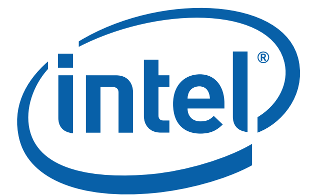


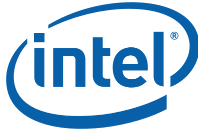
インテル® インテグレーテッド・パフォーマンス・プリミティブ Linux* 版 (IA-32 アーキテクチャー用)

ユーザーガイド

2008 年 9 月

資料番号 : 320271-001JP

World Wide Web: http://developer.intel.com



バージョン	バージョン情報	日付
-001	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) Linux* 版 (IA-32 アーキテクチャー用) ユーザーガイド。インテル ® IPP 6.0 リリースのドキュメント。	2008 年 9 月

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ための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、商品性に関する保証、第三者の特許権、著作権、その他、知的所有権を侵害していないことへの保証を含む）にも一切応じないものとします。インテル製品は、医療、救命、延命措置、重要な制御または安全システム、核施設などの目的に使用することを前提としたものではありません。

インテル製品は、予告なく仕様や説明が変更される場合があります。

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。

本資料で説明されているソフトウェアには、不具合が含まれている可能性があり、公開されている仕様とは異なる動作をする場合があります。現在までに判明している不具合の情報については、インテルのサポートサイトをご覧ください。

本資料およびこれに記載されているソフトウェアはライセンス契約に基づいて提供されるものであり、その使用および複製はライセンス契約で定められた条件下でのみ許可されます。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あり、インテルによる確約と解釈されるべきものではありません。本資料で提供される情報は、予告なく変更されることがあります。インテルは本資料の内容およびこれに関連して提供されるソフトウェアにエラー、誤り、不正確な点が含まれていたとしても一切責任を負わないものとします。

ライセンス契約で許可されている場合を除き、インテルからの書面での承諾なく、本書のいかなる部分も複製したり、検索システムに保持したり、他の形式や媒体によって転送したりすることは禁じられています。

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。

Intel、インテル、Intel ロゴ、Centrino、Intel Atom、Intel Core、Itanium、MMX、Pentium、Xeon 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。

© 2008 Intel Corporation. 無断での引用、転載を禁じます。

目次

第 1 章　概要

テクニカルサポート 1-1
このドキュメントについて 1-1
　目的 1-2
　対象者 1-2
ドキュメントの構成 1-3
　表記規則 1-4

第 2 章　インテル ® IPP 入門

インストールの確認 2-1
バージョン情報の取得 2-1
アプリケーションのビルド 2-1
　環境変数の設定 2-2
　ヘッダーファイルのインクルード 2-2
　IPP 関数の呼び出し 2-2
インテル ® IPP の使用を開始する前に 2-2

第 3 章　インテル ® IPP の構造

ディレクトリー構造 3-1
ライブラリー 3-2
　インテル ® IPP 共有オブジェクト・ライブラリー（SO）の使用 3-2
　インテル ® IPP スタティック・ライブラリーの使用 3-3
ドキュメント・ディレクトリーの内容 3-3

第 4 章　開発環境の設定

Eclipse CDT でインテル ® IPP をリンクするための設定 4-1
　Eclipse CDT 4.0 の設定 4-1
　Eclipse CDT 3.x の設定 4-1

第 5 章　インテル ® IPP とアプリケーションのリンク

ディスパッチ 5-1
　プロセッサーのタイプの検出 5-1
リンク方法の選択 5-2
　ダイナミック・リンク 5-3

スタティック・リンク(ディスパッチあり) 5-3
スタティック・リンク(ディスパッチなし) 5-5
カスタム SO のビルド 5-6
インテル ® IPP のリンク方法の比較 5-7
アプリケーションで必要なインテル ® IPP ライブラリーの選択 5-7
ダイナミック・リンク 5-8
スタティック・リンク(ディスパッチあり) 5-9
定義域別のライブラリー依存関係(スタティック・リンクのみ) 5-9
リンク例 5-10

第 6 章 マルチスレッド・アプリケーションのサポート
インテル ® IPP スレッディングと OpenMP* のサポート 6-1
スレッド数の設定 6-1
共有 L2 キャッシュの使用 6-1
入れ子の並列化 6-2
マルチスレッディングの無効化 6-2

第 7 章 パフォーマンスとメモリーの管理
メモリー・アライメント 7-1
しきい値 7-3
バッファーの再利用 7-4
FFT の使用 7-5
インテル ® IPP パフォーマンス・テスト・ツールの実行 7-6
パフォーマンス・テスト・ツールのコマンドラインの例 7-6

第 8 章 各種プログラミング言語でのインテル ® IPP の使用
言語サポート 8-1
Java アプリケーションでのインテル ® IPP の使用 8-1

付録 A パフォーマンス・テスト・ツールのコマンドライン・オプション

付録 B インテル ® IPP のサンプル
インテル ® IPP サンプルコードのタイプ B-1
インテル ® IPP サンプルのソースファイル B-1
インテル ® IPP サンプルの使用 B-3
動作環境 B-3
ソースコードのビルド B-3
ソフトウェアの実行 B-4
既知の制限事項 B-4

索引

概要

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（インテル ® IPP）は、一般的な信号、イメージ、音声、画像、データ圧縮、暗号化、テキスト文字列とオーディオ処理、ベクトル操作と行列演算を含む広範囲な機能を提供するソフトウェア・ライブラリーです。MP3（MPEG-1 オーディオ、レイヤー 3）、MPEG-4、H.264、H.263、JPEG、JPEG2000、GSM-AMR* および G723 のようなオーディオ、ビデオ、音声コーデック用の高度なプリミティブに加えて、コンピューター・ビジョンもカバーします。

インテル ® IPP ライブラリーは、各関数でさまざまなデータタイプとレイアウトをサポートし、使用されるデータ構造の数を最小化することで、開発者がアプリケーションのデザインと最適化に選択できる豊富なオプションのセットを提供します。データ構造を最小化しているインテル ® IPP ソフトウェアは、最適化されたアプリケーション、より高レベルのソフトウェア・コンポーネント、およびライブラリー関数を構築するための優れた柔軟性を備えています。

インテル ® IPP Linux* 版には複数のパッケージが含まれています。

- 32 ビット インテル ® アーキテクチャー用開発パッケージ - インテル ® IPP Linux* 版（IA-32 アーキテクチャー用）
- インテル ® 64（旧インテル EM64T）ベースのプラットフォーム用開発パッケージ - インテル ® IPP Linux* 版（インテル ® 64 アーキテクチャー用）
- インテル ® Itanium® プロセッサー・ファミリー・プラットフォーム用開発パッケージ - インテル ® IPP Linux* 版（IA-64 アーキテクチャー用）
- インテル ® Atom™ プロセッサー・プラットフォーム用開発パッケージ - インテル ® IPP 6.0 Linux* 版（低消費電力インテル ® アーキテクチャー用）

テクニカルサポート

インテルでは、使い方のヒント、既知の問題点、製品のエラッタ、ライセンス情報、ユーザーフォーラムなどの多くのセルフヘルプ情報を含むサポート Web サイトを提供しています。インテル ® IPP サポート Web サイト http://www.intel.com/software/products/support/ipp を参照してください。

このドキュメントについて

このドキュメント - *インテル ® IPP Linux* 版（IA-32 アーキテクチャー用）ユーザーガイド* - は、IA-32 アーキテクチャー上で実行する Linux アプリケーションで、インテル ® IPP ルーチンを最大限に利用するための*使用方法*を提供することを目的としています。この使用方法では、各アーキテクチャー用の機能を利用します。本ユーザーガイドでは、これらの機能に加えて、特定のアーキテクチャーに依存しない機能についても説明します。

このドキュメントは、`<install path>/doc` ディレクトリー（「ドキュメント・ディレクトリーの内容」を参照）に含まれています。

目的

このドキュメントでは、インテル® IPP に関する次の情報を説明します。

- ライブラリーを使用するために必要な、製品のインストール後に実行する手順。
- ライブラリーの設定方法とライブラリーを使用するための開発環境。
- ライブラリーの構造。
- 用途に最も適しているリンク方法を選択する方法、アプリケーションとライブラリーをリンクする方法の詳細な説明、および単純な使用方法の一般例。
- インテル® IPP を使用してアプリケーションをスレッド化する方法。
- インテル® IPP を使用してアプリケーションを作成、コンパイル、および実行する方法。
- インテル® IPP パフォーマンス・テスト・ツールを使用して関数のパフォーマンス・テストを実行する方法。
- 開発者が利用可能なインテル® IPP サンプルコードの紹介とサンプルを実行する方法。

対象者

このガイドは、ソフトウェア開発の経験がある Linux プログラマー向けに記述されています。

ドキュメントの構成

このドキュメントには、以下の章と付録が含まれています。

第 1 章	「概要」では、ドキュメントの目的と構成、および表記規則について説明します。
第 2 章	「インテル ® IPP 入門」では、インストール後にインテル ® IPP を使用するために必要な手順と情報について説明します。
第 3 章	「インテル ® IPP の構造」では、インストール後のインテル ® IPP ディレクトリーの構造と提供されるライブラリー・タイプについて説明します。
第 4 章	「開発環境の設定」では、インテル ® IPP およびライブラリーを使用するための開発環境の設定方法について説明します。
第 5 章	「インテル ® IPP と アプリケーションのリンク」では、リンク方法を比較します。目的に応じたリンク方法を選択するヒントと、インテル ® IPP ライブラリーをリンクするために使用する一般的なリンクの書式について説明します。また、カスタム・ダイナミック・ライブラリーを築する方法についても説明します。
第 6 章	「マルチスレッド・アプリケーションのサポート」では、マルチスレッド・アプリケーションにおけるスレッド数の設定方法、スレッド数の取得方法、マルチスレッディングの無効化について説明します。
第 7 章	「パフォーマンスとメモリーの管理」では、インテル ® IPP のパフォーマンスを向上するいくつかの方法について説明します。また、インテル ® IPP パフォーマンス・テスト・ツールを使用してインテル ® IPP 関数のパフォーマンスをテストする方法についても説明します。
第 8 章	「各種プログラミング言語でのインテル ® IPP の使用」では、異なるプログラミング言語と Linux 開発環境でインテル ® IPP を使用する方法について説明します。
付録 A	「パフォーマンス・テスト・ツールのコマンドライン・オプション」では、パフォーマンス・テスト・ツールのコマンドライン・オプションについて説明します。
付録 B	「インテル ® IPP のサンプル」では、開発者が利用可能なインテル ® IPP のサンプルコードをカテゴリー別に説明します。また、単純なメディア・プレーヤー・アプリケーションの例を使用して、サンプルを実行する方法について説明します。

ドキュメントには、「索引」も含まれています。

表記規則

このドキュメントでは、以下のフォント表記および記号を使用しています。

表 1-1 表記規則

斜体	*斜体*は、用語を強調するために使用されます。
等幅の小文字	ファイル名、ディレクトリーおよびパスを示します。 `tools/env/ippvars32.csh`
等幅の小文字と大文字	コード、コマンドおよびコマンドライン・オプションを示します。 `export LIB=$IPPROOT/lib:$LIB`
等幅の大文字	システム変数を示します。例：`LD_LIBRARY_PATH`
等幅の斜体	関数パラメーター（*lda* など）や makefile パラメーター（*functions_list* など）のようなパラメーターを示します。 山括弧で囲まれている場合、識別子、式、文字列、記号、または値のプレースホルダーを示します。 *<ipp directory>*
[項目]	角括弧は、括弧で囲まれている項目がオプションであることを示します。
{ 項目 | 項目 }	波括弧は、括弧内にリストされている項目を 1 つだけ選択できることを示します。垂直バー（ | ）は項目の区切りです。

インテル® IPP 入門

2

この章では、インテル® IPP を使用するために必要な情報と、製品のインストール後に実行する手順について説明します。

インストールの確認

インテル® IPP のインストールを完了した後、ライブラリーが適切にインストールされ構成されていることを確認してください。

1. インストールで選択したディレクトリー `<IPP directory>/ia32` が作成されていることを確認します。
2. ファイル `ippvars32.sh` が `tools/env` ディレクトリーに含まれていることを確認します。このファイルを使用して、ユーザーシェルで `LD_LIBRARY_PATH`、`LIB`、および `INCLUDE` 環境変数を設定します。
3. ディスパッチおよびプロセッサー固有のライブラリーがパス上に存在することを確認します。

   エラーメッセージ "No shared object library was found in the Waterfall procedure" が表示された場合、Linux でインテル® IPP 共有オブジェクト・ライブラリーの場所が特定できません。この問題を解決するには、以下を行ってください。

   - インテル® IPP ディレクトリーがパスに含まれていることを確認します。インテル® IPP 共有オブジェクト・ライブラリーを使用する前に、第 3 章の「インテル® IPP 共有オブジェクト・ライブラリー（SO）の使用」で説明されているように、共有オブジェクト・ライブラリーのパスをシステム変数 `LD_LIBRARY_PATH` に追加します。

バージョン情報の取得

バージョン番号、パッケージ ID、およびライセンス情報を含むライブラリーのバージョンに関する情報を取得するには、`ippGetLibVersion` 関数を呼び出します。関数の説明および呼び出し方法については、『インテル® IPP リファレンス・マニュアル』（v.1）の「サポート関数」の章を参照してください。

`include` ディレクトリーの `ippversion.h` ファイルを使用してバージョン情報を取得することもできます。

アプリケーションのビルド

アプリケーションをビルドするには、以下の手順に従ってください。

環境変数の設定

tools/env ディレクトリーにあるシェルスクリプト、ippvars32.sh を実行すると、インテル ® IPP の LD_LIBRARY_PATH、LIB および INCLUDE 環境変数が設定されます。

環境変数を手動で設定するには、第 3 章の「インテル ® IPP 共有オブジェクト・ライブラリー（SO）の使用」で説明されているように、LD_LIBRARY_PATH 変数に共有オブジェクト・ライブラリーのパスを追加してください。また、以下のコマンドを使用してインテル ® IPP のヘッダーファイルとライブラリー・ファイルの場所を指定する必要があります。

export INCLUDE=$IPPROOT/include:$INCLUDE (bash)
setenv INCLUDE=$IPPROOT/include:${INCLUDE} (csh) - ヘッダーファイル

export LIB=$IPPROOT/lib:$LIB (bash)
setenv LIB=$IPPROOT/lib:${LIB} (csh) - ライブラリー・ファイル

マルチスレッド・アプリケーションの環境変数を設定する方法については、「マルチスレッド・アプリケーションのサポート」を参照してください。

ヘッダーファイルのインクルード

インテル ® IPP 関数およびタイプは、関数定義域別に構成された複数のヘッダーファイルで定義されます。これらのヘッダーファイルは、include ディレクトリーに含まれています。例えば、ippac.h ファイルには、すべてのオーディオ・コーディングと処理関数が含まれています。

ipp.h ファイルには、すべてのインテル ® IPP ヘッダーファイルが含まれています。上位互換性を確保するため、プログラムでは ipp.h のみをインクルードしてください。

IPP 関数の呼び出し

共有ライブラリー・ディスパッチャーおよびマージド・スタティック・ライブラリーのメカニズム（「インテル ® IPP と アプリケーションのリンク」を参照）により、インテル ® IPP 関数は他の C 関数と同様に簡単に呼び出すことができます。

インテル ® IPP 関数を呼び出すには、以下の操作を行います。

1. ipp.h ヘッダーファイルをインクルードします。
2. 関数パラメーターを設定します。
3. 関数を呼び出します。

各関数の最適化されたコードが、1 つのエントリーポイントに存在します。関数の説明、パラメーターの一覧、戻り値、その他は、『インテル ® IPP リファレンス・マニュアル』を参照してください。

インテル ® IPP の使用を開始する前に

インテル ® IPP の使用を開始する前に、いくつかの重要な概念について説明します。

表 2-1 は、インテル ® IPP の使用を開始する前に知っておく必要がある項目を要約したものです。

表 2-1　開始前に知っておく必要がある項目

関数定義域	用途に応じたインテル ® IPP 関数定義域を識別します。 理由 : 使用する関数定義域を知っておくと、リファレンス・マニュアルで必要なルーチンを絞り込んで検索できます。 http://www.intel.com/software/products/ipp/samples.htm のサンプルも参照してください。 関数定義域と必要なライブラリーについては、表 5-8 を参照してください。また、クロス定義域の依存関係については、表 5-9 を参照してください。
リンク方法	適切なリンク方法を決定します。 理由 : 最適なリンク方法を選択すると、最適なリンク結果が得られます。各リンク方法の利点、リンクコマンドの構文と例、およびカスタム・ダイナミック・ライブラリーの作成方法のような、その他のリンクに関する情報については、「インテル ® IPP と アプリケーションのリンク」を参照してください。
スレッディング・モデル	アプリケーションをどのようにスレッド化するかに応じて、以下のオプションのいずれかを選択します。 • アプリケーションはすでにスレッド化されている。 • インテル ® IPP のスレッディング機能、つまり、互換 OpenMP* ランタイム・ライブラリー (`libiomp`)、またはサードパーティから提供されているスレッディング機能を使用する。 • アプリケーションをスレッド化しない。 理由 : インテル ® IPP はデフォルトで OpenMP* ソフトウェアを使用してスレッド数を設定します。異なるスレッド数を設定する必要がある場合、利用可能なメカニズムの 1 つを使用して手動で設定する必要があります。詳細は、「マルチスレッド・アプリケーションのサポート」を参照してください。

インテル® IPP の構造

3

この章では、インストール後のインテル® IPP ディレクトリーの構造と提供されるライブラリー・タイプについて説明します。

ディレクトリー構造

表 3-1 は、インストール後のインテル® IPP のディレクトリー構造を示しています。

表 3-1 ディレクトリー構造

ディレクトリー	ファイルタイプ
<ipp directory>	メイン・ディレクトリー
<ipp directory>/ippEULA.txt	インテル® IPP のエンド・ユーザー・ライセンス契約書
<ipp directory>/doc	インテル® IPP ドキュメント・ファイル
<ipp directory>/include	インテル® IPP ヘッダーファイル
<ipp directory>/lib	インテル® IPP スタティック・ライブラリー
<ipp directory>/sharedlib	インテル® IPP 共有オブジェクト・ライブラリー
<ipp directory>/tools	インテル® IPP パフォーマンス・テスト・ツール、リンクツール、および環境変数設定ツール。

ライブラリー

表 3-2 は、インテル ® IPP のライブラリー・タイプとライブラリー・ファイルの例の一覧です。

表 3-2 インテル ® IPP のライブラリー・タイプ

ライブラリー・タイプ	説明	フォルダーの場所	例
ダイナミック	共有オブジェクト・ライブラリーには、プロセッサー・ディスパッチャーと関数実装の両方が含まれます。	ia32/sharedlib	libipps.so.6.0, libippst7.so.6.0
	共有オブジェクト・ライブラリーのソフトリンク	ia32/sharedlib	libipps.so libippst7.so
スタティック・マージド	対応プロセッサー向けの関数実装を含む		
	位置独立コード（PIC）を含むライブラリー	ia32/lib	libippsmerged.a
	非 PIC（*1）ライブラリー	ia32/lib/nonpic	libippsmerged.a
マルチスレッド・スタティック・マージド	マルチスレッド関数実装を含む	ia32/lib	libippsmerged_t.a
スタティック・エマージド	マージド・ライブラリーのディスパッチャー		
	位置独立コード（PIC）を含むライブラリー	ia32/lib	libippsemerged.a
	非 PIC（*1）ライブラリー	ia32/lib/nonpic	libippsemerged.a

（*1）非 PIC ライブラリーはカーネルモードおよびデバイスドライバーでの利用に適しています。

インテル ® IPP 共有オブジェクト・ライブラリー（SO）の使用

ia32/sharedlib ディレクトリーには、インテル ® IPP の共有オブジェクト・ライブラリー（SO）とライブラリーへのソフトリンクがインストールされます。

共有オブジェクト・ライブラリーを使用する前に、tools/env ディレクトリーのシェルスクリプト ippvars32.sh を使用して LD_LIBRARY_PATH システム変数にパスを追加します。

または、LD_LIBRARY_PATH 変数を手動で設定します。例えば、ライブラリーが <*IPP directory*>/ia32/sharedlib ディレクトリーに含まれている場合、次のコマンドを入力します (bash の場合)。

```
export LD_LIBRARY_PATH=
        <IPP directory>/ia32/sharedlib:$LD_LIBRARY_PATH
```

（csh の場合）

```
setenv LD_LIBRARY_PATH=
        <IPP directory>/ia32/sharedlib:${LD_LIBRARY_PATH}
```

共有ライブラリー libipp*.so.6.0（* は適切な関数定義域を示す）は、" ディスパッチャー " ダイナミック・ライブラリーです。実行時にプロセッサーを検出して、適切なプロセッサー固有の共有ライブラリーをロードします。自動的に適切なバージョンが使用されるため、コードが実行されるプロセッサーに関係なくインテル ® IPP 関数を呼び出すことができます。これらのプロセッサー固有ライブラリーには、libipp*px.so.6.0, libipp*w7.so.6.0, libipp*t7.so.6.0,

libipp*v8.so.6.0, および libipp*p8.so.6.0 のように名前が付けられています (表 5-3 を参照)。例えば、ia32/sharedlib ディレクトリーの libippiv8.so.6.0 は、インテル ® Core™2 Duo プロセッサー向けに最適化された画像処理ライブラリーです。

共有ライブラリー自身をインクルードする代わりに、共有ライブラリーへのソフトリンクをインクルードします。これらのソフトリンクには、libipp*.so, libipp*px.so, libipp*w7.so, libipp*t7.so, libipp*v8.so, および libipp*p8.so のように、対応する共有ライブラリーからバージョン識別子を削除した名前が付けられています。

「アプリケーションで必要なインテル ® IPP ライブラリーの選択」を参照してください。

注 : 適切な libiomp5.so も LD_LIBRARY_PATH 環境変数に含まれている必要があります。IA-32 アーキテクチャーのシステムで実行するときは、sharelib ディレクトリーをインクルードしてください。

インテル ® IPP スタティック・ライブラリーの使用

lib インテル ® IPP は、各関数のすべてのプロセッサー・バージョンが含まれる「マージド」スタティック・ライブラリー・ファイルを提供します。これらのファイルは、ia32/lib ディレクトリーにインストールされます (表 3-1 を参照)。

ダイナミック・ディスパッチャーの場合と同じように、関数が呼び出されると、適切な関数のバージョンが実行されます。このメカニズムはダイナミックのメカニズムほど便利ではありませんが、スタティック・ライブラリーのコードサイズの合計がより小さくなります。

これらのスタティック・ライブラリーを使用するには、lib ディレクトリーの libipp*merged.a ファイルにリンクします。次に、インテル ® IPP リンクサンプル (英語) の説明に従って、必要な関数についてスタブのディスパッチを作成します。シェル・スクリプト ファイル ippvars32.sh を使用して LIB 環境変数を設定するか、またはフルパスを使用してこれらのファイルを参照する必要があります。

「アプリケーションで必要なインテル ® IPP ライブラリーの選択」を参照してください。

ドキュメント・ディレクトリーの内容

表 3-3 は、インテル ® IPP インストール・ディレクトリーの /doc サブディレクトリーにインストールされるドキュメントの一覧です。

表 3-3　/doc ディレクトリーの内容

ファイル名	説明	注
ipp_documentation.htm	ドキュメント・インデックス。インテル ® IPP ドキュメントの一覧と各ドキュメントへのリンクが含まれます。	
ReleaseNotes.htm	製品の概要および本リリースについての情報。	このファイルは、製品のインストール前に表示できます。
README.txt	初期ユーザー情報	
INSTALL.htm	インストール・ガイド	
ThreadedFunctionsList.txt	OpenMP* を使用してスレッド化されたインテル ® IPP 関数の一覧	
userguide_lin_ia32.pdf	本ドキュメント - 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・ユーザーガイド	

表 3-3 /doc ディレクトリーの内容

ファイル名	説明	注
	インテル ® IPP リファレンス・マニュアル（全 4 巻）	
ippsman.pdf	*信号処理*（第 1 巻）- 信号処理、オーディオ・コーディング、音声認識およびコーディング、データ圧縮および完全性、ストリング処理、ベクトル演算用のインテル ® IPP 関数とインターフェイスの詳細な説明が含まれます。	
ippiman.pdf	*イメージおよびビデオ処理*（第 2 巻）- イメージ処理および圧縮、カラー変換およびフォーマット変換、コンピューター・ビジョン、ビデオ・コーディング用のインテル ® IPP 関数とインターフェイスの詳細な説明が含まれます。	
ippmman.pdf	*小行列、リアリスティック・レンダリング*（第 3 巻）- ベクトルおよび行列代数、連立 1 次方程式、最小 2 乗問題および固有値問題、リアリスティック・レンダリング、3D データ処理用のインテル ® IPP 関数とインターフェイスの詳細な説明が含まれます。	
ippcpman.pdf	*暗号化*（第 4 巻）- 暗号化用のインテル ® IPP 関数とインターフェイスの詳細な説明が含まれます。	

開発環境の設定

4

この章は、インテル® IPP を使用するための開発環境の設定方法について説明します。

Eclipse CDT でインテル® IPP をリンクするための設定

CDT とインテル® IPP をリンクすると、Eclipse で提供されるコード支援機能を利用できます。詳細は、*Eclipse Help* の *Code/Context Assist* の説明（英語）を参照してください。

Eclipse CDT 4.0 の設定

Eclipse CDT 4.0 とインテル® IPP をリンクするには、次の操作を行います。

1. ツールチェーン / コンパイラー統合が `include` パス・オプションをサポートしている場合、**[C/C++ General]** > **[Paths and symbols]** プロパティー・ページの **[Includes]** タブを開いて、インテル® IPP の `include` パスを設定します。例えば、デフォルトの場合、`<IPP directory>/ia32/include` を設定します。
2. ツールチェーン / コンパイラー統合がライブラリー・パス・オプションをサポートしている場合、**[C/C++ General]** > **[Paths and symbols]** プロパティー・ページの **[Library paths]** タブを開いて、インテル® IPP ライブラリーのパスをターゲットのアーキテクチャーに応じて設定します。例えば、デフォルトの場合、`<IPP directory>/ia32/lib` を設定します。
3. 一部のビルドでは、**[C/C++ Build]** > **[Settings]** プロパティー・ページの **[Tool settings]** タブに移動して、アプリケーションとリンクするインテル® IPP ライブラリーの名前を指定します。ライブラリーの選択については、第 5 章の「アプリケーションで必要なインテル® IPP ライブラリーの選択」を参照してください。ライブラリーを指定する場合の設定名はコンパイラー統合に依存します。

コンパイラー / リンカーは、自動 makefile 生成がオンになっている場合のみ、インクルード・パスとライブラリー・パス設定を自動的に選択することに注意してください。自動 makefile 生成がオンではない場合、使用する makefile でインクルード・パスとライブラリー・パスを直接指定する必要があります。

Eclipse CDT 3.x の設定

Eclipse CDT 3.x とインテル® IPP をリンクするには、次の操作を行います。

Standard Make プロジェクトの場合

1. [C/C++ Include Paths and Symbols] プロパティー・ページを開いて、インテル® IPP の include パスを設定します。例えば、デフォルトの場合、<*IPP directory*>/ia32/include を設定します。
2. [C/C++ Project Paths] プロパティー・ページの [Libraries] タブを開いて、アプリケーションとリンクするインテル® IPP ライブラリーを設定します。ライブラリーの選択方法については、第 5 章の「アプリケーションで必要なインテル® IPP ライブラリーの選択」を参照してください。

Standard Make では、上記の設定は CDT の内部的な機能でのみ必要です。コンパイラー / リンカーは自動的にこれらの設定を選択しません。makefile で直接指定する必要があります。

Managed Make プロジェクトの場合

特定のビルド用の設定を指定します。

1. [C/C++ Build] プロパティー・ページの [Tool Settings] タブを開きます。指定する必要がある設定は、このページにすべて含まれています。特定の設定の名前はコンパイラーの統合に依存するため、ここでは説明しません。
2. コンパイラー統合がインクルード・パス・オプションをサポートしている場合、インテル® IPP include パスを設定します。例えば、デフォルトの場合、<*IPP directory*>/ia32/include を設定します。
3. コンパイラー統合がライブラリー・パス・オプションをサポートしている場合、ターゲット・アーキテクチャーに応じて、インテル® IPP ライブラリーのパスを設定します。例えば、デフォルトの場合、<*IPP directory*>/ia32/lib を設定します。
4. アプリケーションとリンクするインテル® IPP ライブラリーの名前を指定します。ライブラリーの選択については、第 5 章の「アプリケーションで必要なインテル® IPP ライブラリーの選択」を参照してください。

インテル® IPP を使用するプロジェクトが開かれてアクティブになっていることを確認します。

インテル® IPP と アプリケーションのリンク

5

この章は、インテル® IPP とアプリケーションのリンクについて説明します。ユーザーが最も適したリンク方法を選択できるように、開発環境および動作環境、インストールの仕様、ランタイム条件、およびその他のアプリケーション要件によるリンク方法の違いを考慮して、各リンク方法のリンク手順を示します。また、リンク例も紹介します。

ディスパッチ

インテル® IPP は、さまざまな CPU 向けに最適化されたコードを使用します。ディスパッチとは、CPU を検出して、使用しているハードウェアに対応するインテル® IPP バイナリーを選択することです。例えば、`ia32/sharedlib` ディレクトリーの `libippiv8.so.6.0` は、インテル® Core™2 Duo プロセッサー向けに最適化された画像処理ライブラリーです。

インテル® IPP 関数には、さまざまなアーキテクチャー向けのインテル® プロセッサーで実行するために最適化された多くのバージョンが用意されています。例えば、`ippsCopy_8u()` の場合、インテル® Pentium® 4 プロセッサー向けに最適化された関数は `w7_ippsCopy_8u()` です。

表 5-1 は、インテル® IPP で使用されるプロセッサー固有のコードを示しています。

表 5-1　プロセッサー固有のライブラリーと関連付けられているコードの識別子

省略形	意味
	IA-32 インテル® アーキテクチャー
px	すべての IA-32 プロセッサー向けに C の最適化
w7	インテル® ストリーミング SIMD 拡張命令 2（SSE2）対応プロセッサー向けに最適化
t7	インテル® ストリーミング SIMD 拡張命令 3（SSE3）対応プロセッサー向けに最適化
v8	インテル® ストリーミング SIMD 拡張命令 3 補足命令（SSSE3）対応プロセッサー向けに最適化
p8	インテル® ストリーミング SIMD 拡張命令 4.1（SSE4.1）対応プロセッサー向けに最適化

プロセッサーのタイプの検出

コンピューター・システムで使用しているプロセッサーのタイプを検出するには、`ippcore.h` ファイルで宣言されている `ippGetCpuType` 関数を使用します。この関数は、適切な `IppCpuType` 変数値を返します。すべての列挙値は、`ippdefs.h` ヘッダーファイルで指定されます。例えば、戻り値 `ippCpuCoreDuo` は、システムがインテル® Core™ Duo プロセッサーを使用していることを意味します。

表 5-2 は、`ippGetCpuType` の戻り値と意味を示しています。

表 5-2 プロセッサーのタイプの検出、戻り値と意味

戻り値	プロセッサーのタイプ
ippCpuPP	インテル ® Pentium® プロセッサー
ippCpuPMX	MMX® テクノロジー インテル ® Pentium® プロセッサー
ippCpuPPR	インテル ® Pentium® Pro プロセッサー
ippCpuPII	インテル ® Pentium® II プロセッサー
ippCpuPIII	インテル ® Pentium® III プロセッサーおよびインテル ® Pentium® III Xeon® プロセッサー
ippCpuP4	インテル ® Pentium® 4 プロセッサーおよびインテル ® Xeon® プロセッサー
ippCpuP4HT	ハイパースレッディング・テクノロジー対応インテル ® Pentium® 4 プロセッサー
ippCpuP4HT2	インテル ® ストリーミング SIMD 拡張命令 3 対応インテル ® Pentium® プロセッサー
ippCpuCentrino	インテル ® Centrino™ モバイル・テクノロジー
ippCpuCoreSolo	インテル ® Core™ Solo プロセッサー
ippCpuCoreDuo	インテル ® Core™ Duo プロセッサー
ippCpuITP	インテル ® Itanium® プロセッサー
ippCpuITP2	インテル ® Itanium® 2 プロセッサー
ippCpuEM64T	インテル ® 64 命令セット・アーキテクチャー（ISA）
ippCpuC2D	インテル ® Core™2 Duo プロセッサー
ippCpuC2Q	インテル ® Core™2 Quad プロセッサー
ippCpuBonnell	インテル ® Atom™ プロセッサー
ippCpuPenryn	インテル ® ストリーミング SIMD 拡張命令 4.1 対応インテル ® Core™2 プロセッサー
ippCpuSSE	インテル ® ストリーミング SIMD 拡張命令対応プロセッサー
ippCpuSSE2	インテル ® ストリーミング SIMD 拡張命令 2 対応プロセッサー
ippCpuSSE3	インテル ® ストリーミング SIMD 拡張命令 3 対応プロセッサー
ippCpuSSSE3	インテル ® ストリーミング SIMD 拡張命令 3 補足命令対応プロセッサー
ippCpuSSE41	インテル ® ストリーミング SIMD 拡張命令 4.1 対応プロセッサー
ippCpuSSE42	インテル ® ストリーミング SIMD 拡張命令 4.2 対応プロセッサー
ippCpuX8664	64 ビット拡張命令対応プロセッサー
ippCpuUnknown	未知のプロセッサー

リンク方法の選択

インテル ® IPP では、次のようにさまざまなリンク方法を使用できます。

- ランタイム共有オブジェクト・ライブラリー（SO）を使用したダイナミック・リンク
- エマージドおよびマージド・スタティック・ライブラリーを使用したディスパッチありのスタティック・リンク
- マージド・スタティック・ライブラリーを使用した自動ディスパッチなしのスタティック・リンク
- 独自のカスタム SO を使用したダイナミック・リンク

最適なリンク方法を選択するために、以下の点について考えてみてください。

- アプリケーションの実行ファイルのサイズに上限はあるか？アプリケーションのインストール・パッケージのサイズに上限はあるか？
- インテル ® IPP ベースのアプリケーションは、カーネルモードで実行するデバイスドライバーまたは同様の ring 0 ソフトウェアか？
- ユーザーが異なるプロセッサー上にアプリケーションをインストールできるようにするか？それとも単一のプロセッサーのみに対応させるか？アプリケーションは単一プロセッサーを使用する組み込みコンピューター用か？
- カスタマイズしたインテル ® IPP コンポーネントを保守および更新する余裕はあるか？アプリケーションに新しいプロセッサー向けの最適化を追加するためにどの程度の労力をかけることができるか？
- アプリケーションを更新する頻度はどのくらいか？アプリケーションのコンポーネントは独立して配布するか？それとも常にアプリケーションと一緒に含まれるか？

ダイナミック・リンク

ダイナミック・リンクは最も簡単で最も一般的なリンク方法です。この方法は、共有オブジェクト・ライブラリー（SO）でダイナミック・ディスパッチ・メカニズムを最大限に活用できます（「インテル ® IPP の構造」を参照）。次の表は、ダイナミック・リンクの長所と短所の要約です。

表 5-3 ダイナミック・リンクの機能の要約

長所	短所
• プロセッサー固有の最適化の自動ランタイム・ディスパッチ • 再コンパイル / 再リンクしないで新しいプロセッサーの最適化を更新可能 • 複数のインテル ® IPP ベースの実行ファイルを作成する場合に必要なディスク容量が少なくなる • 複数のインテル ® IPP ベースのアプリケーションで実行時により効率的なメモリーの共有が可能	• アプリケーションを実行するときにインテル ® IPP ランタイム共有オブジェクト・ライブラリー（SO）にアクセスする必要がある • カーネルモード / デバイスドライバー /ring-0 コードには不適切 • 非常に小規模なダウンロードが必要な Web アプレット / プラグインには不適切 • インテル ® IPP SO を最初にロードするときにパフォーマンス・ペナルティーが発生する

インテル ® IPP をダイナミックにリンクするには、以下の操作を行います。

1. すべての IPP 定義域のヘッダーファイルを含む `ipp.h` を追加します。
2. 標準の IPP 関数名を使用して IPP 関数を呼び出します。
3. 対応する定義域ソフトリンクをリンクします。例えば、`ippsCopy_8u` 関数を使用する場合、`libipps.so` をリンクします。
4. (現在のセッションでインテル® IPP ライブラリーを使用する前に、シェルスクリプト `<install path>/tools/env/ippvars32.sh` を実行するか、`LD_LIBRARY_PATH` 環境変数を正しく設定してください。次に例を示します。
   `export LD_LIBRARY_PATH =$IPPROOT/sharedlib:$LD_LIBRARY_PATH` (bash) または
   `setenv LD_LIBRARY_PATH =$IPPROOT/sharedlib:${LD_LIBRARY_PATH}` (csh).

スタティック・リンク（ディスパッチあり）

利用するインテル ® IPP 関数の数が少なく、必要なメモリー・フットプリン g が小さいアプリケーションもあります。「エマージド」および「マージド」ライブラリー経由でスタティック・リンク・ライブラリーを使用すると、小さなフットプリントと、複数のプロセッサーにおける最適化という長所を活用できます。 エマージド・ライブラリー（例えば、`libippsemerged.a`）は、非修飾の（標準名

の）IPP 関数のエントリーポイントと、各プロセッサー固有の実装へのジャンプテーブルを提供します。アプリケーションをリンクするとき、関数は、libippcore.a の関数で検出した CPU 設定に従ってマージド・ライブラリー（例えば、libippsmerged.aÅj の対応する関数を呼び出します。エマージド・ライブラリーには実装コードは含まれません。

エマージド・ライブラリーは、関数を呼び出す前に初期化する必要があります。ライブラリーを初期化するには、ippStaticInit() 関数を使用して最適化の選択をライブラリーに任せる方法と、ippStaticInitCpu() 関数を使用して CPU を指定する方法があります。どのような場合でも、この 2 つの関数のいずれかを、他の IPP 関数を使用する前に呼び出す必要があります。初期化関数を呼び出さなかった場合、IPP 関数の「px」バージョンが呼び出されるため、アプリケーションのパフォーマンスが低下します（最もよく発生する問題です)。パフォーマンスの違いは、t2.cpp ファイル（例 5-1）を使用するとよくわかります。

例 5-1　StaticInit を呼び出した場合と呼び出さなかった場合のパフォーマンスの違い

```
#include <stdio.h>
#include <ipp.h>

int main() {
        const int N = 20000, loops = 100;
        Ipp32f src[N], dst[N];
        unsigned int seed = 12345678, i;
        Ipp64s t1,t2;
        /// no StaticInit call, means PX code, not optimized
        ippsRandUniform_Direct_32f(src,N,0.0,1.0,&seed);
        t1=ippGetCpuClocks();
        for(i=0; i<loops; i++)
                ippsSqrt_32f(src,dst,N);
        t2=ippGetCpuClocks();
        printf("without StaticInit: %.1f clocks/element\n",
                    (float)(t2-t1)/loops/N);
        ippStaticInit();
        t1=ippGetCpuClocks();
        for(i=0; i<loops; i++)
                ippsSqrt_32f(src,dst,N);
        t2=ippGetCpuClocks();
        printf("with StaticInit: %.1f clocks/element\n",
                    (float)(t2-t1)/loops/N);
        return 0;
}


t2.cpp

cmdlinetest>t2
without StaticInit: 61.3 clocks/element
with StaticInit: 4.5 clocks/element
```

表 5-4 は、エマージド・ライブラリー経由のスタティック・リンクの長所と短所の要約です。このリンク方法を使用する前に確認してください。

表 5-4　スタティック・リンクの機能の要約（ディスパッチあり）

長所	短所
• ランタイム中にプロセッサー固有の最適化をディスパッチ • ライブラリーを含むアプリケーションの実行ファイルを作成 • フルセットのインテル ® IPP SO よりも小さなフットプリントを生成	• インテル ® IPP コードが複数のインテル ® IPP ベースのアプリケーションに重複して含まれる • プログラムの初期化中にディスパッチャー初期化用の追加関数呼び出しが（1 回）必要

ディスパッチありのスタティック・リンクを使用するには、次の操作を行います。

1. コードに ipp.h をインクルードします。
2. インテル ® IPP 関数を呼び出す前に、ippStaticInit() または ippStaticInitCpu() のいずれかの関数をヘッダーファイル ippcore.h で宣言して使用し、スタティック・ディスパッチャーを初期化します。
3. 標準の IPP 関数名を使用して IPP 関数を呼び出します。
4. 対応するエマージド・ライブラリー、マージド・ライブラリー、libippcore.a を（順に）リンクします。例えば、ippsCopy_8u() 関数を使用する場合、libippsemerged.a、libippsmerged.a、および libippcore.a ライブラリーをリンクします。

スタティック・リンク（ディスパッチなし）

このリンク方法は、マージド・スタティック・ライブラリーを直接リンクします。用意されているエマージド・ディスパッチャーの代わりに独自のスタティック・ディスパッチャーを使用する場合は、この方法を使用します。サンプル mergelib で、このリンク方法を説明しています。

最新のサンプルは、http://www.intel.com/software/products/ipp/samples.htm からインテル ® IPP サンプルをダウンロードして展開した後、ipp-samples/advanced-usage/linkage/mergelib ディレクトリーを参照してください。

実行ファイルにライブラリーを含むアプリケーションを作成しており、対応プロセッサーが 1 種類のみで、実行ファイルのサイズを小さくする必要がある場合に最適なリンク方法です。アプリケーションが 1 種類のプロセッサーとバンドルされる組み込みアプリケーションで一般的に使用されます。

表 5-5 は、このリンク方法の長所と短所の要約です。

表 5-5　スタティック・リンクの機能の要約（ディスパッチなし）

長所	短所
• 対応プロセッサーが 1 種類のみなので実行ファイルが小さい • カーネルモード / デバイスドライバー /ring-0 コードに最適 *) • 非常に小さなファイルのダウンロードが必要な Web アプレットまたはプラグインで対応プロセッサーが 1 種類のみの場合に最適 • 実行ファイルにライブラリーが含まれているため、実行時にインテル ® IPP ランタイム SO が不要 • アプリケーション・パッケージで最小のフットプリント • 最小のインストール・パッケージ	• 実行ファイルは 1 種類のプロセッサー向けにのみ最適化 • プロセッサー固有の最適化の更新にはリビルドまたは再リンクが必要

*) スレッド化されていない非 PIC ライブラリーのみ。

インテル ® IPP パッケージに含まれている（ipp_w7.h のような）プロセッサー固有のヘッダーファイルのセット を、IPPCALL マクロの代わりに使用することもできます。詳細は、ia32/tools/staticlib/readme.htm の「インテル ® IPP 関数のスタティック・リンク（1 つのプロセッサーの場合)」を参照してください。

カスタム SO のビルド

一部のアプリケーションではいくつかの内部モジュールが含まれていて、インテル ® IPP コードはこれらのモジュールとのみ共有しなければならない場合があります。この場合、アプリケーションで使用するインテル ® IPP 関数のみを含むカスタマイズした共有オブジェクト・ライブラリー（SO）をダイナミック・リンクして使用できます。表 5-6 は、カスタム SO の長所と短所の要約です。

表 5-6　カスタム SO の機能

長所	短所
• プロセッサー固有の最適化のランタイム・ディスパッチ • フルセットのインテル ® IPP SO よりもハードドライブのフットプリントが小さい • 複数のアプリケーションで同じインテル® IPP 関数を使用する場合の最小のインストール・パッケージ	• アプリケーションを実行するときにインテル ® コンパイラー固有のランタイム・ライブラリーにアクセスする必要がある • カスタム SO の作成および保守に開発者のリソースが必要 • 新しいプロセッサー固有の最適化を利用するにはカスタム SO のリビルドが必要 • カーネルモード / デバイスドライバー/ring-0 コードには不適切

カスタム SO を作成するには、SO およびスタブを生成する別のビルドステップまたはプロジェクトを作成する必要があります。サンプル `custom so` でこの方法を説明しています。最新のサンプルは、http://www.intel.com/software/products/ipp/samples.htm からインテル® IPP サンプルをダウンロードして展開した後、`ipp-samples/advanced-usage/linkage/customso` ディレクトリーを参照してください。

インテル ® IPP のリンク方法の比較

表 5-7 は、インテル ® IPP のリンク方法の比較です。

表 5-7 インテル ® IPP のリンク方法の比較

特徴	ダイナミック・リンク	スタティック・リンク（ディスパッチあり）	スタティック・リンク（ディスパッチなし）	カスタム SO の使用
プロセッサーの更新	自動	再コンパイルして再配布	新しいプロセッサー固有のアプリケーションをリリース	再コンパイルして再配布
最適化	すべてのプロセッサー	すべてのプロセッサー	1 つのプロセッサー	すべてのプロセッサー
ビルド	スタブ・スタティック・ライブラリーをリンク	スタティック・ライブラリーとスタティック・ディスパッチャーをリンク	マージド・ライブラリーまたはマルチスレッド・マージド・ライブラリーをリンク	別の SO をビルド
呼び出し	規則的な名前	規則的な名前	プロセッサー固有の名前	規則的な名前
合計バイナリーサイズ	大きい	小さい	最小	小さい
実行ファイルのサイズ	最小	小さい	小さい	最小
カーネルモード	いいえ	はい	はい	いいえ

アプリケーションで必要なインテル ® IPP ライブラリーの選択

特定の関数定義域を使用する場合に各リンク方法で使用するライブラリーの一覧を、表 5-8 に示します。

表 5-8 各リンク方法で使用するライブラリー

定義域の説明	ヘッダーファイル	ダイナミック・リンク	スタティック・リンク（ディスパッチあり）& カスタム・ダイナミック・リンク	スタティック・リンク（ディスパッチなし）
オーディオ・コーディング	ippac.h	libippac.so	libippacemerged.a	libippacmerged.a libippacmerged_t.a
カラー変換	ippcc.h	libippcc.so	libippccemerged.a	libippccmerged.a libippccmerged_t.a
ストリング処理	ippch.h	libippch.so	libippchemerged.a	libippchmerged.a libippchmerged_t.a
暗号化	ippcp.h	libippcp.so	libippcpemerged.a	libippcpmerged.a libippcpmerged_t.a
コンピューター・ビジョン	ippcv.h	libippcv.so	libippcvemerged.a	libippcvmerged.a libippcvmerged_t.a
データ圧縮	ippdc.h	libippdc.so	libippdcemerged.a	libippdcpmerged.a libippdcmerged_t.a

表 5-8　　各リンク方法で使用するライブラリー (続き)

定義域の説明	ヘッダーファイル	ダイナミック・リンク	スタティック・リンク (ディスパッチあり) & カスタム・ダイナミック・リンク	スタティック・リンク (ディスパッチなし)
データ完全性	ippdi.h	libippdi.so	libippdipemerged.a	libippdimerged.a libippdimerged_t.a
生成関数	ipps.h	libippgen.so	libippgenemerged.a	libippgenmerged.a libippgenmerged_t.a
イメージ処理	ippi.h	libippi.so	libippiemerged.a	libippimerged.a libippimerged_t.a
イメージ圧縮	ippj.h	libippj.so	libippjemerged.a	libippjmerged.a libippjmerged_t.a
リアリスティック・レンダリング および 3D データ処理	ippr.h	libippr.so	libippremerged.a	libipprmerged.a libipprmerged_t.a
小行列演算	ippm.h	libippm.so	libippmemerged.a	libippmmerged.a libippmmerged_t.a
信号処理	ipps.h	libipps.so	libippsemerged.a	libippsmerged.a libippsmerged_t.a
音声 コーディング	ippsc.h	libippsc.so	libippscpemerged.a	libippscmerged.a libippscpmerged_t.a
音声認識	ippsr.h	libippsr.so	libippsremerged.a	libippsrmerged.a libippsrmerged_t.a
ビデオ・コーディング	ippvc.h	libippvc.so	libippvcemerged.a	libippvcmerged.a libippvcmerged_t.a
ベクトル演算	ippvm.h	libippvm.so	libippvmemerged.a	libippvmmerged.a libippvmmerged_t.a
コア関数	ippcore.h	libippcore.so	libippcore.a	libippcore.a libippcore_t.a

ダイナミック・リンク

共有オブジェクトを使用するには、sharedlib ディレクトリーの libipp*.so ファイルへのソフトリンクを使用する必要があります。* は、適切な関数定義域を示します。アプリケーションで使用している定義域ライブラリーに加えて、libipps.so、libippcore.so および libiomp.so をリンクする必要があります。

例えば、アプリケーションで 3 つのインテル® IPP 関数 ippiCopy_8u_C1R、ippiCanny_16s8u_C1R および ippmMul_mc_32f を使用しているとします。これらの 3 つの関数はそれぞれ、イメージ処理、コンピューター・ビジョンおよび小行列演算定義域に属しています。このため、これらの関数をアプリケーションで使用するには、以下のインテル® IPP ライブラリーをリンクする必要があります。

libippi.so

libippcv.so

libippm.so

libippcore.so

libiomp5.so

スタティック・リンク（ディスパッチあり）

スタティック・リンク・ライブラリーを使用するには、lib*emerged.a、lib*merged.a、libsemerged.a、libsmerged.a および libcore.a をリンクする必要があります。* は、適切な関数定義域を示します。

OpenMP* を使用してスレッド化されたインテル ® IPP 関数を使用する場合は、lib*emerged.a、lib*merged_t.a、libsemerged.a、libsmerged_t.a、libcore_t.a および libiomp.a をリンクする必要があります。

これらのライブラリーはすべて、定義域固有の関数を含む lib ディレクトリーにあります。アプリケーションで使用しているすべての定義域と信号処理定義域のマージド・ライブラリーおよびエマージド・ライブラリーの両方をリンクする必要があることに注意してください。

例えば、アプリケーションで 3 つのインテル ® IPP 関数 ippiCopy_8u_C1R、ippiCanny_16s8u_C1R および ippmMul_mc_32f を使用しているとします。これらの 3 つの関数はそれぞれ、イメージ処理、コンピューター・ビジョンおよび小行列演算定義域に属しています。ライブラリーをリンクする順序は、定義域によるライブラリーの依存関係（下記を参照）に対応する必要があることに注意してください。マルチスレッド関数を使用する場合は、アプリケーションに以下のライブラリーをリンクする必要があります。

libcvemerged.a および libcvmerged_t.a

libmemerged.a および libmmerged_t.a

libiemerged.a および libimerged_t.a

libsemerged.a および libsmerged_t.a

libcore_t.a

libiomp5.a

定義域別のライブラリー依存関係（スタティック・リンクのみ）

表 5-9 は、定義域別のライブラリー依存関係の一覧です。特定のライブラリー（例えば、データ圧縮定義域）をリンクする場合、そのライブラリーが依存するライブラリー（例えば、信号処理およびコア関数）にリンクする必要があります。

ライブラリーをリンクする場合、ライブラリー列のライブラリーを、依存ライブラリー列のライブラリーよりも前にリンクする必要があります。

以下は、定義域別のライブラリー依存関係の一覧です。

表 5-9 定義域別のライブラリー依存関係

定義域	ライブラリー	依存ライブラリー
オーディオ・コーディング	ippac	ippdc, ipps, ippcore
カラー変換	ippcc	ippi, ipps, ippcore
暗号化	ippcp	ippcore
コンピューター・ビジョン	ippcv	ippi, ipps, ippcore
データ圧縮	ippdc	ipps, ippcore
データ完全性	ippdi	ippcore
生成関数	ippgen	ipps, ippcore

表 5-9 定義域別のライブラリー依存関係

定義域	ライブラリー	依存ライブラリー
イメージ処理	ippi	ipps, ippcore
イメージ圧縮	ippj	ippi, ipps, ippcore
小行列演算	ippm	ippi, ipps, ippcore
リアリスティック・レンダリングおよび 3D データ処理	ippr	ippi, ipps, ippcore
信号処理	ipps	ippcore
音声コーディング	ippsc	ipps, ippcore
音声認識	ippsr	ipps, ippcore
ストリング処理	ippch	ipps, ippcore
ビデオ・コーディング	ippvc	ippi, ipps, ippcore
ベクトル演算	ippvm	ippcore

関数が属している定義域は、『インテル® IPP リファレンス・マニュアル』を参照してください。

リンク例

リンク例は、http://www.intel.com/software/products/ipp/samples.htm を参照してください。

サンプルコードの使用についての詳細は、「インテル® IPP のサンプル」を参照してください。

マルチスレッド・アプリケーションのサポート

6

この章では、インテル ® IPP をマルチスレッド・アプリケーションで使用する方法について説明します。

インテル ® IPP スレッディングと OpenMP* のサポート

すべてのインテル ® IPP 関数は、ダイナミック・ライブラリーおよびスタティック・ライブラリーの両方でスレッドセーフであり、マルチスレッド・アプリケーションで使用できます。

一部のインテル ® IPP 関数には、マルチプロセッサーおよびマルチコアシステムで大幅にパフォーマンスが向上する OpenMP コードが含まれています。OpenMP コードは、カラー変換、フィルタリング、たたみ込み、暗号化、相互相関、行列計算、距離の 2 乗、ビット数の削減、その他の関数に含まれています。

すべてのマルチスレッド関数の一覧は、`Documentation` ディレクトリーに含まれている *ThreadedFunctionsList.txt* ファイルを参照してください。

以前のバージョンのマルチスレッド API を含む、インテル ® IPP スレッディングと OpenMP のサポートに関する詳細は、http://www.intel.com/software/products/support/ipp を参照してください。

スレッド数の設定

インテル ® IPP スレッド・ライブラリーのデフォルトのスレッド数は、システムのプロセッサー数と等しく、`OMP_NUM_THREADS` 環境変数の値に依存しません。

インテル ® IPP で内部的に使用するスレッド数を設定するには、アプリケーションの先頭で `ippSetNumThreads(n)` 関数を呼び出します。ここで、`n` はスレッド数（1,...）です。内部で並列化を行わない場合は、`ippetNumThreads(1)` のように呼び出してください。

共有 L2 キャッシュの使用

信号処理定義域の一部の関数は、インテル ® Core™2 プロセッサー・ファミリーでは 2 スレッドにスレッド化され、マージド L2 キャッシュを活用します。これらの関数（単精度および倍精度 FFT、Div、Sqrt、その他）は、2 つのスレッドが同じダイ上で実行された場合にパフォーマンスが最大になります。この場合、これらのスレッドは同じ共有 L2 キャッシュ上で動作します。ダイに 2 つのコアが搭載されているプロセッサーでは、この条件は自動的に満たされます。2 つを超えるコアが搭載されているプロセッサーでは、特別な OpenMP 環境変数を設定する必要があります。

```
KMP_AFFINITY=compact
```

この環境変数を設定しない場合、最適なパフォーマンスが得られません。

入れ子の並列化

OpenMP を使用して作成されたマルチスレッド・アプリケーションでマルチスレッド・バージョンのインテル® IPP 関数を使用する場合、OpenMP では入れ子の並列化がデフォルトで無効になるため、この関数はシングルスレッドで動作します。

他のツールを使用して作成されたマルチスレッド・アプリケーションでマルチスレッド・バージョンのインテル® IPP 関数を使用する場合、入れ子の並列化とパフォーマンスの低下を回避するため、インテル® IPP でマルチスレッディングを無効にすることを推奨します。

マルチスレッディングの無効化

マルチスレッディングを無効にするには、アプリケーションに IPP の非スレッド・スタティック・ライブラリーをリンクするか、または非スレッド・スタティック・ライブラリーを使用してカスタム SO をビルドします。

パフォーマンスとメモリーの管理 7

この章では、メモリーのアライメント、しきい値、バッファーの再利用、FFT を使用したアルゴリズムの最適化（可能な場合）など、インテル ® IPP ソフトウェアを最大限に活用する方法について説明します。最後に、インテル ® IPP パフォーマンス・テスト・ツールを使用してインテル ® IPP 関数のパフォーマンスをテストする方法と、パフォーマンス・ツールのコマンドライン・オプションの例を説明します。

メモリー・アライメント

インテル ® IPP 関数のパフォーマンスは、データがアライメントされているかどうかによって大幅に変わります。データへのポインターがアライメントされている場合、メモリーへのアクセスは高速です。

ポインターのアライメント、メモリーの割り当ておよび割り当て解除には、以下のインテル ® IPP 関数を使用します。

void* ippAlignPtr(void* *ptr*, int *alignBytes*)

ポインターを 2/4/8/16/... バイト境界にアライメントします。

void* ippMalloc(int *length*)

32 バイト境界にアライメントされたメモリーを割り当てます。メモリーを解放するには、ippFree 関数を使用します。

void ippFree(void* *ptr*)

ippMalloc 関数を使用して割り当てられたメモリーを解放します。

Ipp<datatype>* ippsMalloc_<datatype>(int *len*)

異なるデータタイプの信号要素に、32 バイト境界にアライメントされたメモリーを割り当てます。メモリーを解放するには、ippFree 関数を使用します。

void ippsFree(void* *ptr*)

ippsMalloc 関数を使用して割り当てられたメモリーを解放します。

Ipp<datatype>* ippiMalloc_<mod>(int *widthPixels*, int *heightPixels*, int* *pStepBytes*)

イメージのすべての行がゼロでパディングされているイメージに、32 バイト境界にアライメントされたメモリーを割り当てます。メモリーを解放するには、ippiFree 関数を使用します。

void ippiFree(void* *ptr*)

ippsiMalloc 関数を使用して割り当てられたメモリーを解放します。

例 7-1 は、ippiMalloc 関数の使用方法を示しています。割り当て可能なメモリー量は、オペレーティング・システムとシステムのハードウェアによって決まります。ただし、2GB を超えることはできません。

注 : インテル® IPP メモリー関数は、標準の `malloc` および `free` 関数のラッパーで、インテル® アーキテクチャーで最適なパフォーマンスが得られるように 32 バイト境界にメモリーをアライメントします。

注 : `ippFree`、`ippsFree` および `ippiFree` 関数はそれぞれ、`ippMalloc`、`ippsMalloc` および `ippiMalloc` 関数を使用して割り当てられたメモリーを解放します。

注 : `ippFree`、`ippsFree` および `ippiFree` 関数は、標準の `malloc` および `calloc` 関数を使用して割り当てられたメモリーの解放には使用できません。同様に、`ippMalloc`、`ippsMalloc` および `ippiMalloc` 関数を使用して割り当てられたメモリーを、標準の `free` 関数を使用して解放することはできません。

例 7-1 ippiMalloc 関数の呼び出し

```
#include <stdio.h>
#include "ipp.h"


void ipView(Ipp8u* img, int stride, char* str)
{
   int w, h;
   printf("%s:\n", str);
   Ipp8u* p = img;
   for( h=0; h<8; h++ ) {
      for( w=0; w<8*3; w++ ) {
         printf(" %02X", *(p+w));
      }
      p += stride;
      printf("\n");
   }
}

int main(int argc, char *argv[])
{
   IppiSize size = {320, 240};

   int stride;
   Ipp8u* pSrc = ippiMalloc_8u_C3(size.width, size.height, &stride);
   printf("pSrc=%p, stride=%d\n", pSrc, stride);
   ippiImageJaehne_8u_C3R(pSrc, stride, size);
   ipView(pSrc, stride, "Source image");

   int dstStride;
   Ipp8u* pDst = ippiMalloc_8u_C3(size.width, size.height, &dstStride);
   printf("pDst=%p, dstStride=%d\n", pDst, dstStride);
   ippiCopy_8u_C3R(pSrc, stride, pDst, dstStride, size);
   ipView(pDst, dstStride, "Destination image 1");

   IppiSize ROISize = { size.width/2, size.height/2 };
   ippiCopy_8u_C3R(pSrc, stride, pDst, dstStride, ROISize);
   ipView(pDst, dstStride, "Destination image, small");

   IppiPoint srcOffset = { size.width/4, size.height/4 };
   ippiCopy_8u_C3R(pSrc + srcOffset.x*3 + srcOffset.y*stride, stride, pDst,
dstStride, ROISize);
   ipView(pDst, dstStride, "Destination image, small & shifted");

   ippiFree(pDst);
   ippiFree(pSrc);

   return 0;
}
```

しきい値

デノーマル数は、浮動小数点形式の境界値で、プロセッサーにとって特別な値です。デノーマルデータに対する操作を行うと、対応する割り込みが無効な場合でも処理は遅くなります。デノーマルデータは、例えば、固定小数点形式でキャプチャーして浮動小数点形式に変換した信号を、無限インパルス応答 (IIR) フィルターおよび有限インパルス応答 (FIR) フィルターでフィルタリングした場合に発生します。デノーマルデータの処理による影響を回避するため、インテル® IPP しきい値関数をフィルタリングの前に入力信号に適用することができます。

```
if (denormal_data)
```

```
ippsThreshold_LT_32f_I( src, len, 1e-6f );
ippsFIR_32f( src, dst, len, st );
```

`1e-6` はしきい値レベルです。このレベルよりも小さな入力データはゼロに設定されます。インテル ® IPP しきい値関数は非常に高速なので、ソースデータにデノーマル数がある場合、2 つの関数が実行されることで処理速度は大幅に向上します。当然、フィルタリング中にデノーマル数が発生した場合、しきい値関数では対応できません。

この場合、インテル ® Pentium® 4 プロセッサー以降のインテル ® プロセッサーでは、ゼロフラッシュ（FTZ）とデノーマルゼロ（DAZ）の 2 つの特別な計算モードを設定できます。これらのモードを設定するには、`ippsSetFlushToZero` および `ippsSetDenormAreZeros` 関数を使用します。この設定は、計算がストリーミング SIMD 拡張命令（SSE）およびストリーミング SIMD 拡張命令 2（SSE2）を使用して行われた場合にのみ有効になる点に注意してください。

表 7-1 は、デノーマルデータがパフォーマンスに与える影響と、しきい値を設定した場合の効果を示しています。しきい値を設定した場合、クロック数は 3 クロック増加するだけです。しきい値を設定しない場合、デノーマルデータによりパフォーマンスが約 250 倍遅くなることがわかります。

表 7-1　デノーマルデータにしきい値を設定した場合のパフォーマンス結果

データ / 方法	ノーマル	デノーマル	デノーマル + しきい値
要素ごとの CPU サイクル	46	11467	49

バッファーの再利用

一部のインテル ® IPP 関数では、さまざまな最適化を行うための内部メモリーが必要です。同時に、関数の内部メモリーの割り当ては、キャッシュミスのようにいくつかの状況でパフォーマンスに悪影響を与える可能性があります。メモリー割り当てを回避または最小限にしてデータをホットキャッシュに保つため、一部の関数（例えば、フーリエ変換関数）では、メモリーの利用に関するパラメーターが用意されています。

例えば、FFT 関数を何度も呼び出す必要がある場合、外部バッファーを再利用することでパフォーマンスが向上します。この処理の単純な例として、FFT を使用したフィルタリングと、2 つのスレッドで 2 つの FFT を計算する例を示します。

```
ippsFFTInitAlloc_C_32fc( &ctxN2, order-1, IPP_FFT_DIV_INV_BY_N,
ippAlgHintAccurate );


ippsFFTGetBufSize_C_32fc( ctxN2, &sz );
buffer = sz > 0 ? ippsMalloc_8u( sz ) : 0;


int phase = 0;
/// prepare source data for two FFTs


ippsSampleDown_32fc( x, fftlen, xleft, &fftlen2, 2, &phase );
phase = 1;
ippsSampleDown_32fc( x, fftlen, xrght, &fftlen2, 2, &phase );



ippsFFTFwd_CToC_32fc( xleft, Xleft, ctxN2, buffer );
```

```
ippsFFTFwd_CToC_32fc( xrght, Xrght, ctxN2, buffer );
```

外部バッファーは必要ありません。バッファーのポインターが 0 の場合、関数は内部メモリーを割り当てます。

FFT の使用

高速フーリエ変換（FFT）の使用は、特にフィルタリングが不可欠なデジタル信号処理の分野で、データ処理のパフォーマンスを向上するユニバーサルな方法です。

たたみ込み定理では、空間定義域の 2 つの信号のフィルタリングを周波数定義域の各点乗算として計算できるとしています。周波数定義域間のデータ変換は通常、フーリエ変換を使用して行われます。インテル ® プロセッサー上で非常に高速に動作するインテル ® IPP FFT 関数を使用して、入力信号に有限インパルス応答（FIR）フィルターを適用できます。配列をゼロでパディングしてデータ配列の長さを次の 2 の累乗に増加した後、FFT 順方向変換関数を入力信号と FIR フィルター係数に適用することもできます。この方法で得られたフーリエ係数は、各点乗算されていて、結果は簡単に空間定義域に変換しなおすことができます。FFT の使用によりパフォーマンスは大幅に向上します。

適用するフィルターが複数の反復処理で同じ場合、フィルター係数の FFT 変換は 1 回のみ行う必要があります。回転テーブルとビット反転テーブルは、順方向変換および逆方向変換関数で同時に作成されます。この種のフィルタリングにおける主演算を次に示します。

```
ippsFFTInitAlloc_R_32f( &pFFTSpec, fftord, IPP_FFT_DIV_INV_BY_N,
ippAlgHintNone );

/// perform forward FFT to put source data xx to frequency domain
ippsFFTFwd_RToPack_32f( xx, XX, pFFTSpec, 0 );

/// perform forward FFT to put filter coefficients hh to frequency domain
ippsFFTFwd_RToPack_32f( hh, HH, pFFTSpec, 0 );

/// point-wise multiplication in frequency domain is convolution
ippsMulPack_32f_I( HH, XX, fftlen );

/// perform inverse FFT to get result yy in time domain
ippsFFTInv_PackToR_32f( XX, yy, pFFTSpec, 0 );

/// free FFT tables
ippsFFTFree_R_32f( pFFTSpec );
```

パフォーマンスを大幅に向上させる別の方法として、大規模なサイズのデータ処理に FFT と乗算を使用する方法があります。上記の例におけるゼロは、外部メモリーへのポインターであることに注意してください。これが、パフォーマンスを向上させる別の方法です。インテル ® IPP 信号処理 FIR フィルターは FFT を使用して実装されているため、FIR 関数の特別な実装を作成する必要はありません。

インテル ® IPP パフォーマンス・テスト・ツールの実行

インテル ® Pentium® プロセッサー・ベースの Linux* システム用のインテル ® IPP パフォーマンス・テスト・ツールは、インテル ® IPP ライブラリーを実行するプラットフォームと同じハードウェア・プラットフォーム上で、インテル ® IPP 関数のパフォーマンス・テストを実行するように設計された、非常に機能的なタイミングシステムです。テストツールには、さまざまな方法で各インテル ® IPP 関数のパフォーマンスをテストするコマンドライン・プログラムが含まれています。

コマンドライン・オプションを使用することで、テストの進行を制御し、指定した形式で結果を生成できます。結果は .csv ファイルに保存されます。タイミングの進行はコンソールに表示され、 .txt ファイルに保存できます。テストする関数とパラメーター、およびパフォーマンス・テスト中に呼び出す関数の一覧を作成できます。テストする関数とパラメーターの一覧は、 .ini ファイルで定義するか、コンソールから直接入力します。

列挙モードでは、インテル ® IPP パフォーマンス・テスト・ツールは、コンソールで時間を計測した関数の一覧を作成し、 .txt または .csv ファイルに保存します。

さらに、このツールは、すべてのパフォーマンス・テスト・データを .csv 形式で出力します。このファイルには、インテル ® IPP でサポートされている定義域と CPU タイプをすべてカバーするデータが含まれます。例えば、サブディレクトリー tools/perfsys/data の参照データを読み取ることができます。

インテル ® IPP パッケージをインストールすると、パフォーマンス・テストの ファイルは ia32/tools/perfsys ディレクトリーにインストールされます。例えば、ps_ipps は、インテル ® IPP 信号処理関数のパフォーマンスを測定するツールです。同様に、各インテル ® IPP 関数定義域用の実行ファイルが用意されています。

コマンドラインの形式は次のようになります。

```
<ps_FileName> [switch_1] [switch_2] … [switch_n]
```

コマンドライン・オプションの簡単な説明をコンソールに表示できます。説明を表示するには、コマンドラインで -? または -h と入力します。

```
ps_ipps -h
```

コマンドライン・オプションは、機能別に 6 つのグループに分かれています。複数のオプションを、少なくとも 1 つのスペースを入れて任意の順序で入力できます。一部のオプション（-v、-V、-o、-O など）は、異なるファイル名とともに複数回入力します。-f オプションは、異なる関数パターンで複数回入力します。パフォーマンス・テスト・ツールのコマンドライン・オプションの詳細は、「付録 A - パフォーマンス・テスト・ツールのコマンドライン・オプション」を参照してください

パフォーマンス・テスト・ツールのコマンドラインの例

以下の例では、パフォーマンス・ツールの一般的なコマンドラインを使用して、インテル ® IPP 関数のパフォーマンス・データを生成します。

例 1 - スタンダード・モードでの実行

```
ps_ippch -B -v
```

インテル ® IPP ストリング関数をすべて、デフォルトのタイミング方法を使用して標準データでテストします（-B オプション）。結果はファイル ps_ippch.csv に生成されます（-v オプション）。

例 2 - 選択した関数のテスト

```
ps_ipps -fFIRLMS_32f -v firlms.csv
```

FIR フィルター関数 FIRLMS_32f をテストして（-f オプション）、.csv ファイル firlms.csv を生成します（-v オプション）。

例 3 - 関数リストの取得

`ps_ippvc -e -o vc_list.txt`

ファイル vc_list.txt（-o オプション）にすべてのインテル ® IPP ビデオ・コーディング関数の一覧を出力します（-e オプション)。

`ps_ippvc -e -v H264.csv -f H264`

H264 を含む名前の（-f オプション）関数の一覧をコンソールに表示して（-e オプション)、H264.csv ファイルに保存します（-v オプション)。

例 4 - .ini ファイルを使用したパフォーマンス・テスト・ツールの起動

`ps_ipps -B -I`

最初の実行の後、すべての信号処理関数をテストする .ini ファイル ps_ipps.ini を生成し（-I オプション)、デフォルトのタイミング方法を使用して標準データでテストします（-B オプション）.

`ps_ipps -i -v`

2 回目の実行は、ps_ipps.ini ファイルの手順とパラメーター値で関数をテストして（-i オプション)、出力ファイル ps_ipps.csv（-v オプション）を生成します。

パフォーマンス・テスト・ツールのコマンドライン・オプションの詳細は、「付録 A - パフォーマンス・テスト・ツールのコマンドライン・オプション」を参照してください

各種プログラミング言語でのインテル ® IPP の使用

8

この章では、Windows* OS 環境の異なるプログラミング言語でインテル ® IPP を使用する方法について説明します。また、関連するサンプルについての情報も提供します。

言語サポート

C プログラミング言語に加えて、インテル ® IPP 関数は以下の言語と互換性があります（サンプルは、http://www.intel.com/software/products/ipp/samples.htm からダウンロードしてください）。

表 8-1 言語サポート

言語	環境	サンプルの説明
C++	Makefile、 インテル ® C++ コンパイラー、 GNU C/C++ コンパイラー	インテル ® IPP C ライブラリー関数が C++ インターフェイスで多重定義できることを示し、信号およびイメージ処理用のクラスを作成します。
Fortran	Makefile	N/A
Java*	Java Development Kit 1.5.0	Java ラッパークラスでインテル ® IPP イメージ処理関数を使用します。

Java アプリケーションでのインテル ® IPP の使用

JNI（Java Native Interface）を使用して、Java アプリケーションでインテル ® IPP 関数を呼び出すことができます。JNI の使用により（特に入力データのサイズが小さい場合）、多少のオーバーヘッドが発生します。1 回の JNI 呼び出しで複数の関数を組み合わせて使用し、管理されたメモリーを使用することによって、全体的なパフォーマンス向上が期待できます。

パフォーマンス・テスト・ツールのコマンドライン・オプション

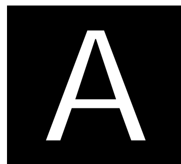


表 A-1 は、利用可能なパフォーマンス・テスト・ツールのコマンドライン・オプションの概要です。

表 A-1　パフォーマンス・テスト・ツールのコマンドライン・オプション

グループ	オプション	説明
1. コンソール入力の調整	-A	すべてのテストの前にパラメーターを確認します。
	-B	バッチモード。
2. 出力の管理	-r[<file-name>]	csv ファイルを作成して PS 結果を書き込みます。
	-R[<file-name>]	テスト結果を csv ファイルに追加します。
	-H[ONLY]	'Interest' 列をテーブルファイルに追加します [ホットテストのみを実行します]。
	-o[<file-name>]	txt ファイルを作成してコンソール出力を書き込みます。
	-O[<file-name>]	コンソール出力を txt ファイルに追加します。
	-L<ERR|WARN|PARM|INFO|TRACE>	コンソール出力のレベルを設定します。
	-u[<file-name>]	csv ファイルを作成してサマリーテーブルを書き込みます（デフォルトのタイトル名に '_sum' が追加されます）。
	-U[<file-name>]	サマリーテーブルを csv ファイルに追加します（デフォルトのタイトル名に '_sum' が追加されます）。
	-e	テストを列挙します。
	-g[<file-name>]	テストの最後に信号ファイルを作成します。
	-s[-]	関数をソートします（デフォルト）。または、ソートしません。
3. テスト用関数の選択	-f < or-pattern>	名前にパターンを含む関数のテストを実行します。大文字と小文字を区別します。
	-f-<not-pattern>	名前にパターンを含む関数のテストを実行しません。大文字と小文字を区別します。
	-f+<and-pattern>	名前にパターンを含む関数のテストのみを実行します。大文字と小文字を区別します。
	-f=< eq-pattern>	パターンと一致する名前の関数のテストを実行します。大文字と小文字を区別します。
	-F<func-name>	指定された名前の関数からテストを開始します。大文字と小文字を区別します。
4. .ini ファイルを使用した操作	-i[<file-name>]	ini ファイルから PS パラメーターを読み取ります。
	-I[<file-name>]	ini ファイルに PS パラメーターを書き込みます。
	-P	ini ファイルからテストした関数名を読み取ります。

表 A-1 パフォーマンス・テスト・ツールのコマンドライン・オプション (続き)

グループ	オプション	説明
5. 入出力用デフォルト・ディレクトリーとファイル名の調整	-n<title-name>	ini ファイルと出力ファイルのデフォルトタイトル名を設定します。
	-p<dir-name>	入力ファイル（ini およびテスト・データ・ファイル）のデフォルト・ディレクトリーを設定します。
	-l<dir-name>	出力ファイルのデフォルト・ディレクトリーを設定します。
6. 直接データ入力	-d<name>=<value>	PS パラメーター値を設定します。
7. プロセスの優先度	-Y<HIGH/NORMAL>	プロセスの優先度を設定します（デフォルトは normal)。
	-T<cpu-name>	ippInitStaticCpu(ippCpu<cpu-name>) を呼び出します。
8. 環境の設定	-N<num-threads>	ippSetNumThreads(<num-treads>) を呼び出します。

インテル ® IPP のサンプル

B

この付録では、開発者が利用可能なインテル ® IPP のサンプルコードをカテゴリー別に紹介します。また、サンプル・アプリケーションのビルド方法および実行方法についても説明します。

インテル ® IPP サンプルコードのタイプ

利用可能なインテル ® IPP サンプルコードのタイプは 3 つあります。これらのタイプはすべて、インテル ® IPP 関数を使用してソフトウェアをビルドする方法を説明します。表 B-1 に、すべてのタイプの一覧を示します。

表 B-1 インテル ® IPP サンプルコードのタイプ

タイプ	説明
アプリケーションレベルのサンプル	これらのサンプルでは、インテル ® IPP API を使用してエンコーダー、デコーダー、ビューアー、プレーヤーなどのさまざまなアプリケーションをビルドする方法を説明します。
ソースコードのサンプル	これらのプラットフォーム別の例では、インテル ® IPP 関数を使用して、パフォーマンス測定、時間定義域フィルタリング、アフィン変換、Canny エッジ検出、その他を実行するための基本的なテクニックを説明します。各サンプルは、1 つから 3 つのソース・コード・ファイル（.cpp）で構成されています。
コード例	コード例（またはコードの一部）は、特定のインテル ® IPP 関数の呼び出し方法を説明する非常に短いプログラムです。多くのコード例が、関数の説明の一部として、『インテル ® IPP リファレンス・マニュアル』（.pdf）に含まれています。

注 : インテル ® IPP サンプルは、異なる開発環境で API を使用してアプリケーションをビルドする方法を説明するために提供されています。

インテル ® IPP サンプルのソースファイル

表 B-2 は、インテル ® IPP サンプルのソースファイルの一覧です。これらのサンプルはすべて Windows* OS 用に作成されていますが、多少修正するだけで Linux* OS でも利用できます。

表 B-2 インテル ® IPP サンプルコードのソースファイル

カテゴリー	サマリー	説明とリンク
基本的な機能	インテル ® IPP 関数を使用したプログラミングの紹介	• パフォーマンス測定 : GetClocks.cpp • データのコピー : Copy.cpp • テーブルベース関数の最適化 : LUT.cpp
デジタル・フィルタリング	信号処理の基本	• DFT の実行 : DFT.cpp • FFT を使用したフィルタリング : FFTFilter.cpp • 時間定義域フィルタリング : FIR.cpp
オーディオ処理	オーディオ信号生成および操作	• DTMF トーンの生成 : DTMF.cpp • IIR を使用したエコーの作成 : IIR.cpp • FIRMR を使用した信号の再サンプリング : Resample.cpp
イメージ処理	イメージ全体またはイメージの一部の作成および処理	• イメージの割り当て、初期化およびコピー : Copy.cpp • 矩形処理サンプルラッパー: ROI.h ROI.cpp ROITest.cpp • マスク・イメージ・サンプル・ラッパー : Mask.h Mask.cpp MaskTest.cpp
イメージ・フィルタリングおよび操作	一般的なイメージアフィン変換	• イメージリサイズ用ラッパー : Resize.h Resize.cpp ResizeTest.cpp • イメージ回転用ラッパー : Rotate.h Rotate.cpp RotateTest.cpp • イメージのアフィン変換実行用ラッパー : Affine.h Affine.cpp AffineTest.cpp
グラフィックスと物理学	ベクトルおよび小行列算術演算関数	• ObjectViewer アプリケーション : ObjectViewerDoc.cpp ObjectViewerDoc.h ObjectViewerView.cpp ObjectViewerView.h 頂点と法線の変換 : CTestView::OnMutateModel オブジェクトの平面への投影 : CTestView::OnProjectPlane カーソル下の三角形の描画 : CTestView::Draw • パフォーマンスの比較、ベクトルとスカラー : perform.cpp • パフォーマンスの比較、バッファーありとバッファーなし : perform2.cpp
特殊目的定義域	暗号化およびコンピューター・ビジョンの使用	• RSA 鍵の生成および暗号化 : rsa.cpp rsa.h rsatest.cpp bignum.h bignum.cpp • Canny エッジ検出クラス : canny.cpp canny.h cannytest.cpp filter.h filter.cpp • ガウシアン角錐クラス : pyramid.cpp pyramid.h pyramidtest.cpp

インテル ® IPP サンプルの使用

http://www.intel.com/software/products/ipp/samples.htm からインテル® IPP サンプルをダウンロードしてください。

これらのサンプルはインテル ® IPP の各バージョンで更新されています。インテル ® IPP の新しいバージョンがあれば、インテル ® IPP サンプルをアップグレードすることを推奨します。

動作環境

サンプルのシステム要件は、各サンプルのルート・ディレクトリーにある *readme.htm* ドキュメントを参照してください。最も一般的な要件を次に示します。

ハードウェア要件 :

- インテル ® Pentium® プロセッサー、インテル ® Xeon® プロセッサー、またはその他の IA-32 アーキテクチャー対応プロセッサー・ベースのシステム

ソフトウェア要件 :

- インテル ® IPP 6.0 Linux* 版
- Red Hat* Enterprise Linux* オペレーティング・システム 3.0 以上
- インテル ® C++ コンパイラー 10.1、10.0 または 9.1 Linux* 版、GNU C/C++ コンパイラー 3.2 以上
- Qt* ラ C ブラリー・ランタイムおよび開発環境

ソースコードのビルド

サンプルのビルド方法は、各サンプルの *readme.htm* ドキュメントを参照してください。最も一般的な手順を次に示します。

インテル ® IPP のルート・ディレクトリーを指す環境変数 `IPPROOT` を作成して、ビルド環境を設定します。

サンプルをビルドするには、サンプルのルートフォルダーに移動して、シェルスクリプト `build32.sh` [`option`] を実行します。

デフォルトでは、スクリプトは以下の表に従ってコンパイラーを（デフォルトのディレクトリーにインストールされていると仮定して）検索します。インテル ® C++ コンパイラーまたは GCC ソフトウェアを使用する場合、以下の表に従って、スクリプトオプションを設定してください。

表 B-3 バッチファイルのオプション

コンパイラー	スクリプトオプション
インテル ® C++ コンパイラー 10.1 Linux 版	`icc101`
インテル ® C++ コンパイラー 10.0 Linux 版	`icc10`
インテル ® C++ コンパイラー 9.1 Linux 版	`icc91`
GCC 4.x.x	`gcc4`
GCC 3.4.x	`gcc3`

ビルドが成功すると、ファイルが対応するサンプル・ディレクトリー
<install_dir>/ipp-samples/<sample-name>/bin/linux32_<compiler> に生成されます。

compiler は、icc101|icc10|icc91|gcc3|gcc4 のいずれかです。

ソフトウェアの実行

各サンプル・アプリケーションを実行するには、システムのパスにインテル ® IPP 共有オブジェクト・ライブラリーが必要です。詳細は、「環境変数の設定」を参照してください。

アプリケーションの実行方法、コマンドライン・オプションおよびメニューコマンドに関する詳細は、各サンプルの *readme.htm* ドキュメントを参照してください。

既知の制限事項

インテル ® IPP サンプルを使用して作成されたアプリケーションは、インテル ® IPP 関数の使用方法を示し、開発者が各自のアプリケーションを作成する際の参考となるように意図された例です。これらのサンプル・アプリケーションには、各サンプルの *readme.htm* ドキュメントの「既知の制限事項」に記述されている制限があります。

索引

E

Eclipse の設定 , 4-1

F

FFT の使用 , 7-5

J

java アプリケーション , 8-1

O

OpenMP のサポート , 6-1

あ

アプリケーションのビルド , 2-1

い

インストールの確認 , 2-1

インテル ® IPP, 1-1

インテル ® IPP と Java の使用 , 8-1

インテル ® IPP の使用
- Visual C++, 4-2
- Visual C++.NET, 4-2
- プログラミング言語 , 8-1

さ

サンプル , B-1
- タイプ , B-1

サンプルのビルド , B-3

サンプルの実行 , B-3

サンプルの実行における既知の制限 , B-4

し

しきい値データ , 7-3

す

スレッディング , 6-1

スレッド数の制御 , 6-1

そ

ソースコードのサンプル , B-1

て

ディスパッチ , 5-1

テクニカルサポート , 1-1

と

ドキュメント
- ファイル名 , 3-3
- 構成 , 1-3
- 場所 , 3-1
- 対象者 , 1-2
- 目的 , 1-2

ドメイン別のライブラリーの依存関係 , 5-9

は

バージョン情報 , 2-1

バッファーの再利用 , 7-4

パフォーマンスの管理 , 7-1
- FFT の使用 , 7-5
- しきい値データ , 7-3
- バッファーの再利用 , 7-4
- メモリー・アライメント , 7-1

パフォーマンス・テスト・ツール , 7-6
- コマンドラインの例 , 7-6
- コマンドライン・オプション , A-1

ふ

プロセッサーのタイプの検出 , 5-1

プロセッサー固有のコード , 5-1

へ

ヘッダーファイル , 2-2

ま

マルチスレッディングの無効化 , 6-2

め

メモリー・アラインメント , 7-1

り

リンク
カスタム・ダイナミック , 5-6
スタティック、ディスパッチあり , 5-3
スタティック、ディスパッチなし , 5-5
ダイナミック , 5-3
リンクモデル , 5-2
リンクモデルの選択 , 5-2
リンクモデルの比較 , 5-7
リンク例 , 5-10

ん

環境変数の設定 , 2-2
関数の呼び出し , 2-2
言語サポート , 8-1
構成
ドキュメント・ディレクトリー , 3-3
ライブラリーのタイプ別 , 3-2
高レベル・ディレクトリー , 3-1
使用
DLL, 3-2
スタティック・ライブラリー , 3-3
使用方法 , 1-1
選択
ライブラリー , 5-7
提供ライブラリー , 3-2
表記規則 , 1-4