

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

ユーザーズガイド

2009 年 3 月

資料番号 : 320271-003JA

Web サイト : <http://developer.intel.com> (英語)



バージョン	バージョン情報	日付
-001	インテル® インテグレートッド・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
-002	インテル® IPP 6.1 ベータ版リリースのドキュメント。	2009 年 1 月
-003	インテル® IPP 6.1 リリースのドキュメント。	2009 年 3 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、Itanium、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。

© 2008-2009 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
	本書について	1-2
	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	
	インテル® IPP について	2-1
	クロスアーキテクチャーの統一	2-1
	入力データの型	2-2
	ドメイン	2-3
	関数の命名規則とパラメーター	2-4
	インストールの確認	2-7
	バージョン情報の取得	2-7
	アプリケーションのビルド	2-7
	環境変数の設定	2-7
	ヘッダーファイルのインクルード	2-8
	IPP 関数の呼び出し	2-8
	インテル® IPP の使用を開始する前に	2-9
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
	プロセッサのタイプと機能	5-2
	リンク方法の選択.....	5-3
	ダイナミック・リンク	5-4
	スタティック・リンク (ディスパッチあり).....	5-4
	スタティック・リンク (ディスパッチなし).....	5-6
	カスタム SO のビルド	5-7
	インテル® IPP のリンク方法の比較.....	5-8
	アプリケーションに必要なインテル® IPP ライブラリーの選択.....	5-8
	ダイナミック・リンク	5-9
	スタティック・リンク (ディスパッチあり).....	5-10
	ドメイン別のライブラリー依存関係 (スタティック・リンクのみ).....	5-10
	リンク例.....	5-11
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
	バッファの再利用	7-4
	FFT の使用	7-4
	インテル® IPP パフォーマンス・テスト・ツールの実行	7-5
	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
	ソースコードのサンプル	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® インテグレートッド・パフォーマンス・プリミティブ (インテル® IPP) は、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。この機能には、一般的な信号処理とイメージ処理、コンピューター・ビジョン、音声認識、データ圧縮、暗号化、ストリング操作、オーディオ処理、ビデオ・コーディング、リアリスティック・レンダリングおよび 3D データ処理が含まれます。また、MP3 (MPEG-1 オーディオ、レイヤー 3)、MPEG-4、H.264、H.263、JPEG、JPEG2000、GSM-AMR、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も含まれています。

インテル® IPP ライブラリーは、各関数でさまざまなデータ型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設計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 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、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® OS 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® OS 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® OS 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・ファミリー用開発パッケージ - インテル® IPP Linux® OS 版 (IA-64 アーキテクチャー用)
- インテル® Atom™ プロセッサー用開発パッケージ - インテル® IPP Linux® OS 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® IPP の製品 Web サイトでは、製品の機能、ホワイトペーパー、技術資料を含む、タイムリーで包括的な製品情報を提供しています。最新情報は、<http://developer.intel.com/software/products/> を参照してください。

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、その他の多くのセルフヘルプ情報を含むサポート Web サイト (<http://support.intel.co.jp/>) も提供しています。

製品を登録すると、インテル® プレミアサポートを利用して、1 年間のテクニカルサポートと製品アップデート・サービスを受けることができます。インテル® プレミアサポートは、次のサービスを提供する、対話型の問題管理およびコミュニケーション Web サイトです。

- 問題の報告とそのステータスの確認。
- 製品アップデートのダウンロード。

製品の登録、インテルへの問い合わせ、製品サポート情報の検索は、次のサイトにアクセスしてください。

<http://www.intel.com/software/products/support/ipp/>

本書について

このユーザーズガイドは、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方法を説明します。また、プラットフォーム固有の機能に加えて、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。

このガイドは、<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

本書では、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行う手順
- 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
- 最適な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
- 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

本書は、ソフトウェア開発の初心者から熟練者までを含む、Linux プログラマーを対象としています。

ドキュメントの構成

本書には、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・ライブラリーを構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示します。 |
| 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、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、サンプルを実行する方法について説明します。 |

本ガイドには、[索引](#)も含まれています。

表記規則

本書では、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、強調および文書名を示します。
等幅の小文字	ファイル名、ディレクトリー名およびパス名を示します。 例： 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。 例： 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 (<i>lda</i> など) やメイクファイル・パラメーター (<i>functions_list</i> 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 ディレクトリー>.
[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行う必要がある手順について説明します。

インテル® IPP について

インテル® IPP は、広範囲な機能を提供するハイパフォーマンス・コードのコレクションです。この機能には、一般的な信号処理とイメージ処理、コンピューター・ビジョン、音声認識、データ圧縮、暗号化、ストリング操作、オーディオ処理、ビデオ・コーディング、リアリスティック・レンダリングおよび 3D データ処理、行列演算が含まれます。また、MP3 (MPEG-1 オーディオ、レイヤー 3)、MPEG-4、H.264、H.263、JPEG、JPEG2000、GSM-AMR、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も含まれています。

インテル® IPP は、インテル® パフォーマンス・ライブラリーの開発と使用の経験に基づく、以下の特長を備えています。

- インテル® IPP は、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、イメージ処理、ビデオ・コーディング、小行列演算、リアリスティック・レンダリング、3D データ処理のような、さまざまな異なるドメインでアプリケーションを作成するための基本的な低レベルの関数を提供します。詳細は、「[ドメイン](#)」セクションを参照してください。
- インテル® IPP 関数は、同じインターフェイス規則に従っており、異なるアプリケーション・ドメインを参照するプリミティブに対して、共通の命名規則と同様の構成のプロトタイプを使用します。関数の命名規則については、「[関数の命名規則とパラメーター](#)」を参照してください。
- インテル® IPP 関数は、アプリケーション・プログラムで優れた性能を発揮するのに最適な抽象レベルを使用しています。

プログラムのパフォーマンスを向上させるために、インテル® IPP 関数は、インテル® アーキテクチャ・プロセッサの利点をすべて活用するように最適化されています。また、ほとんどのインテル® IPP 関数は、複雑なデータ構造を使用しないため、全体的な実行のオーバーヘッドを減らすことができます。

インテル® IPP は、クロスプラットフォーム・アプリケーションに最適です。例えば、IA-32 アーキテクチャー・ベースのプラットフォーム用に開発された関数を、Itanium® ベースのプラットフォームに簡単に移植できます（「[クロスアーキテクチャーの統一](#)」を参照）。

クロスアーキテクチャーの統一

インテル® IPP は、さまざまなインテル® アーキテクチャーにおけるアプリケーション開発をサポートするように設計されています。根本的な関数の実装ではプロセッサ・アーキテクチャーによる違いを考慮する一方、API 定義はすべてのプロセッサで共通です。

単一のクロスアーキテクチャー API が提供されているので、開発者は、インテル® プロセッサー・ベースのデスクトップ、サーバー、モバイルなどのさまざまなプラットフォーム用のソフトウェア・アプリケーションを開発することができます。開発者はコードを一度記述するだけで、多くのプロセッサーでアプリケーションのパフォーマンスを最適化することができます。

入力データの型

インテル® IPP の演算は、演算が実行される入力データの型に従って、以下の 3 つのグループに分けられています。

1 次元配列と信号

このグループには、1 次元配列のデータを演算するほとんどの関数が含まれます。多くの場合、これらの配列は信号で、ほとんどの演算は信号処理演算です。

1 次元配列演算の例：

- ベクトル化されたスカラー算術、論理、統計演算
- デジタル信号処理
- データ圧縮
- オーディオ処理とオーディオ・コーディング
- 音声認識と音声コーディング
- 暗号化とデータ完全性
- ストリング処理

イメージ

イメージは、ピクセルの 2 次元配列です。イメージには、一般的な 2 次元配列と区別されるいくつかの特徴があります。

イメージ演算の例：

- 算術、論理、統計演算
- カラー変換
- イメージ・フィルタリング
- イメージの 1 次変換とジオメトリ変換
- モルフォロジー演算
- コンピューター・ビジョン
- イメージ圧縮
- ビデオ・コーディング

行列

このグループには、1 次元配列と 2 次元配列の行列とベクトル、および行列とベクトルの配列の演算を行う関数が含まれます。これらの配列は線形方程式またはデータベクトルとして処理され、線形代数演算が行われます。

行列演算の例：

- ベクトルと行列代数
- 線形方程式系の解法
- 最小二乗問題の解法
- 固有値問題の計算

3D オブジェクト

このグループには、3D オブジェクトの演算関数が含まれます。入力データは使用する手法に依存します。

3D 演算の例：

リアリスティック・レンダリング
リサイズとアフィン変換

インテル® IPP の関数は、上記にリストされている入力データの型に従ってグループ化されています。グループごとに、独自の接頭辞が関数名に付けられます (詳細は「関数の命名規則」を参照)。

コア関数

インテル® IPP の一部の関数は、入力データの型に応じた処理を行いません。これらの関数は、システムの検出と設定、およびインテル® IPP の構成に使用されます。例えば、CPU タイプの取得、指定されたバイト数へのポインターのアライメント、マージド・スタティック・ライブラリーのディスパッチャーの制御、その他の処理を行います。これらの関数はコア関数と呼ばれ、独自のヘッダーファイル、スタティック・ライブラリー、SO が用意されています。

表 2-1

コード	ヘッダー ファイル	スタティック・ ライブラリー	SO	関数名の 接頭辞
ippCore	ippcore.h	libippcore.a libippcore_t.a	libippcore.so.*.*	ipp

. は製品のバージョン番号。例：6.1

ドメイン

インテル® IPP では、関連する関数を内部的にいくつかの細区分に分けています。細区分はそれぞれドメイン (または関数ドメイン) と呼ばれ、一般に、ヘッダーファイル、スタティック・ライブラリー、DLL、およびテストが含まれます。これらのドメインには、入力データの型と接頭辞が対応付けられています。『インテル® IPP リファレンス・マニュアル』では、関数が含まれているヘッダーファイルを示します。以下の表は、各ドメインのコード、ヘッダーファイルとライブラリー名、関数エリアの一覧です。

表 2-2

コード	ヘッダー ファイル	スタティック・ ライブラリー	SO	接頭辞	説明
ippAC	ippac.h	libippac*.a	libippac***.so.***	ipps	オーディオ・コーディング
ippCC	ippcc.h	libippac*.a	libippac***.so.***	ippi	カラー変換
ippCH	ippch.h	libippac*.a	libippac***.so.***	ipps	ストリング処理
ippCP	ippcp.h	libippac*.a	libippac***.so.***	ipps	暗号化
ippCV	ippcv.h	libippac*.a	libippac***.so.***	ippi	コンピューター・ビジョン
ippDC	ippdc.h	libippac*.a	libippac***.so.***	ipps	データ圧縮
ippDI	ippdi.h	libippac*.a	libippac***.so.***	ipps	データ完全性
ippGEN	ipps.h	libippac*.a	libippac***.so.***	ippg	生成関数

表 2-2

コード	ヘッダー ファイル	スタティック・ ライブラリー	SO	接頭辞	説明
ippIP	ippi.h	libippac*.a	libippac**. so.**	ippi	イメージ処理
ippJP	ippj.h	libippac*.a	libippac**. so.**	ippi	イメージ圧縮
ippMX	ippm.h	libippac*.a	libippac**. so.**	ippm	小行列演算
ippRR	ippr.h	libippac*.a	libippac**. so.**	ippr	リアリスティック・レンダリング および 3D データ 処理
ippSC	ippsc.h	libippac*.a	libippac**. so.**	ipps	音声コーディング
ippSP	ipps.h	libippac*.a	libippac**. so.**	ipps	信号処理
ippSR	ippsr.h	libippac*.a	libippac**. so.**	ipps	音声認識
ippVC	ippvc.h	libippac*.a	libippac**. so.**	ippi	ビデオ・コーディ ング
ippVM	ippvm.h	libippac*.a	libippac**. so.**	ipps	ベクトル演算

* - 次のいずれか:emerged、merged、merged_t

** - プロセッサ固有のコード。例:s8

*** - バージョン番号。例:6.1

関数の命名規則とパラメーター

インテル® IPP の関数名は、簡単に識別して利用できるように構成されています。インテル® IPP の命名規則を理解すると、特定の関数の目的を調べる時間や労力を大幅に減らすことができます。多くの場合、関数の名前からその関数の基本的な情報を得られます。

インテル® IPP 関数の命名規則は、すべてのドメインで共通です。

インテル® IPP の関数名には、データドメイン、演算、データ型、実行モードを示す複数のフィールドが含まれます。フィールドにはそれぞれ、事前に定義された値のみ指定できます。

関数名の一般的な形式は次のようになります。

```
ipp<data-domain><name>[_<datatype>][_<descriptor>](<parameters>);
```

この形式の要素については、次のセクションで説明します。

データドメイン

data-domain は、入力データの型を示す 1 文字です。

インテル® IPP の現在のバージョンでは、以下のデータドメインをサポートしています。

- s 信号 (想定されるデータ型は 1D 配列)
- g 固定長の信号 (想定されるデータ型は 1D 配列)
- i イメージおよびビデオ (想定されるデータ型はピクセルの 2D 配列)
- m ベクトルおよび行列 (想定されるデータ型は行列またはベクトル)
- r リアリスティック・レンダリングおよび 3D データ処理 (想定されるデータ型はサポートしているレンダリング手法に依存)

インテル® IPP のコア関数は、入力データの型に応じた処理を行いません (「[コア関数](#)」を参照)。これらの関数には、接頭辞として `ipp` が付けられ、データドメイン・フィールドはありません (例: `ippGetStatusString`)。

名前

name は、関数が実行するアルゴリズムまたは演算を示します。形式は次のようになります。

`<name> = <operation>[_modifier]`

operation フィールドは、基本演算を示す 1 つ以上の単語、頭文字、および省略形です (例: `Set`, `Copy`)。フィールドが複数の部分からなる場合、各部分はそれぞれ大文字で始まり、下線は含まれません (例: `HilbertInitAlloc`)。

modifier が存在する場合、指定された関数のバリエーションを示します。例えば、`ippsFFTInv_CToC_32fc` 関数の `CToC` 修飾子は、複素数データを使用して逆高速フーリエ変換を行い、複素数から複素数の (`CToC`) 変換を実行することを意味します。行列演算の関数にはオブジェクト・タイプの説明が修飾子として含まれます (例えば、`ippmMul_mv` は行列とベクトルの乗算です)。

データ型

datatype フィールドは、関数で使われるデータ型を示します。形式は次のようになります。

`<datatype> = <bit_depth><bit_interpretation>`,

ここで、

`bit_depth = <1|8|16|32|64>`

また

`bit_interpretation = <u|s|f>[c]`

u は符号なし整数、*s* は符号付き整数、*f* は浮動小数点、

c は複素数を示します。

単一のデータ型で演算を行う関数の場合、*datatype* フィールドには 1 つの値のみ含まれます。

関数でデータ型の異なるソース・オブジェクトとデスティネーション・オブジェクトの演算を行う場合、次のように、データ型の識別子がソース、デスティネーションの順で関数名に記述されます。

`<datatype> = <src1Datatype>[src2Datatype][dstDatatype]`

例えば、`ippsDotProd_16s16sc` 関数は 16 ビット short ソースベクトルと 16 ビット complex short ソースベクトルのドット積を計算して、16 ビット complex short デスティネーション・ベクトルに格納します。2 番目のオペランドと結果が同じ型なので、関数名に *dstDatatype* 修飾子は含まれていません。

記述子

オプションの *descriptor* フィールドには、演算に関連するデータが記述されます。このフィールドには、暗黙のパラメーターや追加の必須パラメーターを記述できます。

関数内のコード分岐の数を最小限に抑え、不要な実行のオーバーヘッドを減らすため、ほとんどの汎用関数はプリミティブ関数に分割されます。パラメーターの一部は、記述子としてプリミティブ関数の名前に含まれます。

ただし、関数の置換数が多すぎる場合、一部の関数では、内部演算を指定するパラメーターを使用できます (例: `ippiThreshold`)。

インテル® IPP では、以下の記述子を使用されています。

- A イメージデータは最後のチャンネルとしてアルファチャンネルを含みます。C4 が必要です。アルファチャンネルは処理されません。

A0	イメージデータは最初のチャンネルとしてアルファチャンネルを含みます。C4 が必要です。アルファチャンネルは処理されません。
Axx	高度な算術演算の結果の精度をビット数で指定します。
C	関数は各ソースイメージ用に指定された対象チャンネル (COI) で演算を行います。
Cn	イメージデータは n 個の離散インターリーブ・チャンネル (n= 1, 2, 3, 4) で構成されます。
Dx	信号は x 次元です (デフォルトは D1)。
I	演算をインプレースで実行します (デフォルトはノット・インプレース)。
L	行列演算のオブジェクトのレイアウト記述子、または 1 つのポインターが信号処理の D2 配列の各行で使用されることを示します。
M	演算はマスクを使用して処理するピクセル数を決定します。
P	行列演算のオブジェクトのポインター記述子、または信号処理で処理されるベクトルの数。
Pn	イメージデータは、n 個の離散プレーン (非インターリーブ) チャンネル (n=1, 2 3, 4) と各プレーンへのポインタで構成されます。
R	関数は各ソースイメージ用に定義された対象領域 (ROI) で演算を行います。
S	行列演算用オブジェクトの標準記述。
Sfs	飽和処理を行い、固定スケーリング・モードを使用します (デフォルトは飽和処理を行い、スケーリングしません)。
s	飽和処理を行い、スケーリングしません。

関数名の中では、記述子は常にアルファベット順で記述します。

特定の演算では、一部のデータ記述子が暗黙的に含まれます。例えば、イメージ処理関数はデフォルトで 2 次元イメージを演算し、結果をスケーリングしないで飽和処理を行います。この場合、暗黙的な省略形 D2 (2 次元信号) と s (飽和処理を行ってスケーリングしない) は関数名に含まれません。

パラメーター

parameters フィールドには、関数のパラメーター (引数) を指定します。

パラメーターの順序は以下のとおりです。

1. すべてのソースオペランド。配列の後に定数が続きます。
2. すべてのデスティネーション・オペランド。配列の後に定数が続きます。
3. その他、演算固有のパラメーター。

パラメーター名には以下の規則があります。

ポインターとして定義された引数は *p* で始まります (例: *pPhase*、*pSrc*、*pSeed*)。ダブルポインターとして定義された引数は *pp* で始まります (例: *ppState*)。値として定義された引数は小文字で始まります (例: *val*、*src*、*srcLen*)。

引数名の新しい部分は大文字で始まり、下線は含まれません (例: *pSrc*、*lenSrc*、*pDlyLine*)。

引数名には、その機能を指定します。ソース引数の名前は *pSrc* や *src* のようになります。後に名前や数字が続くこともあります (例: *pSrc2*、*srcLen*)。出力引数の名前は、*pDst* や *dst* の後に名前や数字が続きます (例: *pDst1*、*dstLen*)。インプレース演算の場合は、入力 / 出力引数に *pSrcDst* が含まれます。

関数構文の例:

```
ippSIIR_32f_I(Ipp32f* pSrcDst, int len, IppSIIRState_32f* pState);
ippiConvert_8ulu_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, int dstBitOffset, IppiSize roiSize, Ipp8u threshold);
```

```
ippmSub_vac_32f(const Ipp32f* pSrc, int srcStride0, int srcStride2,
Ipp32f val, Ipp32f* pDst, int dstStride0, int dstStride2, int len, int
count).
```

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、以下の手順に従って、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 ディレクトリー > //ia32 が作成されていることを確認します。デフォルトのインストール・ディレクトリーは、
/opt/intel/Compiler/11.1/xxx/ipp//ia32 です。
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、ユーザーシェルで環境変数 LD_LIBRARY_PATH、LIB および INCLUDE を設定します。
3. ディスパッチおよびプロセッサ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。
4. 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 の関数とタイプは、関数ドメイン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、以下の手順を行います。

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、いくつかの基本的な概念を理解しておくに役立ちます。

[表 2-3](#) は、インテル® IPP の使用を開始する前に知っておくべき項目を要約したものです。

表 2-3 **開始前に知っておくべき項目**

関数ドメイン	<p>用途に応じたインテル® IPP 関数ドメインを識別します。</p> <p>理由: 使用する関数ドメインを知っておくと、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。</p> <p>http://www.intel.com/software/products/ipp/samples.htm (英語) のサンプルも参照してください。</p> <p>関数ドメインと必要なライブラリーについては、表 5-10 を参照してください。また、クロスドメインの依存関係については、表 5-10 を参照してください。</p>
リンク方法	<p>適切なリンク方法を決定します。</p> <p>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。</p>
スレッディング・モデル	<p>以下のオプションを選択して、アプリケーションをどのようにスレッド化するかを決定します。</p> <ul style="list-style-type: none"> 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、OpenMP* 互換ランタイム・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 <p>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。異なる数が必要な場合、プログラマーが数を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。</p>

インテル® IPP の構造

この章は、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 ディレクトリー>	メイン・ディレクトリー (デフォルトの場合、 /opt/intel/Compiler/11.1/xxx/ipp/ia32)
<ipp ディレクトリー>/ippEULA.txt	インテル® IPP のエンド・ユーザー・ソフトウェア使用許諾契約書
<ipp ディレクトリー>/doc	インテル® IPP ドキュメント・ファイル
<ipp ディレクトリー>/include	インテル® IPP ヘッダーファイル
<ipp ディレクトリー>/lib	インテル® IPP スタティック・ライブラリー
<ipp ディレクトリー>/sharedlib	インテル® IPP 共有オブジェクト・ライブラリー
<ipp ディレクトリー>/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・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.6.1, libippst7.so.6.1
	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を含む		
	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 システム変数にライブラリーのパスを追加します。

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 /opt/intel/Compiler/11.1/xxx/ipp/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=
/opt/intel/Compiler/11.1/xxx/ipp/ia32/sharedlib:$LD_LIBRARY_PATH
```

(csh の場合)

```
setenv LD_LIBRARY_PATH=
/opt/intel/Compiler/11.1/xxx/ipp/ia32/sharedlib:${LD_LIBRARY_PATH}
```

共有ライブラリー libipp*.so.6.1 (* は適切な関数ドメインを示す) 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を検出して、適切なプロセッサ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を考慮することなく、インテル® IPP 関数を呼び出すコードを記述することができます。これらのプロセッサ固有ライブラリーには、libipp*px.so.6.1、libipp*w7.so.6.1、libipp*t7.so.6.1、libipp*v8.so.6.1、および libipp*p8.so.6.1 のように名前が付けられています (表 5-4 を参照)。例えば、ia32/sharedlib ディレクトリーの libippiv8.so.6.1 は、インテル® Core™ 2 Duo プロセッサ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*-6.1.so、libipp*px-6.1.so、libipp*w7-6.1.so、libipp*t7-6.1.so、libipp*v8-6.1.so、および libipp*p8-6.1.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。

「[アプリケーションに必要なインテル® 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 が 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

インテル® IPP は、各関数のすべてのプロセッサ・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、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。

「[アプリケーションに必要なインテル® 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の内容です。

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.pdf	製品の概要および本リリースについての情報	
README.txt	初期ユーザー情報	このファイルは、製品のインストール前に表示できます。
INSTALL.htm	インストール・ガイド	

3 インテル® IPP ユーザーズガイド

表 3-3 /doc ディレクトリーの内容

ファイル名	説明	注
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_lnx_ia32.pdf	本ドキュメント - インテル® インテグレートッド・パフォーマンス・プリミティブ ユーザーガイド	
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、データ圧縮と完全性、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、線型方程式系、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

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 / コンパイラーの統合で 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、
`/opt/intel/Compiler/11.1/xxx/ipp/ia32/include` (xxx はパッケージ番号) を設定します。
2. ツールチェーン / コンパイラーの統合でライブラリー・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、`/opt/intel/Compiler/11.1/xxx/ipp/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「[アプリケーションに必要なインテル® 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際の項目設定の名前は、コンパイラーの統合に依存します。

コンパイラー / リンカーは、自動メイクファイル生成が有効な場合のみ、インクルードとライブラリーのパス設定を自動的に選択することに注意してください。有効でない場合、メイクファイルにインクルードとライブラリーの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、
`/opt/intel/Compiler/11.1/xxx/ipp/ia32/include` (xxx はパッケージ番号) 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「[アプリケーションに必要なインテル® 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 / リンカーはこれらの情報を自動的に取得しないため、メイクファイル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すべて、このページに含まれています。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、ここでは明記していません。
2. コンパイラーの統合でインクルード・パス・オプションをサポートしている場合、インテル® IPP の include パスを設定します。例えば、デフォルトの場合、
`/opt/intel/Compiler/11.1/xxx/ipp/ia32/include` を設定します。
3. コンパイラーの統合でライブラリー・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、`/opt/intel/Compiler/11.1/xxx/ipp/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「[アプリケーションに必要なインテル® 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



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、さまざまなプロセッサ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、プロセッサを検出して、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.6.1` は、インテル® Core™2 Duo プロセッサ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向けのインテル® プロセッサ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固有のコードを示しています。

表 5-1 プロセッサ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 (インテル® SSE2) 対応プロセッサ向けに最適化
t7	インテル® ストリーミング SIMD 拡張命令 3 (インテル® SSE3) 対応プロセッサ向けに最適化
v8	インテル® ストリーミング SIMD 拡張命令 3 補足命令 (インテル® SSSE3) 対応プロセッサ向けに最適化
p8	インテル® ストリーミング SIMD 拡張命令 4.1 (インテル® SSE4.1) 対応プロセッサ向けに最適化
s8	インテル® Atom™ プロセッサ向けに最適化

プロセッサのタイプと機能

プロセッサの機能

コンピューター・システムで使用しているプロセッサの機能についての情報を取得するには、ippcore.h ファイルで宣言されている `ippGetCpuFeatures` 関数を使用します。この関数は、`CPUID.1` によって返される主なプロセッサの機能を取得して、関数によって返されるマスクに格納します。[表 5-2](#) は、取得可能なプロセッサの機能の一覧です (詳細は、『インテル® IPP リファレンス・マニュアル』(第 1 巻) の `ippGetCpuFeatures` 関数の説明を参照してください)。

表 5-2 プロセッサの機能

マスク値	名前	機能
1	<code>ippCPUID_MMX</code>	MMX®テクノロジー
2	<code>ippCPUID_SSE</code>	インテル® ストリーミング SIMD 拡張命令
4	<code>ippCPUID_SSE2</code>	インテル® ストリーミング SIMD 拡張命令 2
8	<code>ippCPUID_SSE3X</code>	インテル® ストリーミング SIMD 拡張命令 3
16	<code>ippCPUID_SSSE3</code>	インテル® ストリーミング SIMD 拡張命令 3 補足命令
32	<code>ippCPUID_MOVBE</code>	MOVBE 命令をサポート
64	<code>ippCPUID_SSE41</code>	インテル® ストリーミング SIMD 拡張命令 4.1
128	<code>ippCPUID_SSE42</code>	インテル® ストリーミング SIMD 拡張命令 4.2
256	<code>ippCPUID_AVXX</code>	インテル® Advanced Vector Extensions (インテル® AVX) 命令セットをサポート
512	<code>ippAVX_ENABLEDBYOS</code>	インテル® AVX をサポートするオペレーティング・システム
1024	<code>ippCPUID_AES</code>	AES 命令をサポート
2048	<code>ippCPUID_CLMUL</code>	PCLMULQDQ 命令をサポート

プロセッサのタイプ

コンピューター・システムで使用しているプロセッサ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 プロセッサを使用していることを意味します。

[表 5-3](#) は、`ippGetCpuType` の戻り値とその意味を示しています。

表 5-3 プロセッサのタイプの検出 戻り値とその意味

戻り値	プロセッサのタイプ
<code>ippCpuPP</code>	インテル® Pentium® プロセッサ
<code>ippCpuPMX</code>	MMX®テクノロジー インテル® Pentium® プロセッサ
<code>ippCpuPPR</code>	インテル® Pentium® Pro プロセッサ
<code>ippCpuPII</code>	インテル® Pentium® II プロセッサ
<code>ippCpuPIII</code>	インテル® Pentium® III プロセッサおよびインテル® Pentium® III Xeon® プロセッサ
<code>ippCpuP4</code>	インテル® Pentium® 4 プロセッサおよびインテル® Xeon® プロセッサ
<code>ippCpuP4HT</code>	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
<code>ippCpuP4HT2</code>	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

表 5-3 プロセッサのタイプの検出 戻り値とその意味 (続き)

戻り値	プロセッサのタイプ
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
ippCpuCoreDuo	インテル® Core™ Duo プロセッサ
ippCpuITP	インテル® Itanium® プロセッサ
ippCpuITP2	インテル® Itanium® 2 プロセッサ
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
ippCpuC2Q	インテル® Core™2 Quad プロセッサ
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
ippCpuBonnell	インテル® Atom™ プロセッサ
ippCpuNehalem	インテル® Core™ i7 プロセッサ
ippCpuSSE	インテル® ストリーミング SIMD 拡張命令対応プロセッサ
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
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
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
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
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
ippCpuAVX	インテル® Advanced Vector Extensions 命令セット対応プロセッサ
ippCpuX8664	64 ビット拡張命令対応プロセッサ
ippCpuUnknown	不明なプロセッサ

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 [(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の考慮事項は次のとおりです。

- 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のか？それとも常にアプリケーションと一緒に含まれるのか？

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 5-4 ダイナミック・リンクの機能の要約

長所	短所
<ul style="list-style-type: none"> • プロセッサ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 / 再リンクせずに新しいプロセッサ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 	<ul style="list-style-type: none"> • 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 アプリレット / 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、以下の手順に従います。

1. すべての IPP ドメイン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. 対応するドメイン・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・フットプリント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、`libippsemmerged.a`) は、非修飾の (標準名の) IPP 関数のエントリポイントと、各プロセッサ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` の関数で検出されたプロセッサの設定に従ってマージド・ライブラリー (例えば、`libippsmerged.a`) 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()` 関数を使用してプロセッサ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、アプリケーションのパフォーマンスが低下します。例 5-1 に、パフォーマンスの違いを示します。この例は、t2.cpp ファイルに含まれています。

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
    ippRandUniform_Direct_32f(src,N,0.0,1.0,&seed);
    t1=ippGetCpuClocks();
    for(i=0; i<loops; i++)
        ippSqrt_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++)
        ippSqrt_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
```

エマージド・ライブラリー経由でスタティック・リンクを実行する場合、考慮すべき項目がいくつかあります。表 5-5 は、この種のスタティック・リンクの長所と短所を要約したものです。

表 5-5 スタティック・リンクの機能の要約 (ディスパッチあり)

長所	短所
<ul style="list-style-type: none"> ランタイム中にプロセッサ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 	<ul style="list-style-type: none"> 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 ippInitCpu() 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、libippsemmerged.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。

最新のサンプルについては、<http://www.intel.com/software/products/ipp/samples.htm> (英語) 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いて、対応プロセッサが 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です。1 種類のプロセッサ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 5-6 は、このリンク方法の長所と短所の要約です。

表 5-6 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
<ul style="list-style-type: none"> • 対応プロセッサが 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、対応プロセッサが 1 種類の場合に最適 • 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 	<ul style="list-style-type: none"> • 実行ファイルは 1 種類のプロセッサ向けにのみ最適化 • プロセッサ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おり、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 5-7](#) は、カスタム SO の長所と短所の要約です。

表 5-7 カスタム SO の機能

長所	短所
<ul style="list-style-type: none"> プロセッサ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 	<ul style="list-style-type: none"> 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/ 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。最新のサンプルについては、<http://www.intel.com/software/products/ipp/samples.htm> (英語) 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 5-8 は、インテル® IPP のリンク方法の比較です。

表 5-8 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
プロセッサの更新	自動	再コンパイルして再配布	新しいプロセッサ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	すべてのプロセッサ	1つのプロセッサ	すべてのプロセッサ
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

アプリケーションに必要なインテル® IPP ライブラリーの選択

表 5-9 に、関数ドメインと、各リンク方法で使用するヘッダーファイルおよびライブラリーを示します。

表 5-9 各リンク方法で使用するライブラリー

ドメインの説明	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	libippacmerged.a	libippacmerged.a libippacmerged_t.a
カラー変換	ippcc.h	libippcc.so	libippccmerged.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	libippcpmerged.a	libippcpmerged.a libippcpmerged_t.a
コンピューター・ビジョン	ippcv.h	libippcv.so	libippcvmerged.a	libippcvmerged.a libippcvmerged_t.a

表 5-9 各リンク方法で使用するライブラリー (続き)

ドメインの説明	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)
データ圧縮	ippdc.h	libippdc.so	libippdcmerged.a	libippdcmerged.a libippdcmerged_t.a
データ完全性	ippdi.h	libippdi.so	libippdipmerged.a	libippdimerged.a libippdimerged_t.a
生成関数	ipps.h	libippgen.so	libippgenmerged.a	libippgenmerged.a libippgenmerged_t.a
イメージ処理	ippi.h	libippi.so	libippimerged.a	libippimerged.a libippimerged_t.a
イメージ圧縮	ippj.h	libippj.so	libippjmerged.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	libipprmerged.a	libipprmerged.a libipprmerged_t.a
小行列演算	ippm.h	libippm.so	libippmmerged.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	libippscmerged.a	libippscmerged.a libippscmerged_t.a
音声認識	ippsr.h	libippsr.so	libippsrmerged.a	libippsrmerged.a libippsrmerged_t.a
ビデオ・コーディング	ippvc.h	libippvc.so	libippvcmerged.a	libippvcmerged.a libippvcmerged_t.a
ベクトル演算	ippvm.h	libippvm.so	libippvmmerged.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、適切な関数ドメインを示します。アプリケーションで使用しているドメイン・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ドメインに属しています。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
libiomp.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、libsmmerged.a および libcore.a をリンクする必要があります。* は適切な関数ドメインを示します。

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、libsmmerged_t.a、libcore_t.a および libiomp5.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、ドメイン固有の関数を含む lib ディレクトリーにあります。アプリケーションで使用するすべてのドメインと信号処理ドメインのマージド・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ドメインに属しています。ライブラリーをリンクする順序は、ドメイン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ります。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。

```
libcvmerged.a および libcvmerged_t.a
libmmerged.a および libmmerged_t.a
libiemerged.a および libiemerged_t.a
libsemerged.a および libsmmerged_t.a
libcore_t.a
libiomp5.a
```

ドメイン別のライブラリー依存関係 (スタティック・リンクのみ)

表 5-10 は、ドメイン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、データ圧縮ドメイン) 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。

表 5-10 ドメイン別のライブラリー依存関係

ドメイン	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, ippcore
カラー変換	ippcc	ippi, ippcore
暗号化	ippcp	ippcore
コンピューター・ビジョン	ippcv	ippi, ippcore

表 5-10 ドメイン別のライブラリー依存関係

ドメイン	ライブラリー	依存ライブラリー
データ圧縮	ippdc	ipps, ippcore
データ完全性	ippdi	ippcore
生成関数	ippgen	ipps, ippcore
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

関数が属しているドメインは、『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> (英語) 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、`ippSetNumThreads(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

信号処理ドメインの一部の関数は、インテル® Core™2 プロセッサ・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、パラメーター 1 で `ippSetNumThreads` 関数を呼び出すか、IPP の非スレッド・スタティック・ライブラリーをリンクしてアプリケーションをビルドします。

パフォーマンスとメモリの管理

7

この章は、メモリのアライメント、デノーマルデータへのしきい値の設定、バッファの再利用、高速フーリエ変換 (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 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 )
```

異なるデータ型の信号要素に、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 )
```

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。

[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容量は、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};
```

例 7-1 `ippiMalloc` 関数の呼び出し (続き)

```

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データが発生した場合、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 プロセッサ以降のインテル® プロセッサでは、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、計算がインテル® ストリーミング SIMD 拡張命令 (インテル® SSE) およびインテル® ストリーミング SIMD 拡張命令 2 (インテル® 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
要素ごとのプロセッサ・サイクル	46	11467	49

バッファの再利用

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が必要です。同時に、関数の内部メモリの割り当ては、キャッシュミスのように状況によっては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。この処理の一般的な例として、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 );
```

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

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、空間ドメインの2つの信号のフィルタリングを周波数ドメインの各点乗算として計算できるとしています。周波数ドメイン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の累乗に増やした後、順方向のFFT関数を入力信号とFIRフィルタ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、結果は簡単に空間ドメイン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。

適用するフィルタが複数の反復処理で同じ場合、フィルタ係数のFFT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®IPPパフォーマンス・テスト・ツールは、インテル®Pentium®プロセッサおよびインテル®Itanium®プロセッサ・ベースのWindows*オペレーティング・システムで利用できます。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・オプションを使用して、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ドメインとプロセッサ・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 関数ドメイン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> [option_1] [option_2] ... [option_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。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、オプション名の間に少なくとも 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 (`-r`、`-R`、`-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 の「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・テスト・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 -r
```

このコマンドは、すべてのインテル® IPP スtring 関数を、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 `ps_ippch.csv` ファイルに生成されます (`-r` オプション)。

例 2 - 選択した関数のテスト

```
ps_ipps -fFIRLMS_32f -r firlms.csv
```

このコマンドは、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 (`-r` オプション)。

例 3 - 関数リストの取得

```
ps_ippvc -e -o vc_list.txt
```

このコマンドは、`vc_list.txt` ファイル (`-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 -r H264.csv -f H264
```

このコマンドは、`H264` を含む (`-f` オプション) 名前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 (`-r` 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
```

このコマンドは、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_ippi -i -r
```

このコマンドは、`ps_ipps.ini` ファイルで指定されたタイミングと関数パラメーター値を使用してすべての関数をテストする (`-i` オプション) 2 回目の実行を行い、出力ファ

イル `ps_ipps.csv` (-r 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 の「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

この章は、Windows® OS 開発環境の異なるプログラミング言語でインテル® IPP を使用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> (英語) 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、 GNU C/C++ コンパイラ	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する方法を示します。

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

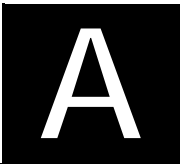


表 A-1 は、パフォーマンス・テスト・ツール (PTT) の利用可能なコマンドライン・オプションの概要です。

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。
	-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]。
2. 出力の管理	-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。

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
4. .ini ファイルを使用した操作	-i[<file-name>]	.ini ファイルから PTT パラメーターを読み取ります。
	-l[<file-name>]	.ini ファイルに PTT パラメーターを書き込みます。
	-p	.ini ファイルからテスト関数名を読み取ります。
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>	ini ファイルと入力テスト・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>	PTT 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)。
8. 環境の設定	-N<num-threads>	ippSetNumThreads(<num-treads>) を呼び出します。
9. ヘルプの表示	-h	簡易ヘルプを表示します。
	-hh	拡張ヘルプを表示します。
	-h<option>	指定されたオプションの拡張ヘルプを表示します。

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示します。[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示します。
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示します。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示す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示す目的で提供されています。

ソースコードのサンプル

[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、多少修正することで 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> ObjectViewer アプリケーション: ObjectViewerDoc.cpp ObjectViewerDoc.h ObjectViewerView.cpp ObjectViewerView.h <ul style="list-style-type: none"> 頂点と法線の変換: <code>CTestView::OnMutateModel</code> オブジェクトの平面への投影: <code>CTestView::OnProjectPlane</code> カーソル下の三角形の描画: <code>CTestView::Draw</code> 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、バッファありとバッファなし: perform2.cpp

表 B-2 インテル® IPP サンプルコードのソースファイル (続き)

カテゴリー	サマリー	説明とリンク
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	<ul style="list-style-type: none"> 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> (英語) からインテル® IPP サンプルをダウンロードしてください。

これらのアプリケーションレベルのサンプルは、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利用可能な場合、インテル® IPP サンプルをアップグレードすることを強く推奨します。

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 6.1 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 11.1、11.0 または 10.1 Linux* OS 版、GNU C/C++ コンパイラー 3.2 以上
- Qt* ライブラリー・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。例えば、`/opt/intel/Compiler/11.1/xxx/ia32/` と設定します。

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 ソフトウェアの特定のバージョンを使用する場合、以下の表に従って、スクリプトのオプションを設定してください。

表 B-3 スクリプトのオプション

コンパイラー	オプション
インテル® C++ コンパイラー 11.1 Linux 版	icc111
インテル® C++ コンパイラー 11.0 Linux 版	icc110
インテル® C++ コンパイラー 10.1 Linux 版	icc101
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、icc111|icc110|icc101|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ています。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, 7-4

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, 2-7

い

インストールの確認, 2-7

インテル (R) IPP, 1-1

インテル (R) IPP と Java の使用, 8-1

インテル (R) IPP の使用

Visual C++, 4-2

Visual C++.NET, 4-2

プログラミング言語, 8-1

か

環境の設定, 4-2

環境変数の設定, 2-7

関数の呼び出し, 2-8

け

言語サポート, 8-1

こ

構成

高レベル・ディレクトリー, 3-1

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

さ

サンプル, B-1

タイプ, B-1

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

サンプルのビルド, B-3

し

しきい値データ, 7-3

使用

DLL, 3-2

スタティック・ライブラリー, 3-3

す

スレッディング, 6-1

スレッド数の制御, 6-1

せ

選択

ライブラリー, 5-8

そ

ソースコードのサンプル, B-1

て

提供ライブラリー, 3-2

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

構成, 1-3

対象者, 1-2

場所, 3-1

ファイル名, 3-4

目的, 1-2
ドメイン別のライブラリーの依存関係, 5-10

は

バージョン情報, 2-7
バッファの再利用, 7-4
パフォーマンスの管理, 7-1
 FFT の使用, 7-4
 しきい値データ, 7-3
 バッファの再利用, 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, 7-5
 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

ひ

表記規則, 1-4

ふ

プロセッサ固有のコード, 5-1
プロセッサのタイプの検出, 5-2

へ

ヘッダーファイル, 2-8

ま

マルチスレッディングの無効化, 6-2

め

メモリー・アライメント, 7-1

り

リンク
 カスタム・ダイナミック, 5-7
 スタティック、ディスパッチあり, 5-4
 スタティック、ディスパッチなし, 5-6
 ダイナミック, 5-4
リンクモデル, 5-3
リンクモデルの選択, 5-3
リンクモデルの比較, 5-8
リンク例, 5-11