使用するには、次のコマンドを実行します。

```
$ /opt/intel/composerxe-2011.x.xxx/Activation/ActivationTool
[シリアル番号]
```

2.2 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://software.intel.com/en-us/articles/licensing-setting-up-the-client-floating-license/ (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。

2.3 インストール先フォルダー

コンパイラーは、デフォルトでは /opt/intel にインストールされます。本リリースノートでは、この場所を <install-dir> と表記します。別のインストール場所を指定することもでき

ます。Xcode* 統合がインストールされると、これらのファイルの 2 つめのコピーが /Developer/opt/intel 以下に置かれます。

本リリースではディレクトリー構成が インテル® コンパイラー 11.1 から変更されています。

<install-dir> 以下には次のサブディレクトリーがあります。

- bin – インストールされている最新バージョンの実行ファイルへのシンボリック・リンク
- lib – インストールされている最新バージョンの lib ディレクトリーへのシンボリック・リンク
- include – インストールされている最新バージョンの include ディレクトリーへのシンボリック・リンク
- man – インストールされている最新バージョンの man ページが含まれているディレクトリーへのシンボリック・リンク
- ipp – インストールされている最新バージョンのインテル® インテグレーテッド・パフォーマンス・プリミティブのディレクトリーへのシンボリック・リンク
- mkl – インストールされている最新バージョンのインテル® マス・カーネル・ライブラリーのディレクトリーへのシンボリック・リンク
- tbb – インストールされている最新バージョンのインテル® スレッディング・ビルディング・ブロックのディレクトリーへのシンボリック・リンク
- composerxe – composerxe-2011 ディレクトリーへのシンボリック・リンク
- composerxe-2011 – インストールされている最新バージョンのインテル® Composer XE コンパイラーのサブディレクトリーへのシンボリック・リンク
- composerxe-2011-<n>.<pkg> – 特定のコンパイラー・バージョンのファイルが含まれている物理ディレクトリー。<n> はリビジョン番号、<pkg> はパッケージビルド ID。

各 composerxe-2011 ディレクトリーには、インストールされている最新のインテル® Composer XE 2011 コンパイラーを参照する次のサブディレクトリーが含まれています。

- bin – コンパイラー環境とホスト環境用のコンパイラー実行ファイルへのシンボリック・リンクを設定するためのスクリプト
- pkg_bin – コンパイラーの bin ディレクトリーへのシンボリック・リンク
- include – コンパイラーの include ディレクトリーへのシンボリック・リンク
- lib – コンパイラーの lib ディレクトリーへのシンボリック・リンク
- ipp – ipp ディレクトリーへのシンボリック・リンク
- mkl – mkl ディレクトリーへのシンボリック・リンク
- tbb – tbb ディレクトリーへのシンボリック・リンク
- debugger – debugger ディレクトリーへのシンボリック・リンク
- man – man ディレクトリーへのシンボリック・リンク
- Documentation – Documentation ディレクトリーへのシンボリック・リンク
- Samples – Samples ディレクトリーへのシンボリック・リンク

各 composerxe-2011-<n>.<pkg> ディレクトリーには、特定のリビジョン番号のインテル® Composer XE 2011 コンパイラーを参照する次のサブディレクトリーが含まれています。

- bin – すべての実行ファイル
- compiler – 共有ライブラリーとヘッダーファイル
- debugger – デバッガーファイル
- Documentation – ドキュメント・ファイル
- man – man ディレクトリーへのシンボリック・リンク

- ipp - 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とヘッダーファイル
- mkl - インテル® マス・カーネル・ライブラリーのライブラリーとヘッダーファイル
- tbb - インテル® スレッディング・ビルディング・ブロックのライブラリーとヘッダーファイル
- Samples - サンプルプログラムとチュートリアル・ファイル

インテル® C++ コンパイラーとインテル® Fortran コンパイラーの両方がインストールされている場合、所定のバージョンおよびリビジョン番号のフォルダーが共有されます。

このディレクトリー構成により、任意のバージョン/リビジョン番号のインテル® Composer XE 2011 コンパイラーを選択することができます。<install-dir>/bin にある compilervars.sh [.csh] スクリプトを参照すると、インストールされている最新のコンパイラーが使用されます。このディレクトリー構成は、将来のリリースでも保持される予定です。

2.4 インテル® インテグレーテッド・パフォーマンス・プリミティブ暗号化ライブラリーのインストール

インテル® IPP は、暗号化ルーチンのライブラリーを含むオプションのコンポーネントを提供します。暗号化ライブラリーのインストールと使用には、別途ライセンスが必要です。このライセンスは、インテル® IPP のライセンスの登録後に無償で入手できます。ただし、輸出規制が適用されます。詳細は、http://software.intel.com/en-us/articles/intel-integrated-performance-primitives-cryptography-library/ (英語) を参照してください。

2.5 インストール後の製品の配置変更

Xcode 統合は、Xcode ディレクトリー・ツリーを別の場所にドラッグアンドドロップするだけで移動できます。移動した Xcode ディレクトリー・ツリーを使用してコマンドプロンプトから idb を使用する場合は、http://software.intel.com/en-us/articles/running-idb-from-command-line-after-relocating-xcode-environment/ (英語) を参照して、必要なその他の手順を確認してください。idb は Xcode* IDE 内では利用できないことに注意してください。

2.6 削除/アンインストール

インストールされているパフォーマンス・ライブラリー・コンポーネントを残したまま、コンパイラーのみを削除することはできません。

1. 端末を開いて、<install-dir> 以外のフォルダーに移動 (cd) します。
2. その後、次のコマンドを使用します。
   <install-dir>/compilerpro-12.0.<n>.<pkg>/uninstall_cproc.sh
3. 画面の指示に従ってオプションを選択します。

root でログインしていない場合は root パスワードの入力が求められます。

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 新機能と変更された機能

詳細は、コンパイラーのドキュメントを参照してください。

- C++0x からの機能
  - 右辺値参照
  - 標準的なアトミック演算
  - “Windows* C++” モードでの C99 の 16 進浮動小数点定数のサポート
  - 右角括弧
  - 拡張 friend 宣言
  - 混在した文字列リテラルの結合
  - `long long` のサポート
  - 可変引数マクロ
  - スタティック・アサーション
  - auto 型変数
  - extern テンプレート
  - `__func__` 事前定義済み指定子
  - 式の型宣言 (decltype)
  - ユニバーサル文字名
  - 強い型付けの列挙型
  - ラムダ
- より高速でやや精度が低い算術ライブラリー関数を使用するためのオプション
- プロセッサーのモデルや製造元に関係なく一貫した結果を返す算術ライブラリー関数を使用するためのオプション

3.2 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。

- -ansi-alias-check
- -ffriend-injection
- -fzero-initialized-in-bss
- -fimf-absolute-error
- -fimf-accuracy-bits
- -fimf-arch-consistency
- -fimf-max-error
- -fimf-precision
- -fms-dialect
- -fp-trap
- -fp-trap-all
- -fvar-tracking
- -fvar-tracking-assignments
- -opt-args-in-regs
- -prof-value-profiling
- -profile-functions
- -profile-loops
- -regcall
- -simd
- -Wremarks

- -Wsign-compare
- -Wstrict-aliasing

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 その他の変更

3.3.1 最適化レポートがデフォルトで無効に設定

バージョン 11.1 以降、コンパイラーは、ベクトル化、自動並列化、OpenMP* スレッド化ループに関する最適化レポートメッセージをデフォルトで表示しないようになりました。これらのメッセージを表示するには、`-diag-enable vec`、`-diag-enable par`、`-diag-enable openmp` を設定するか、`-vec-report`、`-par-report`、`-openmp-report` を使用する必要があります。

また、バージョン 11.1 以降、最適化レポートメッセージは `stdout` ではなく、`stderr` に送られます。

3.3.2 環境設定スクリプトの変更

コンパイラー環境は、`compilervars.sh` スクリプトを使用して設定します。

コマンドの形式は以下のとおりです。

`source <install-dir>/bin/compilervars.sh` *`argument`*

`argument` にはターゲット・アーキテクチャーに応じて、`ia32` または `intel64` を指定します。コンパイラー環境を設定すると、インテル® デバッガー、インテル® パフォーマンス・ライブラリー、インテル® Fortran コンパイラー (インストールされている場合) の環境も設定されます。

3.3.3 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 既知の問題

3.4.1 インテル® C++ コンパイラーと XCode* 3.2.2 でビルドした際に 32 ビット・カーネルで起こる 64 ビット実行ファイルのランタイムクラッシュ

32 ビット・カーネル上で 64 ビット実行ファイルのランタイムクラッシュが発生するというインテル® コンパイラーと XCode* 3.2.2 のリンカーの既知の問題があります。

1 つのパターンとして、コードに 5 つ以上の `case` を持つ `switch` 文があると、これが発生します。この場合、インテル® C++ コンパイラーは、"L" から始まるシンボルを生成します。このようなシンボルは、XCode* 3.2.2 のリンカーでは正しく解決されません。

この問題を回避する簡単な方法があります。詳細は、http://software.intel.com/en-us/articles/intel-compiler-and-xcode-322-linker-runtime-crash-with-switch-statement/ (英語) を参照してください。

3.4.2 __GXX_EXPERIMENTAL_CXX0X__ マクロの未サポート

GNU* 4.3 以降の環境で -std=c++0x または -std=gnu++0x オプションを使用すると、次のような診断が出力されることがあります。

```
This file requires compiler and library support for the upcoming ISO
C++ standard, C++0x. This support is currently experimental, and must
be enabled with the -std=c++0x or -std=gnu++0x compiler options. (この
ファイルには、新しい ISO C++ 規格である C++0x 用のコンパイラーとライブラリーのサ
ポートが必要です。現在このサポートはまだ試験段階であり、-std=c++0x または
-std=gnu++0x オプションと一緒に指定する必要があります。)
```

__GXX_EXPERIMENTAL_CXX0X__ マクロは、C++ 標準ライブラリー・ヘッダーのマクロで有効にされる一部の C++0x 機能 (可変個引数テンプレートなど) をまだサポートしていないため、インテル® コンパイラーでは、現在どのモードでもこのマクロを定義していません。このため、-std=c++0x または -std=gnu++0x モードで C++ 標準ライブラリーを使用すると、g++ との互換性問題が発生することがあります。例えば、va_copy マクロが stdarg.h で定義されていない可能性があります。この問題は、-Dva_copy=__builtin_va_copy コンパイラー・フラグを追加することで回避できます。

4 インテル® デバッガー (IDB)

4.1 コンパイル要件

Xcode 2.3 より、Dwarf デバッグ情報はオブジェクト (.o) ファイルに保存されています。これらのオブジェクト・ファイルは、デバッグ対象のアプリケーションに関連した情報を得るためにデバッガーによりアクセスされます。そのため、シンボリック・デバッグが利用可能でなければなりません。

次のように、1 つのコマンドでプログラムがコンパイルされ、リンクされた場合、

```
icc -g -o hello.exe hello.c
```

コンパイラーによりオブジェクト・ファイルは生成されますが、コマンドが完了する前に削除されます。このコマンドで作成されたバイナリーファイルにはデバッグ情報は含まれません。アプリケーションをデバッグ可能にするには、次の 2 つの方法があります。

アプリケーションを 2 つの手順でビルドして .o ファイルを明示的に作成します。

```
icc -c -g -o hello.o hello.c

icc -g -o hello.exe hello.o
```

または、-save-temps コンパイラー・スイッチを使用して作成された .o ファイルが削除されないようにします。

```
icc -g -save-temps -o hello.exe hello.c
```

デバッガーは "dsymutil" ユーティリティーの出力を使用しません。

4.2 既知の問題

4.2.1 Dwarf と Stabs デバッグ・フォーマット

デバッガーでは、デバッグ情報が Dwarf2 フォーマットの実行ファイルのデバッグのみをサポートしており、Stabs デバッグ・フォーマットはサポートしていません。gcc と g++ で Dwarf 出力を生成するには、コンパイルコマンドで -gdwarf-2 フラグを使用します。インテル® コンパイラー (icc と ifort) では、-g フラグで Dwarf2 デバッグ・フォーマットを作成します。

4.2.2 共有ライブラリーのデバッグ情報

デバッガーは共有ライブラリーのデバッグ情報を読みません。そのため、システム・ライブラリーの一部である _exit のようなシンボルへブレークポイントを設定できません。

4.2.3 非ローカルのバイナリーファイルとソースファイルのアクセス

デバッガーは、ネットワーク・マウント・ファイル・システム (NFS など) からバイナリーファイルにアクセスできません。次のようなエラーメッセージが表示されます。

```
Internal error: cannot create absolute path for: /home/me/hello
(内部エラー: /home/me/hello の絶対パスを作成できません。)
You cannot debug "/home/me/hello" because its type is "unknown".
("/home/me/hello" はデバッグできません。型が "不明" です。)
```

また、デバッガーは、ネットワーク・マウント・ファイル・システム (NFS など) からソースファイルにアクセスできません。次のようなエラーメッセージが表示されます。

```
Source file not found or not readable, tried... (ソースファイルが見つからないか読み取りできません...)
./hello.c
/auto/mount/site/foo/usr1/user_me/c_code/hello.c
(Cannot find source file hello.c (ソースファイル hello.c が見つかりません。))
The file-path specified will be correct. (指定されたファイルパスは修正されます。)
```

ファイルは、ローカル・ファイル・システム (例: ネットワークでマウントされていないシステム) にコピーして使用してください。

4.2.4 fork アプリケーションのデバッグ

fork を呼び出すアプリケーションの子プロセスのデバッグはまだサポートされていません。

4.2.5 exec アプリケーションのデバッグ

$catchexecs 制御変数はサポートされていません。

4.2.6 スナップショット

マニュアルで説明されているスナップショットはまだサポートされていません。

4.2.7 最適化コードのデバッグ

最適化コードのデバッグはまだ完全にはサポートされていません。最適化を有効にしてコードをコンパイルすると、一部の関数名、パラメーター、変数、パラメーターと変数の内容をデバッガーが参照できないことがあります。

4.2.8 ウォッチポイント

書き込みアクセスを検知するよう作成されたウォッチポイントは、元の値と同一の値が書き込まれたときにはトリガーしません。これは、Mac OS* X オペレーティング・システムの制限によるものです。

ウォッチポイントの実装には、SIGSEGV シグナルではなく SIGBUS シグナルがデバッガーで使用されているため、SIGBUS シグナルをキャッチするシグナル・ディテクターを作成することができません。

4.2.9 グラフィック・ユーザー・インターフェイス (GUI)

本バージョンのデバッガーでは GUI はサポートされていません。

4.2.10 MPP デバッグの制限

マニュアルで説明されている MPP デバッグはサポートされていません。

4.2.11 関数ブレークポイント

関数に設定されたブレークポイント ("`stop in`" コマンドを使用して設定) では、最初の文でユーザープログラムの実行が停止されないことがあります。これは、生成された Dwarf デバッグ情報で関数プロローグに関する情報が不十分なために発生します。回避策として、"`stop at`" コマンドで該当文にブレークポイントを設定します。

コンパイラーは "`__dyld_func_lookup`" への呼び出しを関数のプロローグの一部として作成します。この関数にブレークポイントを設定すると、デバッガーはその位置で停止しますが、ローカル変数値が有効ではありません。回避策として、ブレークポイントを関数内の最初の文に設定します。

4.2.12 コアファイルのデバッグ

コアファイルのデバッグは、サポートされていません。

4.2.13 ユニバーサル・バイナリーのサポート

ユニバーサル・バイナリーのデバッグはサポートされています。デバッガーは IA-32 上の IA-32 Dwarf セクションのバイナリーと、インテル® 64 上の IA-32 セクションまたはインテル® 64 セクションのデバッグをサポートしています。

4.2.14 `$threadlevel` デバッガー変数

マニュアルでは、"`$threadlevel`" デバッガー変数について「On Mac OS* X, the debugger supports POSIX threads, also known as pthreads. (Mac OS* X では、デバッガーは POSIX スレッド (pthreads とも呼ばれる) をサポートしています。)」という記述があります。この文章では別の種類のスレッドもサポートされているようにとれますが、そうではなく、POSIX スレッドのみが Mac OS* X 上でサポートされています。

4.2.15 オープンファイル記述子の制限

デバッガーはデバッグ対象の .o ファイルを開いてデバッグ情報を読み取るため、ファイルの制限を緩和する必要があります。

Mac OS* では、オープンできるファイル記述子の数を 256 に制限していますが、次のように上限を上げることができます。

```
ulimit -n 2000
```

デバッガーを起動する前に、このコマンドを使用してオープンファイル記述子の数の制限を上げてください。

これは、デバッガーが多くのファイルに対してオープンファイル記述子の制限数を適切に共有できるようになるまでの回避策です。

4.2.16 $cdir ディレクトリー、$cwd ディレクトリー

$cdir はコンパイル・ディレクトリーです (記録されている場合)。$cdir は、ディレクトリーが設定されている場合にサポートされます。シンボルとしてサポートされるわけではありません。

$cwd は現在の作業ディレクトリーです。セマンティクスもシンボルもサポートされていません。

$cwd と '.' の違いは、$cwd はデバッグセッション中に変更された現在の作業ディレクトリーを追跡する点です。'.' は、ソースパスへのエントリーが追加されると直ちに現在のディレクトリーに展開されます。

4.2.17 info stack の使用

デバッガーコマンド info stack は、以下のオプションの構文では現在、負のフレームカウントをサポートしていません。

```
info stack [num]
```

フレームカウント num が正の場合、最内 num フレームを出力します。カウントが負またはゼロの場合、(最外 num フレームを出力するのではなく) フレームを出力しません。

4.2.18 $stepg0 のデフォルト値の変更

デバッガー変数 $stepg0 のデフォルト値が 0 に変更されました。値 "0" の設定では、"step" コマンドを使用する場合、デバッガーはデバッグ情報なしでコードにステップオーバーします。以前のデバッガーバージョンと互換性を保つようにするには、次のようにデバッガー変数を 1 に設定します。

```
(idb) set $stepg0 = 1
```

5 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://software.intel.com/en-us/intel-ipp (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://software.intel.com/en-us/intel-ipp (英語)) のドキュメントのリンクを参照してください。

5.1 新機能と変更された機能

- JPEG-XR (HD Photo) コーデックが、ピクセルの深さが 8、16、32 ビット整数および 16、32 ビット浮動小数点数のグレースケール、RGB、RGBA イメージ向け IPP UIC サンプル・フレームワークに追加されました。
- 新しく `interfaces` ディレクトリーが追加されました。このディレクトリーには、ハイレベルのアプリケーション・コードのソースバイナリーとビルド前のバイナリーが含まれます。いくつかのよく使用されるデータ圧縮ライブラリー (bzip2、zlib and gzip など) が IPP ライブラリーでも使用できるように変更され、`interfaces` ディレクトリーに格納されています。
- 本リリースの一部として、新しい ipp_lzopack (データ圧縮) ライブラリーが `interfaces` ディレクトリーにあります。
- 256 ビットの AVX SIMD 命令セット向けの最適化拡張 (開発コード名が “Sandy Bridge” というインテル® プロセッサーで利用可能) が組み込まれています。
- 暗号化ドメイン (別途ダウンロードが必要。以下を参照。) とデータ圧縮 (ipp_bzip2 向けの CRC32) においてより多くの AES-NI 最適化が適用されており、AES-NI 命令をサポートしているプロセッサーでパフォーマンスが大幅に向上します。
- OpenMP* マルチスレッド・ライブラリーの使用により、ipp_zlib ライブラリーの一部がマルチスレッドに対応しました。
- 新しいディレクトリー構造により、インテル® IPP ライブラリーとインテル® コンパイラー製品の統合が単純化されました。この変更に伴い、ビルドスクリプトや makefile の更新が必要になることがあります。
- これまでの “`em64t`” ディレクトリーが “`intel64`”ディレクトリーに変更されました。この変更に伴い、ビルドスクリプトや makefile の更新が必要になることがあります。
- 32 ビットと 64 ビットのアーキテクチャー間で一貫性を保持するために、ライブラリー・ファイル名が正規化されました (例えば、すべての 64 ビットのライブラリー・ファイル名から “`em64t`” が削除されました)。この変更に伴い、ビルドスクリプトの更新が必要になることがあります。
- ドメイン固有の “emerged”および “merged” スタティック・ライブラリー・ファイルは、参照を容易にするために 1 つにまとめられ (例: ippsemerged.lib + ippsmerged_t.lib ⇒ ipps_t.lib)、シングル・スレッド・スタティック・ライブラリーのサフィックスは “_l” に変更されました (マルチスレッド・スタティック・ライブラリーのサフィックスはこれまでと同様に “_t” です)。この変更に伴い、ビルドスクリプトや makefile の更新が必要になることがあります。
- 16s、32s、32f データ型の JPEG-XR (HD Photo) の正変換/逆変換のサポート、および 32s データ型の可変長符号 (VLC) のエンコード/デコード関数のサポートが追加されました。
- 本リリースには音声認識関数 (ippSR ドメイン) は含まれていません。このドメインは IPP 6.1 製品で継続してサポートされます。

- SPIRAL 生成関数 (ippGEN ドメイン) は、別途ダウンロードにて配布されるようになりました。詳細は、下記の手順を参照してください。

5.2 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://software.intel.com/en-us/articles/download-ipp-cryptography-libraries/ (英語) を参照してください。

5.3 別途ダウンロード可能なインテル® IPP SPIRAL ドメイン (ippGEN)

IPP ライブラリーのインストール・パッケージのサイズを縮小するために、SPIRAL ドメイン (ippGEN) はライブラリー・アドオンとして別途配布されるようになりました。IPP ライブラリーの ippGEN コンポーネントは 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 からダウンロードしてください。

IPP の SPIRAL は、ippGEN ドメインの関数を活用するのに必要なバイナリーとヘッダーファイルが含まれたインストール・パッケージです。SPIRAL は IPP ライブラリーの*アドオン*なので、*システムにコア IPP ライブラリーがインストールされていなければなりません。*

IPP ライブラリーをインストールしてから SPIRAL アドオン・ライブラリーをインストールしてください。

5.4 インテル® IPP コードサンプル

インテル® IPP コードサンプルとして、Windows* 版、Linux* 版、Mac OS* 版のダウンロード・パッケージが用意されています。以下の Web サイトから入手できます。
http://www.intel.com/software/products/ipp (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。

6 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・ライブラリーの変更点、新機能、および最新情報をまとめています。

6.1 本バージョンでの変更

1) BLAS
   - 一度に 2 つの行列-ベクトル積を計算するための新しい関数: [D/S]GEM2VU、[Z/C]GEM2VC
   - 混合精度の一般的な行列-ベクトル積を計算するための新しい関数: [DZ/SC]GEMV
   - 2 つのスケールされたベクトルの和を計算するための新しい関数: *AXPBY
   - 主要関数においてインテル® AVX による最適化: SMP LINPACK、レベル 3 BLAS、DDOT、DAXPY
2) LAPACK
   - 行優先順に対応した LAPACK 用の C インターフェイス

- 1 つの新しい計算ルーチン (*GEQRFP)、2 つの新しい補助ルーチン (*GEQR2P と *LARFGP)、LAPACK 3.2.1 のアップデートを含む Netlib LAPACK 3.2.2 との統合
- 主要関数においてインテル® AVX による最適化: DGETRF、DPOTRF、DGEQRF

3) PARDISO
   - マルチコア環境で問題と解のステップのパフォーマンスが向上
   - スパースの右辺の解算出と部分解ベクトルを出力する部分解算出の追加
   - アウトオブコア (OOC) 因数分解のパフォーマンスが向上
   - ゼロベース (C スタイル) の配列インデックスのサポート
   - 対称行列のスパースデータ構造で行列の対角上のゼロが不要
   - 新しい ILP64 PARDISO インターフェイスにより、LP64 ライブラリーにリンクされている場合に LP64 と ILP64 の両バージョンを使用可能
   - OOC モードでディスクにファイルを格納するのに必要なメモリーを並べ替え直後に予測可能
4) スパース BLAS
   - 形式変換関数ですべてのデータ型に対応 (単精度/倍精度の実数/複素数データ)、および関数の戻り値として並べ替えあり/並べ替えなし配列を使用可能
5) FFT
   - すべての 1D/2D/3D FFT においてインテル® AVX による最適化
   - SSE4.2 命令セットをサポートするすべてのシステムにおいて、基数が混在する単精度/倍精度データの 2D/3D FFT のパフォーマンスが向上
   - 2D/3D FFT における 2 つの実数配列として表される分割複素数データのサポート
   - 長さが大きな素数である 1D 複素数-複素数変換のサポート
6) VML
   - (ax+b)/(cy+d) の計算を行うための新しい関数。a、b、c、d はスカラー、x、y は実数ベクトル: v[s/d]LinearFrac()
   - 主要関数においてインテル® AVX による最適化
   - デノーマル数をゼロに設定するための新しいモデル、複素ベクトルのオーバーフロー・サポート、各 VML 関数に対して精度を設定するための追加パラメーターを含む新しい関数
7) VSL
   - 新しいサマリー統計関数群。基礎統計、共分散/相関関係、プールされたグループ/部分/厳密な共分散/相関関係、分位数/変量分位数、外れ値検出アルゴリズム、欠測値をサポート
     - パフォーマンスが最適化されたアルゴリズム: 欠測値をサポートするための MI アルゴリズム、厳密な共分散を計算するための TBS アルゴリズム、外れ値を検出するための BACON アルゴリズム、(変量データの) 分位数を計算するための ZW アルゴリズム、プールされた共分散を計算するための 1PASS アルゴリズム
   - SFMT19937 基本乱数ジェネレーター (BRNG) のパフォーマンスが向上
   - インテル® AVX による最適化: MT19937 と MT2203 BRNG
8) ランタイムにディスパッチされるダイナミック・ライブラリーの追加により、ランタイムに検出された CPU またはライブラリー関数呼び出しに応じて、依存性のあるライブラリーを動的にロードする単一のインターフェイス・ライブラリーへのリンクが可能
9) カスタム・ダイナミック・ライブラリー・ビルダーは、Linux* および Mac OS* X オペレーティング・システムにおいてランタイムにディスパッチされるライブラリーを使用
10) 新しいディレクトリー構造により、インテル® MKL ライブラリーとインテル® Parallel Studio XE 製品ファミリーの統合が単純化され、これまでの "em64t" ディレクトリーが "intel64" ディレクトリーに変更
11) スパースソルバー機能をインテル® MKL のコア・ライブラリーに完全統合。また名前に "solver" を含むライブラリーを製品から削除

6.2 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 (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 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。本リリースのインテル® MKL の一部の FFT 関数は、ヒューストン大学からライセンスを受けて、UHFFT ソフトウェア生成システム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。

7 インテル® スレッディング・ビルディング・ブロック

インテル® スレッディング・ビルディング・ブロック (インテル® TBB) の変更に関する詳細は、インテル® 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。

8 著作権と商標について

本資料に掲載されている情報は、インテル製品の概要説明を目的としたものです。本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスを許諾するものではありません。製品に付属の売買契約書『Intel's Terms and Conditions of Sale』に規定されている場合を除き、インテルはいかなる責任を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証 (特定目的への適合性、商適格性、あらゆる特許権、著作権、その他知的財産権の非侵害性への保証を含む)に関してもいかなる責任も負いません。インテルによる書面での合意がない限り、インテル製品は、その欠陥や故障によって人身事故が発生するようなアプリケーションでの使用を想定した設計は行われていません。

インテル製品は、予告なく仕様や説明が変更されることがあります。機能または命令の一覧で「留保」または「未定義」と記されているものがありますが、その「機能が存在しない」あるいは「性質が留保付である」という状態を設計の前提にしないでください。これらの項目は、インテルが将来のために留保しているものです。インテルが将来これらの項目を定義したことにより、衝突が生じたり互換性が失われたりしても、インテルは一切責任を負いません。この情報は予告なく変更されることがあります。この情報だけに基づいて設計を最終的なものとしないでください。

本書で説明されている製品には、エラッタと呼ばれる設計上の不具合が含まれている可能性があり、公表されている仕様とは異なる動作をする場合があります。現在確認済みのエラッタについては、インテルまでお問い合わせください。

最新の仕様をご希望の場合や製品をご注文の場合は、お近くのインテルの営業所または販売代理店にお問い合わせください。

本書で紹介されている注文番号付きのドキュメントや、インテルのその他の資料を入手するには、1-800-548-4725 (アメリカ合衆国) までご連絡いただくか、インテルの Web サイトを参照してください。

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 ロゴは、アメリカ合衆国および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。