

# 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 Windows\* 版 (IA-32 アーキテクチャー用)

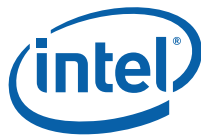
ユーザーガイド

---

2008年9月

資料番号 : 318254-004JP

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® インテグレートッド・パフォーマンス・プリミティブ (インテル® IPP) Windows* 版 (IA-32 アーキテクチャー用) ユーザーガイドのオリジナルバージョン。	2007 年 7 月
-002	インテル® IPP 5.3 リリースのドキュメント。	2007 年 11 月
-003	インテル® IPP 6.0 ベータ版リリースのドキュメント。	2008 年 2 月
-004	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。

© 2007 - 2008 Intel Corporation. 無断での引用、転載を禁じます。

# 目次

---

<b>第 1 章</b>	<b>概要</b>	
	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
<b>第 2 章</b>	<b>インテル® IPP 入門</b>	
	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.....	2-2
	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.....	2-3
<b>第 3 章</b>	<b>インテル® IPP の構造</b>	
	ディレクトリー構造.....	3-1
	ライブラリー .....	3-2
	インテル® IPP ダイナミック・リンク・ライブラリー (DLL) 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
<b>第 4 章</b>	<b>開発環境の設定</b>	
	インテル® IPP を Microsoft* Visual C++* .NET 2003 または Microsoft Visual C++* 2005 でリンクするための設定 .....	4-1
	インテル® IPP サンプル用の Visual C++ 2005 プロジェクト・ファイルの作成.....	4-1
	UMC サンプルコード用の Microsoft Visual C++ .NET ソリューションの構築.....	4-2
	インテル® C++ コンパイラーでのインテル® IPP の使用 .....	4-2
	Borland C++ Builder* 統合開発環境でのインテル® IPP の使用.....	4-3
<b>第 5 章</b>	<b>インテル® IPP とアプリケーションのリンク</b>	
	ディスパッチ .....	5-1

	プロセッサのタイプの検出 .....	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 DLL のビルド .....	5-6
	インテル® IPP のリンク方法の比較.....	5-7
	アプリケーションに必要なインテル® IPP ライブラリーの選択.....	5-7
	ダイナミック・リンク .....	5-9
	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) .....	5-10
	リンク例.....	5-10
<b>第 6 章</b>	<b>マルチスレッド・アプリケーションのサポート</b>	
	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
<b>第 7 章</b>	<b>パフォーマンスとメモリーの管理</b>	
	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
<b>第 8 章</b>	<b>各種プログラミング言語でのインテル® IPP の使用</b>	
	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
<b>付録 A</b>	<b>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</b>	
<b>付録 B</b>	<b>インテル® IPP のサンプル</b>	
	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-2
	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

インテル® インテグレートド・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 Windows\* 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 Windows\* 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 Windows\* 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 Windows\* 版 (IA-64 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 Windows\* 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 Windows 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 Windows 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 | 「 <a href="#">概要</a> 」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 | 「 <a href="#">インテル® IPP 入門</a> 」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 | 「 <a href="#">インテル® IPP の構造</a> 」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 | 「 <a href="#">開発環境の設定</a> 」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 | 「 <a href="#">インテル® IPP とアプリケーションのリンク</a> 」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 | 「 <a href="#">マルチスレッド・アプリケーションのサポート</a> 」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 | 「 <a href="#">パフォーマンスとメモリーの管理</a> 」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 | 「 <a href="#">各種プログラミング言語でのインテル® IPP の使用</a> 」では、異なるプログラミング言語と Windows 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  | 「 <a href="#">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</a> 」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  | 「 <a href="#">インテル® IPP のサンプル</a> 」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。 <code>\tools\env\ippenv.bat</code>
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。 <code>ippsFFTGetBufSize_C_32fc( ctxN2, &amp;sz );</code>
等幅の大文字	システム変数を示します。例: <code>PATH</code>
等幅の斜体	関数パラメーター ( <code>lda</code> など) や <code>makefile</code> パラメーター ( <code>functions_list</code> 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。 <code>&lt;ipp directory&gt;</code>
[ 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 `ippenv.bat` 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 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. <Ipp directory>\ia32\demo で `ippiDemo.exe` (または `ippsDemo.exe`) を実行します。

エラーメッセージ "This application has failed to start because ippcore.dll was not found" または "No DLL were found in the waterfall procedure" が表示された場合、オペレーティング・システムでインテル® IPP ダイナミック・リンク・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 ダイナミック・リンク・ライブラリーを使用する前に、「[環境変数の設定](#)」で説明されているように、<IPP directory>\ia32 を PATH 環境変数に追加します。
- \ia32\bin の内容を \system32 ディレクトリーにコピーします。
- \ia32\bin の内容をアプリケーション・ディレクトリーにコピーします。

インテル® IPP の以前のリリースでインストールしたすべての DLL を C:\winnt\system および C:\winnt\system32 ディレクトリーから削除する必要があることに注意してください。古いバージョンのライブラリーのパスが PATH 環境変数に含まれていないことを確認します。

## 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 にあるバッチファイル `ippenv.bat` を実行すると、インテル® IPP のコマンド・プロンプト・セッション用の LIB、INCLUDE および PATH 環境変数が設定されます。

これらの環境変数を、コマンド・プロンプト・セッションではなく、システムの環境変数として設定するには、以下のステップを完了してください（Windows\* XP の場合を説明します）。

1. デスクトップまたは Windows エクスプローラから [マイ コンピュータ] アイコンを右クリックして、[プロパティ] を選択します（または、[コントロール パネル] を開いて [システム] を選択します）。
2. [詳細設定] タブを選択します。
3. [環境変数] ボタンをクリックします。
4. 現在のユーザー（ダイアログボックスの上部）またはシステムのすべてのユーザー（ダイアログボックスの下部）の環境変数を設定するインターフェイスを使用します。
5. 編集する変数を選択して、[編集] ボタンをクリックします。
6. インテル® IPP ファイルに関連するパスを既存のリストに追加します。次に例を示します。  
LIB を選択して、インテル® IPP スタブ・ライブラリーのディレクトリーを入力します（デフォルトは `<IPP directory>\ia32\stublib`）。  
INCLUDE を選択して、インテル® IPP ヘッダーファイルのディレクトリーを入力します（デフォルトは `<IPP directory>\ia32\include`）。  
PATH を選択して、インテル® IPP バイナリーのディレクトリーを入力します（デフォルトは `<IPP directory>\ia32\bin`）。
7. [ユーザー変数の編集] ダイアログボックスで [OK] をクリックします（ユーザー環境変数を編集した場合）。
8. [環境変数] ダイアログボックスで [OK] をクリックします。
9. [システムのプロパティ] ダイアログボックスで [OK] をクリックします。

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

DLL 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	<p>用途に応じたインテル® IPP 関数定義域を識別します。</p> <p>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。</p> <p><a href="http://www.intel.com/software/products/ipp/samples.htm">http://www.intel.com/software/products/ipp/samples.htm</a> のサンプルも参照してください。</p> <p>関数定義域と必要なライブラリーについては、<a href="#">表 5-8</a> 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、<a href="#">表 5-9</a> 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、「<a href="#">インテル® IPP とアプリケーションのリンク</a>」を参照してください。</p>
スレッディング・モデル	<p>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。</p> <ul style="list-style-type: none"> <li>• アプリケーションはすでにスレッド化されている。</li> <li>• 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 (<code>libiomp</code>)、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。</li> <li>• アプリケーションをスレッド化しない。</li> </ul> <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、「<a href="#">マルチスレッド・アプリケーションのサポート</a>」を参照してください。</p>



# インテル® IPP の構造

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.rtf	インテル® IPP のエンド・ユーザー・ライセンス契約書
<ipp directory>\bin	インテル® IPP ダイナミック・リンク・ライブラリー (DLL)
<ipp directory>\demo	各種イメージ処理および信号処理機能を示すサンプルプログラム
<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>\stublib	インテル® IPP インポート・ライブラリー。DLL のリンクで使用。
<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	ダイナミック・リンク・ライブラリー (DLL) 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\bin	ipp6-6.0.dll, ippst7-6.0.dll
スタティック (インポート)	"スタブ"スタティック・ライブラリー・ファイル。必要な DLL をロードして正しいエントリーポイントにリンクします。	\ia32\stublib	ipp6.lib
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	\ia32\lib	ippmerged.lib
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	ippmerged_t.lib
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	\ia32\lib	ippsemerged.lib

## インテル® IPP ダイナミック・リンク・ライブラリー (DLL) の使用

\ia32\bin ディレクトリーには、インテル® IPP のダイナミック・リンク・ライブラリー (DLL) がインストールされます。

インテル® IPP DLL をロードして正しいエントリーポイントにリンクするには、\ia32\stublib ディレクトリーにインストールされる"スタブ"ライブラリー・ファイルを使用します (表 3-1 を参照)。

DLL を使用するには、ipp\*.lib ファイルにリンクします。ippenv.bat ファイルを使用して 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。適切なプロセッサ向けの DLL を動的にリンクするには、これらのライブラリーをインクルードするだけです。

DLL ipp\*-6.0.dll (\* 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を検出して、適切なプロセッサ固有の DLL 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固有ライブラリーには、ipp\*px-6.0.dll, ipp\*w7-6.0.dll, ipp\*t7-6.0.dll, ipp\*v8-6.0.dll, および ipp\*p8-6.0.dll 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\bin ディレクトリーの ippiv8-6.0.dll 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。

-6.0-6.0dll-6.0dll-6.0dll-6.0dll-6.0 いったん、"スタブ"スタティック・ライブラリーがリンクされると、インテル® IPP DLL を使用するために必要なことは、プログラムを実行する際にディスパッチャー DLL とプロセッサ固有の DLL がパスに存在するかどうかを確認することだけです。

「[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 libiomp5md.dll も 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、bin ディレクトリーをインクルードしてください。

以下に IPP 関数呼び出しの最も簡単なコード例 (t1.cpp ファイル) を示します。

**例 3-1      コード呼び出し例**

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

int main() {
    const IppLibraryVersion* libver = ippGetLibVersion();
    printf("%s %s\n", libver->Name, libver->Version);
    return 0;
}

t1.cpp

cmdlinetest>t1.exe
ippcore-6.0.dll 6.0 build 81
```

## 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・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 ipp\*merged.lib ファイルにリンクします。ippenv.bat ファイルを使用して 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。	

# 3 インテル® IPP 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
ReleaseNotes.htm	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	
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_win_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)		
ippsmn.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・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

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。

## インテル® IPP を Microsoft\* Visual C++\* .NET 2003 または Microsoft Visual C++\* 2005 でリンクするための設定

インテル® IPP を Microsoft Visual C++ .NET 2003 または Microsoft Visual C++ 2005 環境でリンクするには、以下の操作を行います。

1. [表示]>[ソリューション エクスプローラ]を選択します (このウィンドウがアクティブになっていることを確認します)。
2. [ツール]>[オプション]>[プロジェクトおよびソリューション]>[VC++ ディレクトリ]を選択します。
3. [ディレクトリを表示するプロジェクト]ドロップダウン・メニューで[インクルード ファイル]を選択し、インテル® IPP インクルード・ファイルのディレクトリを入力します (例えば、デフォルトの場合、<IPP directory>\ia32\include)。
4. [ディレクトリを表示するプロジェクト]ドロップダウン・メニューで[ライブラリ ファイル]を選択し、インテル® IPP ライブラリー・ファイルのディレクトリを入力します (例えば、デフォルトの場合、<IPP directory>\ia32\stplib または <IPP directory>\ia32\lib)。
5. [ディレクトリを表示するプロジェクト]ドロップダウン・メニューで[実行可能ファイル]を選択し、インテル® IPP バイナリーのディレクトリを入力します (例えば、デフォルトの場合、<IPP directory>\ia32\bin)。
6. [プロジェクト]>[プロパティ]>[構成プロパティ]>[リンカ]>[入力]を選択し、[追加の依存ファイル]にリンクするライブラリーを追加します (例えば、ipps.lib または ippsmerged.lib)。インテル® IPP アプリケーションの最適なリンク方法の選択に関する詳細は、「[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。

## インテル® IPP サンプル用の Visual C++ 2005 プロジェクト・ファイルの作成

インテル® IPP サンプル用の Microsoft Visual C++ 2005 プロジェクト・ファイルを作成するには、以下の jpegview サンプルコードを使用する例の手順に従ってください。

1. メディア・コーデック・サンプル jpegview を <http://www.intel.com/software/products/ipp/samples.htm> からダウンロードします。
2. [ファイル]>[新規作成]>[既存のコードからプロジェクトを作成]を選択します。

3. ポップアップ・ウィンドウのドロップダウン・メニューから、作成するプロジェクトの種類として **Visual C++** を選択し、プロジェクト・ファイルの場所と名前を設定します。[次のフォルダからプロジェクトにファイルを追加します] チェックボックスをオンにします。[追加] をクリックし、jpegview フォルダを選択して、[OK] をクリックします。
4. [表示]>[ソリューション エクスプローラ] を選択します (このウィンドウがアクティブになっていることを確認します)。
5. [ツール]>[オプション]>[プロジェクトおよびソリューション]>[VC++ ディレクトリ] を選択します。
6. [ディレクトリを表示するプロジェクト] ドロップダウン・メニューで [インクルード ファイル] を選択し、インテル® IPP インクルード・ファイルのディレクトリを入力します (例えば、デフォルトの場合、<IPP directory>\ia32\include)。
7. [ディレクトリを表示するプロジェクト] ドロップダウン・メニューで [ライブラリ ファイル] を選択し、インテル® IPP ライブラリー・ファイルのディレクトリを入力します (例えば、デフォルトの場合、<IPP directory>\ia32\stublib または <IPP directory>\ia32\lib)。
8. [ディレクトリを表示するプロジェクト] ドロップダウン・メニューで [実行可能ファイル] を選択し、インテル® IPP バイナリーのディレクトリを入力します (例えば、デフォルトの場合、<IPP directory>\ia32\bin)。

## UMC サンプルコード用の Microsoft Visual C++ .NET ソリューションの構築

インテル® IPP UMC サンプルコード用の Microsoft Visual C++ .NET プロジェクトとソリューション・ファイルを作成するには、gen\_vsproj.pl を使用します。このスクリプトファイルは、<http://www.intel.com/support/performance/tools/libraries/ipp/win/ia/sb/cs-022835.htm> からダウンロードできます。このスクリプトは、ActivePerl スクリプトがインストールされている Microsoft Windows OS で動作し、インテル® C++ コンパイラ 10.0 および Microsoft Visual C++ 2005 用のソリューション・ファイルをビルドできます。このスクリプトを使用する場合は、以下の手順に従ってください。

1. スクリプトファイルを、インテル® IPP UMC サンプル・コード・フォルダ `\ipp-samples\audio-video-codecs` に保存します。
2. コマンドラインで、gen\_vsproj.pl コマンドを使用してソリューション・ファイルを作成します。「gen\_vsproj.pl」と入力すると、すべてのコマンドメッセージが表示されます。次に、IA-32 アーキテクチャー用アプリケーションの Microsoft Visual C++ .NET 2003 ソリューションを生成する例を示します。

```
>gen_vsproj.pl -vs2003 -noicl -con application\* -gui
application\umc_reverb_demo -dll plug-in\audio_codecs -lib
codec\* core\* io\* pipeline\* plug-in\object_factory -
I"javascript:void(null);" -win32 -L "javascript:void(null);"
"javascript:void(null);" -l "javascript:void(null);" ddraw.lib
dsound.lib
```

ソリューション・ファイルは `ipp-samples\audio-video-codecs\application\xxxx` ディレクトリにあります。

インテル® IPP サンプルコードの実行に関する詳細は、「[付録 B - インテル® IPP のサンプル](#)」を参照してください。

## インテル® C++ コンパイラでのインテル® IPP の使用

インテル® C++ コンパイラでのインテル® IPP の使用は、Microsoft C++ コンパイラでのインテル® IPP の使用と似ています。

Microsoft Visual C++ .NET 環境で、[ ツール ] > [ オプション ] > [ プロジェクトおよびソリューション ] > [ VC++ ディレクトリ ] の設定ではなく、[ ツール ] > [ オプション ] > [ インテル (R) C++ ] > [ コンパイラー ] の設定を使用します。

1. [ 表示 ] > [ ソリューション エクスプローラ ] を選択します (このウィンドウがアクティブになっていることを確認します)。
2. [ ツール ] > [ オプション ] > [ インテル (R) C++ ] > [ コンパイラー ] を選択します。
3. [ インクルード ] に、インテル® IPP インクルード・ファイルのディレクトリーを入力します (例えば、デフォルトの場合、`<IPP directory>\ia32\include`)。
4. [ ライブラリー ] に、インテル® IPP ライブラリー・ファイルのディレクトリーを入力します (例えば、デフォルトの場合、`<IPP directory>\ia32\stublic` または `<IPP directory>\ia32\lib`)。
5. [ 実行可能ファイル ] に、インテル® IPP バイナリーのディレクトリーを入力します (例えば、デフォルトの場合、`<IPP directory>\ia32\bin`)。
6. [ プロジェクト ] > [ プロパティ ] > [ 構成プロパティ ] > [ リンカ ] > [ 入力 ] を選択し、[ 追加の依存ファイル ] にリンクするライブラリーを追加します (例えば、`ipps.lib` または `ippsmerged.lib`)。

## Borland C++ Builder\* 統合開発環境でのインテル® IPP の使用

Borland C++ Builder でインテル® IPP スタティック・ライブラリーを直接リンクすることはできません。代わりに、以下の操作を行います。

インテル® IPP マージド・スタティック・ライブラリーを使用して、カスタム DLL を作成します (アプリケーションで呼び出す関数をエクスポートします)。カスタム DLL およびマージド・スタティック・ライブラリー・ツールは統合サンプルパッケージの一部として利用可能です。インテル® IPP スタティック・ライブラリーは、`ipp\ia32\lib` ディレクトリーにあります。

(DLL に) `IMPLIB` を使用して、Borland 環境でインポート・ライブラリーを作成します。



# 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\bin` ディレクトリーの `ippiv8-6.0.dll` 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ー向けのインテル® 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 (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 (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 (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 (SSE4.1) 対応プロセッサ向けに最適化

## プロセッサのタイプの検出

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

表 5-2 プロセッサのタイプの検出、戻り値と意味

戻り値	プロセッサのタイプ
ippCpuPP	インテル® Pentium® プロセッサ
ippCpuPMX	MMX® テクノロジー インテル® Pentium® プロセッサ
ippCpuPPR	インテル® Pentium® Pro プロセッサ
ippCpuPII	インテル® Pentium® II プロセッサ
ippCpuPIII	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
ippCpuP4	インテル® Pentium® 4 プロセッサおよびインテル® Xeon® プロセッサ
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
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
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
ippCpuBonnell	インテル® Atom™ 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
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
ippCpuX8664	64 ビット拡張命令対応プロセッサ
ippCpuUnknown	未知のプロセッサ

## 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・ダイナミック・リンク・ライブラリー (DLL) 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 DLL 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、ダイナミック・リンク・ライブラリー (DLL) 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
<ul style="list-style-type: none"> <li>• 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</li> <li>• 再コンパイル/再リンクしないで新しいプロセッサの最適化を更新可能</li> <li>• 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</li> <li>• 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</li> </ul>	<ul style="list-style-type: none"> <li>• アプリケーションを実行するときにインテル® IPP ランタイムダイナミック・リンク・ライブラリー (DLL) にアクセスする必要がある</li> <li>• 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</li> <li>• 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</li> <li>• インテル® IPP DLL 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</li> </ul>

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. 対応する定義域インポート・ライブラリーをリンクします。例えば、`ippsCopy_8u` 関数を使用する場合、`ipps.lib` をリンクします。
4. ランタイム・ライブラリー (例えば、`ipps.dll`) が実行時の検索パスに存在することを確認します。`\tools\env` ディレクトリーで `ippenv.bat` を実行し、インテル® IPP ダイナミック・リンク・ライブラリーを使用して構築されたアプリケーションが適切なプロセッサ固有のライブラリーをロードすることを保証します。

## 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、`ippsemmerged.lib`) は、非修飾の (標準名の) IPP 関数のエントリーポイントと、各プロセッサ固有の実装へのジャンプテーブルを提供します。ア

アプリケーションをリンクするとき、関数は、ippcore1.lib 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、ippsmerged.libÅj 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
<ul style="list-style-type: none"> <li>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</li> <li>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</li> <li>フルセットのインテル® IPP よりも小さなフットプリントを生成</li> </ul>	<ul style="list-style-type: none"> <li>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</li> <li>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</li> </ul>



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、`ippcore1.lib` を（順に）リンクします。例えば、`ippsCopy_8u()` 関数を使用する場合、`ippsemerged.lib`、`ippsmerged.lib`、および `ippcore1.lib` 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
<ul style="list-style-type: none"> <li>• 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</li> <li>• カーネルモード / デバイスドライバー / ring-0 コードに最適*)</li> <li>• 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</li> <li>• 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 DLL が不要</li> <li>• アプリケーション・パッケージで最小のフットプリント</li> <li>• 最小のインストール・パッケージ</li> </ul>	<ul style="list-style-type: none"> <li>• 実行ファイルは1種類のプロセッサ向けのみ最適化</li> <li>• 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</li> </ul>

\*) スレッド化されていないライブラリーのみ。

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 DLL 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ダイナミック・リンク・ライブラリー (DLL) 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 DLL の長所と短所の要約です。

表 5-6 カスタム DLL の機能

長所	短所
<ul style="list-style-type: none"><li>プロセッサ固有の最適化のランタイム・ディスパッチ</li><li>フルセットのインテル® IPP DLL よりもハードドライブのフットプリントが小さい</li><li>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</li></ul>	<ul style="list-style-type: none"><li>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</li><li>カスタム DLL の作成および保守に開発者のリソースが必要</li><li>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 DLL のリビルドが必要</li><li>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</li></ul>

カスタム DLL を作成するには、DLL 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 dll` 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\customdll` 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 DLL 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 DLL 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

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	ippac.lib	ippacmerged.lib	ippacmerged.lib ippacmerged_t.lib
カラー変換	ippcc.h	ippcc.lib	ippccmerged.lib	ippccmerged.lib ippccmerged_t.lib
ストリング処理	ippch.h	ippch.lib	ippchemerged.lib	ippchmerged.lib ippchmerged_t.lib
暗号化	ippcp.h	ippcp.lib	ippcpmerged.lib	ippcpmerged.lib ippcpmerged_t.lib

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)
コンピューター・ ビジョン	ippcv.h	ippcv.lib	ippcvmerged.lib	ippcvmerged.lib ippcvmerged_t.lib
データ圧縮	ippdc.h	ippdc.lib	ippdcmerged.lib	ippdcmerged.lib ippdcmerged_t.lib
データ完全性	ippdi.h	ippdi.lib	ippdimerged.lib	ippdimerged.lib ippdimerged_t.lib
生成関数	ipps.h	ippgen.lib	ippgenmerged.lib	ippgenmerged.lib ippgenmerged_t.lib
イメージ処理	ippi.h	ippi.lib	ippimerged.lib	ippimerged.lib ippimerged_t.lib
イメージ圧縮	ippj.h	ippj.lib	ippjmerged.lib	ippjmerged.lib ippjmerged_t.lib
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	ippr.lib	ippremerged.lib	ipprmerged.lib ipprmerged_t.lib
小行列演算	ippm.h	ippm.lib	ippmmerged.lib	ippmmerged.lib ippmmerged_t.lib
信号処理	ipps.h	ipps.lib	ippsemerged.lib	ippsmerged.lib ippsmerged_t.lib
音声 コーディング	ippsc.h	ippsc.lib	ippscmerged.lib	ippscmerged.lib ippscmerged_t.lib
音声認識	ippsr.h	ippsr.lib	ippsremerged.lib	ippsrmerged.lib ippsrmerged_t.lib
ビデオ・ コーディング	ippvc.h	ippvc.lib	ippvcmerged.lib	ippvcmerged.lib ippvcmerged_t.lib
ベクトル演算	ippvm.h	ippvm.lib	ippvmmerged.lib	ippvmmerged.lib ippvmmerged_t.lib

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	動的リンク	スタティック・リンク (ディスパッチあり) & カスタム・動的リンク	スタティック・リンク (ディスパッチなし)
コア関数	ippcore.h	ippcore.lib	ippcore1.lib	ippcore1.lib ippcore_t.lib

## 動的リンク

動的リンク・ライブラリーを使用するには、\stublic ディレクトリーの `ipp*.lib` ファイルへの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、`ipps.lib`、`ippcore.lib` および `libiomp.lib` 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。

```
ippi.lib
ippcv.lib
ippm.lib
ippcore.lib
libiomp5.lib
```

## 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、`ipp*emerged.lib`、`ipp*merged.lib`、`ippsemerged.lib`、`ippsmerged.lib` および `ippcore1.lib` 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、`ipp*emerged.lib`、`ipp*merged_t.lib`、`ippsemerged.lib`、`ippsmerged_t.lib`、`ippcore_t.lib` および `libiomp.lib` 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。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。

```
ippcvmerged.lib および ippcvmerged_t.lib
ippmerged.lib および ippmerged_t.lib
ippiemerged.lib および ippiemerged_t.lib
ippsemerged.lib および ippsmerged_t.lib
```

## 5 インテル® IPP ユーザーガイド

ippcore\_t.lib

libiomp5.lib

### 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。

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, ipp, ippcore
カラー変換	ippcc	ippi, ipp, ippcore
暗号化	ippcp	ippcore
コンピューター・ビジョン	ippcv	ippi, ipp, ippcore
データ圧縮	ippdc	ipp, ippcore
データ完全性	ippdi	ippcore
生成関数	ippgen	ipp, ippcore
イメージ処理	ippi	ipp, ippcore
イメージ圧縮	ippj	ippi, ipp, ippcore
小行列演算	ippm	ippi, ipp, 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, ipp, ippcore
信号処理	ipp	ippcore
音声コーディング	ippsc	ipp, ippcore
音声認識	ippsr	ipp, ippcore
ストリング処理	ippch	ipp, ippcore
ビデオ・コーディング	ippvc	ippi, ipp, 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・ライブラリおよびスタティック・ライブラリ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

信号処理定義域の一部の関数は、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、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 DLL 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 )
```

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 "stdafx.h"
#include "ipp.h"
#include "tools.h"

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

    int stride;
    Ipp8u* pSrc = ippiMalloc_8u_C3(size.width, size.height, &stride);
    ippiImageJaehne_8u_C3R(pSrc, stride, size);
    ipView_8u_C3R(pSrc, stride, size, "Source image", 0);

    int dstStride;
    Ipp8u* pDst = ippiMalloc_8u_C3(size.width, size.height, &dstStride);
    ippiCopy_8u_C3R(pSrc, stride, pDst, dstStride, size);
    ipView_8u_C3R(pDst, dstStride, size, "Destination image 1", 0);

    IppiSize ROISize = { size.width/2, size.height/2 };
    ippiCopy_8u_C3R(pSrc, stride, pDst, dstStride, ROISize);
    ipView_8u_C3R(pDst, dstStride, ROISize, "Destination image, small", 0);

    IppiPoint srcOffset = { size.width/4, size.height/4 };
    ippiCopy_8u_C3R(pSrc + srcOffset.x*3 + srcOffset.y*stride, stride,
        pDst, dstStride, ROISize);
    ipView_8u_C3R(pDst, dstStride, ROISize, "Destination image, small &
shifted", 1);

    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, xright, &fftlen2, 2, &phase );

ippsFFTFwd_CToC_32fc( xleft, Xleft, ctxN2, buffer );
ippsFFTFwd_CToC_32fc( xright, Xright, 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。

```

ippFFTInitAlloc_R_32f( &pFFTSpec, fftord, IPP_FFT_DIV_INV_BY_N,
ippAlgHintNone );

// perform forward FFT to put source data xx to frequency domain
ippFFTFwd_RToPack_32f( xx, XX, pFFTSpec, 0 );

// perform forward FFT to put filter coefficients hh to frequency domain
ippFFTFwd_RToPack_32f( hh, HH, pFFTSpec, 0 );

// point-wise multiplication in frequency domain is convolution
ippMulPack_32f_I( HH, XX, fftlen );

// perform inverse FFT to get result yy in time domain
ippFFTInv_PackToR_32f( XX, yy, pFFTSpec, 0 );

// free FFT tables
ippFFTFree_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 of 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・ベースの Windows\* 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 .exe 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.exe 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>.exe [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.exe -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.exe -B -v
```

インテル® IPP スtring 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.exe -fFIRLMS_32f -v firllms.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 firllms.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.exe -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.exe -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.exe -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.exe -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® 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。
	Microsoft* eMbedded Visual C++* 4.0 以上 WinCE 5.0 SDK for x86	x86 プラットフォーム上の Windows CE 環境向けアプリケーションの開発でインテル® IPP ライブラリーを使用します。
Fortran	Makefile	N/A
C#	Microsoft .NET* C#	C# ラッパークラスでインテル® IPP 関数を使用します。
Visual Basic*	Microsoft .NET Visual Basic	Visual Basic ラッパークラスからインテル® IPP 関数を呼び出します。
Object Pascal	Borland Delphi*	Borland Delphi でインテル® IPP イメージ処理プリミティブを使用します。
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

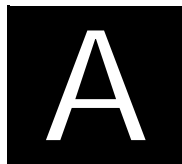


表 A-1 は、利用可能なパフォーマンス・テスト・ツールのコマンドライン・オプションの概要です。

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。
3. テスト用関数の選択	-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。
	-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。
4. .ini ファイルを使用した操作	-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。
	-i[<file-name>]	ini ファイルから PS パラメーターを読み取ります。
	-l[<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せん。

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)。
8. 環境の設定	-T<cpu-name>	ippInitStaticCpu(ippCpu<cpu-name>) を呼び出します。
	-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 サンプル用の Microsoft\* Visual C++\* プロジェクト・ファイルの設定方法については、「[インテル® IPP サンプル用の Visual C++ 2005 プロジェクト・ファイルの作成](#)」を参照してください。

インテル® IPP UMC サンプルコード用の Microsoft Visual C++ .NET プロジェクトとソリューション・ファイルの生成については、「[UMC サンプルコード用の Microsoft Visual C++ .NET ソリューションの構築](#)」を参照してください。

## 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	<ul style="list-style-type: none"> <li>パフォーマンス測定: <a href="#">GetClocks.cpp</a></li> <li>データのコピー: <a href="#">Copy.cpp</a></li> <li>テーブルベース関数の最適化: <a href="#">LUT.cpp</a></li> </ul>
デジタル・フィルタリング	信号処理の基本	<ul style="list-style-type: none"> <li>DFT の実行: <a href="#">DFT.cpp</a></li> <li>FFT を使用したフィルタリング: <a href="#">FFTFilter.cpp</a></li> <li>時間定義域フィルタリング: <a href="#">FIR.cpp</a></li> </ul>
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"> <li>DTMF トーンの生成: <a href="#">DTMF.cpp</a></li> <li>IIR を使用したエコーの作成: <a href="#">IIR.cpp</a></li> <li>FIRMR を使用した信号の再サンプリング: <a href="#">Resample.cpp</a></li> </ul>
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"> <li>イメージの割り当て、初期化およびコピー: <a href="#">Copy.cpp</a></li> <li>矩形処理サンプルラッパー: <a href="#">ROI.h</a> <a href="#">ROI.cpp</a> <a href="#">ROITest.cpp</a></li> <li>マスク・イメージ・サンプル・ラッパー: <a href="#">Mask.h</a> <a href="#">Mask.cpp</a> <a href="#">MaskTest.cpp</a></li> </ul>
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"> <li>イメージリサイズ用ラッパー: <a href="#">Resize.h</a> <a href="#">Resize.cpp</a> <a href="#">ResizeTest.cpp</a></li> <li>イメージ回転用ラッパー: <a href="#">Rotate.h</a> <a href="#">Rotate.cpp</a> <a href="#">RotateTest.cpp</a></li> <li>イメージのアフィン変換実行用ラッパー: <a href="#">Affine.h</a> <a href="#">Affine.cpp</a> <a href="#">AffineTest.cpp</a></li> </ul>
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"> <li>ObjectViewer アプリケーション: <a href="#">ObjectViewerDoc.cpp</a> <a href="#">ObjectViewerDoc.h</a> <a href="#">ObjectViewerView.cpp</a> <a href="#">ObjectViewerView.h</a></li> <li>頂点と法線の変換: <a href="#">CTestView::OnMutateModel</a></li> <li>オブジェクトの平面への投影: <a href="#">CTestView::OnProjectPlane</a></li> <li>カーソル下の三角形の描画: <a href="#">CTestView::Draw</a></li> <li>パフォーマンスの比較、ベクトルとスカラー: <a href="#">perform.cpp</a></li> <li>パフォーマンスの比較、バッファありとバッファなし: <a href="#">perform2.cpp</a></li> </ul>
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"> <li>RSA 鍵の生成および暗号化: <a href="#">rsa.cpp</a> <a href="#">rsa.h</a> <a href="#">rsatest.cpp</a> <a href="#">bignum.h</a> <a href="#">bignum.cpp</a></li> <li>Canny エッジ検出クラス: <a href="#">canny.cpp</a> <a href="#">canny.h</a> <a href="#">cannytest.cpp</a> <a href="#">filter.h</a> <a href="#">filter.cpp</a></li> <li>ガウシアン角錐クラス: <a href="#">pyramid.cpp</a> <a href="#">pyramid.h</a> <a href="#">pyramidtest.cpp</a></li> </ul>

## 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 Windows\* 版
- Microsoft Windows Vista\*、Microsoft Windows XP、Microsoft Windows Server\* 2008、または Microsoft Windows Server 2003 オペレーティング・システム
- Microsoft DirectX API: 9.0 SDK Update (February 2005) または SDK (December 2005)
- インテル® C++ コンパイラー 10.1、10.0 または 9.1 Windows\* 版 Microsoft Visual C++ 2008 または Microsoft Visual C++ 2005 開発環境
- インテル® 64 アーキテクチャー対応プロセッサー用にビルドする場合は、Microsoft EM64T Platform SDK が必要です。
- インテル® Itanium® プロセッサー用にビルドする場合は、Platform SDK for Microsoft Windows Server 2003 SP1 が必要です。
- Microsoft eMbedded Visual C++ 4.0 tool with the Service Pack 4 (SP4)  
Windows CE OS 5.0 for x86 用にビルドする場合は、Standard Software Development Kit (SDK) for Windows CE 5.0 が必要です。

### 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、バッチファイル *build32.bat* [option] を実行します。

デフォルトでは、バッチファイル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 Microsoft C/C++ 2005 コンパイラーを使用する場合、以下の表に従って、バッチファイルの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 Windows 版	icl1101
インテル® C++ コンパイラー 10.0 Windows 版	icl110
インテル® C++ コンパイラー 9.1 Windows 版	icl91
Microsoft Visual C++ 2008	cl9
Microsoft Visual C++ 2005	cl8
Microsoft Visual C++ .NET 2003	cl7



**注:** Windows CE 5.0 を使用している場合、wceplatform.bat で PLATFORM を設定する必要があります。該当プラットフォーム用の SDK を使用していることを確認してください。通常は、次のように、SDKROOT 環境変数を手動で設定する必要があります。

```
set SDKROOT=C:\Program Files\Windows CE Tools
```

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\win32\_<compiler> に生成されます。  
compiler は、cl7|cl8|cl9|icl91|icl110|icl1101 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 ダイナミック・リンク・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」に記載されている制限があります。





# 索引

---

## 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

## U

UMC サンプルコード, 4-2

## あ

アプリケーションのビルド, 2-2

## 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++, 4-1

Visual C++.NET, 4-1

コンパイラー, 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-2

## 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, 3-4

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, 5-10

## は

バージョン情報, 2-1

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

## ふ

プロセッサのタイプの検出, 5-1

プロセッサ固有のコード, 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

環境の設定, 4-1

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