

# インテル® マス・カーネル・ライブラリー Linux\* OS 版

ユーザーズガイド

---

2009 年 3 月

資料番号 : 314774-009JA

Web サイト : <http://www.intel.com/software/products/> ( 英語 )

<http://www.intel.co.jp/jp/software/products/> ( 日本語 )



バージョン	バージョン情報	日付
-001	初版。インテル® マス・カーネル・ライブラリー (インテル® MKL) 9.0 gold リリースについて説明。	2006 年 9 月
-002	インテル® MKL 9.1 beta リリースについて説明。「はじめに」、「LINPACK ベンチマークと MP LINPACK ベンチマーク」の章と「サードパーティー・インターフェイスのサポート」の付録を追加。既存の章を拡張。ドキュメントを再構成。例のリストを追加。	2007 年 1 月
-003	インテル® MKL 9.1 gold リリースについて説明。既存の章を拡張。ドキュメントを再構成。ILP64 追加。「インテル® MKL を Eclipse* IDE CDT でリンクする場合の構成」セクションを第 3 章に追加。クラスターに関する内容を 1 つの独立した第 9 章「インテル® マス・カーネル・ライブラリー・クラスター・ソフトウェアの使用」に移動して再構成し、適切なリンクを追加。	2007 年 6 月
-004	インテル® MKL 10.0 Beta リリースについて説明。レイヤーモデルについての記述を第 3 章に追加し、本書の内容をモデルに合わせて調整。スタートアップ時の環境変数の自動設定についての記述を第 4 章に追加。新しいインテル® MKL スレッド化コントロールについての記述を第 6 章に追加。インテル® MKL のユーザーズガイドとインテル® MKL クラスター・エディションのユーザーズガイドを併合し、それぞれの製品の統合を反映。	2007 年 9 月
-005	インテル® MKL 10.0 Gold リリースについて説明。インテル® MKL を Eclipse CDT 4.0 でリンクする場合の説明を第 3 章に追加。インテル® 互換 OpenMP* ランタイム・コンパイラー・ライブラリー (libiomp) について記述。	2007 年 10 月
-006	インテル® MKL 10.1 beta リリースについて説明。表「高レベル・ディレクトリー構造」のダミー・ライブラリーの情報をより詳細に変更。インテル® MKL 構成ファイルの情報を削除。「Man ページのアクセス」セクションを第 3 章に追加。「Boost uBLAS 行列・行列乗算のサポート」セクションを第 7 章に追加。「Eclipse* IDE でのプログラミング」の章を追加。	2008 年 5 月
-007	インテル® MKL 10.1 gold リリースについて説明。IA-32 アーキテクチャーのリンク例と「計算ライブラリーのリンク」セクションを第 5 章に追加。DSS/PARDISO のレイヤー構造への統合について説明。2 つの Fortran コード例を追加。	2008 年 8 月
-008	インテル® MKL 10.2 beta リリースについて説明。BLAS/LAPACK 用の事前構築 Fortran 95 インターフェイス・ライブラリーとモジュールについて説明。インテル® Advanced Vector Extensions (インテル® AVX) のサポートについて説明。ダミー・ライブラリーとレガシー・リンク・モデルのサポート終了について説明。第 5 章の構成を変更。	2009 年 1 月
-009	インテル® MKL 10.2 gold リリースについて説明。本書の構成を大幅に変更。第 2 章の内容を拡張。SP2DP インターフェイスの説明を第 3 章に追加。Web ベースのリンク・アドバイザーの説明を第 2 章および第 5 章に追加。	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。

インテルによる書面での同意がない限り、インテル製品は、インテル製品の停止を起因とする人身傷害または死亡を想定して設計されていません。

インテル製品は、予告なく仕様や説明が変更される場合があります。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。この情報は予告なく変更されることがあります。この情報だけに基づいて設計を最終的なものとししないでください。

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

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

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

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

本資料には、開発の設計段階にある製品についての情報が含まれています。

Intel、インテル、Intel ロゴ、Intel Core、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。

© 2006 - 2009 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
	本書について .....	1-1
	本書の構成 .....	1-2
	用語と表記規則 .....	1-3
<b>第 2 章</b>	<b>はじめに</b>	
	開始する前に .....	2-1
	サポートするコンパイラ .....	2-2
	開始前の準備 .....	2-3
	インストールの確認 .....	2-3
	環境変数の設定 .....	2-3
	Web ベースのリンク・アドバイザーの使用 .....	2-3
	インテル® MKL コードサンプルの使用 .....	2-4
<b>第 3 章</b>	<b>インテル® マス・カーネル・ライブラリーの構造</b>	
	上位ディレクトリー構造 .....	3-1
	レイヤーモデルの概念 .....	3-2
	ライブラリーの逐次モード .....	3-4
	ILP64 プログラミングのサポート .....	3-4
	サポートしているアーキテクチャー .....	3-6
	詳細なディレクトリー構造 .....	3-6
	インテル® MKL ドキュメントへのアクセス .....	3-15
	ドキュメント・ディレクトリーの内容 .....	3-15
	Man ページへのアクセス .....	3-15
<b>第 4 章</b>	<b>開発環境の構成</b>	
	環境変数の自動設定 .....	4-1
	インテル® MKL を Eclipse* IDE CDT でリンクする場合の構成 .....	4-2
	Eclipse IDE CDT 4.0 でリンクする場合の構成 .....	4-2
	Eclipse IDE CDT 3.x の構成 .....	4-2
	Out-of-Core (OOC) DSS/PARDISO* ソルバーの構成 .....	4-3
<b>第 5 章</b>	<b>アプリケーションとインテル® マス・カーネル・ライブラリーのリンク</b>	
	リンク行のライブラリーのリスト .....	5-1
	リンクするライブラリーの選択 .....	5-2

	Fortran 95 インターフェイス・ライブラリーのリンク .....	5-2
	スレッド化ライブラリーのリンク .....	5-3
	計算ライブラリーのリンク .....	5-5
	コンパイラ・サポート RTL のリンク .....	5-7
	システム・ライブラリーのリンク .....	5-7
	リンクの例 .....	5-7
	カスタム共有オブジェクトの構築 .....	5-10
	インテル® MKL カスタム共有オブジェクト・ビルダー .....	5-10
	メイクファイル・パラメーターの指定 .....	5-10
	関数のリストの指定 .....	5-11
<b>第 6 章</b>	<b>パフォーマンスとメモリーの管理</b>	
	インテル® MKL 並列処理の使用 .....	6-1
	スレッド数を設定する手法 .....	6-3
	実行環境における競合の回避 .....	6-3
	OpenMP 環境変数を使用したスレッド数の設定 .....	6-4
	ランタイムのスレッド数の変更 .....	6-4
	新しいスレッド化コントロールの使用 .....	6-7
	インテル® Advanced Vector Extensions ( インテル® AVX) のディスパッチ ....	6-10
	パフォーマンスを向上するためのヒントと手法 .....	6-10
	コーディング手法 .....	6-11
	ハードウェア構成のヒント .....	6-11
	マルチコア・パフォーマンスの管理 .....	6-12
	非正規化数の演算 .....	6-13
	FFT 最適化基数 .....	6-13
	インテル® MKL メモリー管理の使用 .....	6-13
	メモリー関数の再定義 .....	6-14
<b>第 7 章</b>	<b>言語固有の使用法オプション</b>	
	言語固有インターフェイスとインテル® MKL の使用 .....	7-1
	混在言語プログラミングとインテル® MKL .....	7-4
	LAPACK、BLAS、および CBLAS ルーチンの C 言語環境からの呼び出し .....	7-4
	C/C++ での複素数型の使用 .....	7-6
	C/C++ コードで複素数を返す BLAS 関数の呼び出し .....	7-6
	Boost uBLAS 行列 - 行列乗算のサポート .....	7-8
	インテル® MKL 関数の Java アプリケーションからの呼び出し .....	7-10
<b>第 8 章</b>	<b>コーディングのヒント</b>	
	数値計算安定性のためのデータの整列 .....	8-1
<b>第 9 章</b>	<b>インテル® マス・カーネル・ライブラリー・クラスター・ソフトウェアの使用</b>	
	ScaLAPACK およびクラスター FFT とのリンク .....	9-1
	スレッド数の設定 .....	9-2
	共有ライブラリーの使用 .....	9-2

	ScaLAPACK テスト .....	9-3
	ScaLAPACK およびクラスター FFT とのリンク例 .....	9-3
	C アプリケーションのリンク例 .....	9-3
	Fortran アプリケーションのリンク例 .....	9-3
<b>第 10 章</b>	<b>Eclipse* IDE でのプログラミング支援機能</b>	
	Eclipse IDE 内でインテル® MKL リファレンス・マニュアルを表示 .....	10-1
	Eclipse IDE からインテルの Web サイトを検索 .....	10-2
	Eclipse IDE CDT での状況依存ヘルプの使用 .....	10-3
<b>第 11 章</b>	<b>LINPACK ベンチマークと MP LINPACK ベンチマーク</b>	
	Intel® Optimized LINPACK Benchmark for Linux OS .....	11-1
	内容 .....	11-1
	ソフトウェアの実行 .....	11-2
	既知の制限事項 .....	11-3
	Intel® Optimized MP LINPACK Benchmark for Clusters .....	11-3
	内容 .....	11-4
	MP LINPACK の構築 .....	11-7
	新機能 .....	11-7
	クラスターのベンチマーク .....	11-8
<b>付録 A</b>	<b>インテル® マス・カーネル・ライブラリー言語インターフェイスのサポート</b>	
<b>付録 B</b>	<b>サードパーティー・インターフェイスのサポート</b>	
	GMP* 関数 .....	B-1
	FFTW インターフェイスのサポート .....	B-1
<b>索引</b>		
<b>表の目次</b>		
	表 1-1 表記規則 .....	1-3
	表 2-1 開始前に知っておくべき項目 .....	2-1
	表 3-1 上位ディレクトリー構造 .....	3-1
	表 3-2 ILP64 および LP64 インターフェイス用のコンパイル .....	3-5
	表 3-3 整数型 .....	3-5
	表 3-4 アーキテクチャー固有の実装 .....	3-6
	表 3-5 IA-32 アーキテクチャー固有のディレクトリー lib/32 の構造の詳細 .....	3-7
	表 3-6 インテル® 64 アーキテクチャー固有のディレクトリー lib/em64t の構造の詳細 .....	3-10
	表 3-7 IA-64 アーキテクチャー固有のディレクトリー lib/64 の構造の詳細 .....	3-12
	表 3-8 doc ディレクトリーの内容 .....	3-15
	表 5-1 一般的なリンク・ライブラリー .....	5-1
	表 5-2 スレッド化ライブラリーの選択 .....	5-4

表 5-3 リンクする計算ライブラリー、関数ドメイン別.....	5-5
表 6-1 スレッド化モデル別の実行環境における競合の回避方法.....	6-3
表 6-2 スレッド化コントロール用の環境変数.....	6-7
表 6-3 MKL_DOMAIN_NUM_THREADS の値の解釈.....	6-9
表 7-1 インターフェイス・ライブラリーとモジュール.....	7-1
表 11-1 LINPACK Benchmark の内容.....	11-1
表 11-2 MP LINPACK Benchmark の内容.....	11-4

## 例の目次

例 6-1 スレッド数の変更.....	6-4
例 6-2 スレッド数を 1 に設定.....	6-8
例 6-3 インテル® コンパイラーを使用してオペレーティング・システムで アフィニティー・マスクを設定.....	6-12
例 6-4 メモリー関数の再定義.....	6-15
例 7-1 複素レベル 1 BLAS 関数の C からの呼び出し.....	7-7
例 7-2 複素レベル 1 BLAS 関数の C++ からの呼び出し.....	7-7
例 7-3 BLAS を C から直接呼び出す代わりに CBLAS インターフェイスを 使用.....	7-8
例 8-1 16 バイト境界でアドレスをアライメント.....	8-2

## 図の目次

図 7-1 列優先と行優先.....	7-5
図 10-1 Eclipse IDE のインテル® MKL ヘルプ.....	10-2
図 10-2 Eclipse IDE Help の検索でインテルの Web サイトに見つけた数.....	10-3
図 10-3 Infopop ウィンドウに表示されたインテル® MKL 関数の説明.....	10-4
図 10-4 Eclipse IDE の F1 ヘルプ.....	10-5
図 10-5 Eclipse IDE CDT での F1 ヘルプの検索.....	10-5

# 概要

# 1

インテル® マス・カーネル・ライブラリー (インテル® MKL) は、最大限のパフォーマンスが求められる、科学、工学、金融アプリケーション向けに高度に最適化された、スレッドセーフな数値演算ルーチンを提供します。

## 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。詳細は、インテル® MKL サポート Web サイト (<http://www.intel.com/software/products/support/>) を参照してください。

## 本書について

『インテル® MKL ユーザーズガイド』では、ライブラリーの使用法を説明します。この使用法には、インテル® MKL の構成、パフォーマンスおよび精度、混在言語プログラミングにおけるルーチン呼び出し、リンク、その他の情報が含まれています。

本書の内容と前提条件は以下のとおりです。

- Linux® OS 上で実行しているユーザーのアプリケーションからインテル® MKL ルーチン呼び出すために必要な使用法を説明します。
- インテル® MKL の OS 固有の使用法と OS に依存しない機能について説明します。
- [表 A-1](#) (付録 A) でインテル® MKL 関数ドメインをリストしています。
- インテル® MKL のインストールが完了した後にお読みください。インストールが完了していない場合、『インテル® マス・カーネル・ライブラリー インストール・ガイド』(Install.txt) を参照してインストールを完了してください。

アプリケーションでライブラリーを使用する方法については、本書のほか、次のドキュメントも併せて参照してください。

- 『インテル® MKL リファレンス・マニュアル』は、ルーチンの機能、パラメーターの説明、インターフェイス、呼び出し構文と戻り値についてのリファレンス情報を提供します。
- インテル® マス・カーネル・ライブラリー Linux® 版リリースノート

このユーザーズガイドは、Linux 上でインテル® MKL の使用法を習得することを目的として記述されています。特に、以下の項目について説明しています。

- ライブラリーを開始するために必要なインストール後の手順
- ライブラリーと開発環境を構成する方法
- ライブラリーの構造
- アプリケーションとライブラリーをリンクする方法と、簡単な使用法の例
- インテル® MKL Linux 版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



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

## 本書の構成

本書は、以下の章と付録から構成されています。

- |        |  |
|--------|--|
| 第 1 章  | 「 <a href="#">概要</a> 」。インテル® MKL の使用法を紹介します。また、本書の表記規則について説明します。   |
| 第 2 章  | 「 <a href="#">はじめに</a> 」。インストール後の手順と、インテル® MKL を使用するために必要な情報を説明します。  |
| 第 3 章  | 「 <a href="#">インテル® マス・カーネル・ライブラリーの構造</a> 」。インストール後のインテル® MKL ディレクトリーの構造について説明します。   |
| 第 4 章  | 「 <a href="#">開発環境の構成</a> 」。インテル® MKL と開発環境を構成する方法を説明します。  |
| 第 5 章  | 「 <a href="#">アプリケーションとインテル® マス・カーネル・ライブラリーのリンク</a> 」。特定のプラットフォーム用にアプリケーションとリンクするライブラリーを説明します。また、カスタム・ダイナミック・ライブラリーのビルド方法を説明します。   |
| 第 6 章  | 「 <a href="#">パフォーマンスとメモリーの管理</a> 」。インテル® MKL のスレッド化について説明し、ライブラリーのパフォーマンスを向上するためのコーディング・テクニックとハードウェア構成を示します。また、インテル® MKL のメモリー管理機能の特徴について説明し、デフォルトで使用するライブラリーのメモリー関数を独自の関数と置換する方法を示します。 |
| 第 7 章  | 「 <a href="#">言語固有の使用法オプション</a> 」。混在言語プログラミングと言語固有のインターフェイスの使用について説明します。   |
| 第 8 章  | 「 <a href="#">コーディングのヒント</a> 」。特定の用途に役立つコーディングのヒントを示します。   |
| 第 9 章  | 「 <a href="#">インテル® マス・カーネル・ライブラリー・クラスター・ソフトウェアの使用</a> 」。ScaLAPACK とクラスター FFT の使用法について、C と Fortran 固有のリンク例を含む、関数ドメインを使用するアプリケーションのリンク方法を説明します。また、サポートしている MPI の情報も記載しています。               |
| 第 10 章 | 「 <a href="#">Eclipse* IDE でのプログラミング支援機能</a> 」。ソフトウェア・エンジニアが Eclipse IDE で作業するときに役立つインテル® MKL の機能を説明します。   |
| 第 11 章 | 「 <a href="#">LINPACK ベンチマークと MP LINPACK ベンチマーク</a> 」。Intel® Optimized LINPACK Benchmark for Linux および Intel® Optimized MP LINPACK Benchmark for Clusters について説明します。                     |
| 付録 A   | 「 <a href="#">インテル® マス・カーネル・ライブラリー言語 インターフェイスのサポート</a> 」。ヘッダーファイルを含む、インテル® MKL で各関数ドメイン用に用意されている言語インターフェイス情報の要約です。   |
| 付録 B   | 「 <a href="#">サードパーティー・インターフェイスのサポート</a> 」。インテル® MKL でサポートされているインターフェイスについて説明します。  |

本書の最後には、[索引](#) も含まれています。

## 用語と表記規則

本書では、オペレーティング・システムについて以下の用語を使用しています。

Linux OS サポートしているすべての Linux オペレーティング・システムで有効な情報を指します。

インテル® MKL ディレクトリーは、以下のように表記しています。

<mk1 ディレクトリー> インテル® MKL がインストールされているメイン・ディレクトリー。構成、リンク、およびビルドの際には、この <mk1 ディレクトリー> の代わりに実際のパス名を指定してください。詳細は、「[はじめに](#)」を参照してください。

< インテル® コンパイラー・プロフェッショナル・ディレクトリー> インテル® C++ コンパイラー・プロフェッショナル・エディションまたはインテル® Fortran コンパイラー・プロフェッショナル・エディションのインストール・ディレクトリー。詳細は、「[はじめに](#)」を参照してください。

[表 1-1](#) に、その他の表記規則を示します。

表 1-1 表記規則

斜体	強調および文書名を示します。 例： 『インテル® MKL リファレンス・マニュアル』を参照してください。
等幅小文字	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： libmkl_core.a , /opt/intel/mkl/10.2.0.004
等幅小文字 / 大文字	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： icc myprog.c -L\$MKLPATH -I\$MKLINCLUDE -lmkl -lguid -lpthread ; C/C++ コードの一部を示します。 例： a = new double [SIZE*SIZE];
等幅大文字	システム変数を示します。 例： \$MKLPATH
等幅斜体	説明しているパラメーターを示します。 例： lda (ルーチン・パラメーター)、functions_list (makefile パラメーター) など。 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。 例： <mk1 ディレクトリー> プレースホルダーの代わりに、これらの項目のいずれかをusingしてください。
[ 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 つだけ選択することを示します。 垂直バー ( ) は項目の区切りです。

# はじめに

# 2

本章は、Linux® OS でインテル® マス・カーネル・ライブラリー (インテル® MKL) 版の使用を開始するために必要な基本的な情報とインストール後の手順を説明します。

## 開始する前に

インテル® MKL の使用を開始する前に、[表 2-1](#) で説明されているいくつかの重要な基本概念に目を通すようにしてください。

**表 2-1 開始前に知っておくべき項目**

ターゲット・プラットフォーム	<p><b>処理:</b> ターゲットマシンのアーキテクチャーを特定します。</p> <ul style="list-style-type: none"><li>IA-32 または IA-32 互換</li><li>インテル® 64 またはインテル® 64 互換</li><li>IA-64 (Itanium® プロセッサ・ファミリー)</li></ul> <p><b>理由:</b> インテル® MKL ライブラリーは、使用するアーキテクチャー (<a href="#">「サポートしているアーキテクチャー」</a> を参照) に対応するディレクトリーにあるため、リンク行で適切なパスを指定する必要があります (<a href="#">「リンクの例」</a> を参照)。インテル® MKL を使用する開発環境を構成するには、使用するアーキテクチャーに対応するスクリプトを使用して環境変数を設定します (詳細は <a href="#">「環境変数の設定」</a> を参照)。</p>
算術問題	<p><b>処理:</b> 必要なインテル® MKL 関数ドメインをすべて特定します。</p> <ul style="list-style-type: none"><li>BLAS</li><li>スパース BLAS</li><li>LAPACK</li><li>PBLAS</li><li>ScaLAPACK</li><li>スパース・ソルバー・ルーチン</li><li>ベクトル・マス・ライブラリー関数</li><li>ベクトル・スタティスティカル・ライブラリー関数</li><li>フーリエ変換関数 (FFT)</li><li>クラスター FFT</li><li>三角変換ルーチン</li><li>ポアソン、ラプラス、およびヘルムホルツ・ソルバー・ルーチン</li><li>最適化 (Trust-Region) ソルバールーチン</li><li>GMP® 数学関数</li></ul> <p><b>理由:</b> 使用する関数ドメインを特定することで、『インテル® MKL リファレンス・マニュアル』でルーチンを検索する項目が少なくなります。さらに、インテル® MKL クラスターを使用している場合、リンク行は関数ドメイン固有になります (<a href="#">「インテル® マス・カーネル・ライブラリー・クラスター・ソフトウェアの使用」</a> を参照)。コーディングのヒントは、関数ドメインにより異なります (<a href="#">「パフォーマンスを向上するためのヒントと手法」</a> を参照)。</p>

表 2-1 開始前に知っておくべき項目 ( 続き )

プログラミング言語	<p><b>処理:</b> インテル® MKL は Fortran と C/C++ プログラミングの両方をサポートしますが、すべての関数ドメインが特定の言語環境 ( 例えば、C/C++ または Fortran 90/95 ) をサポートするとは限りません。使用する関数ドメインでサポートされる言語インターフェイスを特定してください (「<a href="#">インテル® マス・カーネル・ライブラリー言語 インターフェイスのサポート</a>」を参照)。</p> <p><b>理由:</b> 関数ドメインが必要な環境を直接サポートしていない場合、混在言語プログラミングを使用することができます (「<a href="#">混在言語プログラミングとインテル® MKL</a>」を参照)。</p> <p>言語固有のインターフェイス・ライブラリーとモジュールの一覧、および使用例は、「<a href="#">言語固有インターフェイスとインテル® MKL の使用</a>」を参照してください。</p>
整数データの範囲	<p><b>処理:</b> インテル® 64 または IA-64 アーキテクチャー・ベースのシステムの場合、アプリケーションで大規模なデータ配列 (<math>2^{31}-1</math> 以上の要素を含む配列) の計算を実行するかどうかを決定します。</p> <p><b>理由:</b> 大規模なデータ配列を処理するには、ILP64 インターフェイス ( 整数が 64 ビット ) を選択する必要があります。その他の場合は、デフォルトの LP64 インターフェイス ( 整数が 32 ビット ) を使用します (「<a href="#">ILP64 プログラミングのサポート</a>」を参照)。</p>
スレッド化モデル	<p><b>処理:</b> アプリケーションをスレッド化するかどうか、スレッド化する場合はその方法を決定します。</p> <ul style="list-style-type: none"> <li>• インテル® コンパイラーを使用してスレッド化する</li> <li>• サードパーティ製のコンパイラーを使用してスレッド化する</li> <li>• スレッド化しない</li> </ul> <p><b>理由:</b> アプリケーションのスレッド化に使用するコンパイラーにより、アプリケーションとリンクするスレッド化ライブラリーが決まります。サードパーティ製のコンパイラーを使用してアプリケーションをスレッド化する場合、インテル® MKL を逐次モードで使用する必要があります ( 詳細は、「<a href="#">ライブラリーの逐次モード</a>」および「<a href="#">スレッド化ライブラリーのリンク</a>」を参照)。</p> <p><b>処理:</b> インテル® MKL で使用するスレッド数を決定します。</p> <p><b>理由:</b> インテル® MKL は、OpenMP* スレッド化をベースにしています。OpenMP ソフトウェアは、インテル® MKL が使用するスレッドの数を自動的に設定します。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、「<a href="#">インテル® MKL 並列処理の使用</a>」を参照してください。</p>
リンクモデル	<p><b>処理:</b> アプリケーションとインテル® MKL ライブラリーをリンクする適切なリンクモデルを決定します。</p> <ul style="list-style-type: none"> <li>• スタティック</li> <li>• ダイナミック</li> </ul> <p><b>理由:</b> 各リンクモデルの利点、リンクコマンドの構文と例、リンク・ライブラリー、その他のリンクに関する情報 ( カスタム・ダイナミック・ライブラリーを作成してディスク容量を節約する方法など ) は、「<a href="#">アプリケーションとインテル® マス・カーネル・ライブラリーのリンク</a>」を参照してください。</p>
使用する MPI	<p><b>処理:</b> インテル® MKL クラスターで使用する MPI を決定します。インテル® MPI 3.x を使用することを強く推奨します。</p> <p><b>理由:</b> アプリケーションと ScaLAPACK やクラスター FFT をリンクする際に、使用する MPI に対応するライブラリーをリンク行で指定する必要があるためです (「<a href="#">インテル® マス・カーネル・ライブラリー・クラスター・ソフトウェアの使用</a>」を参照)。</p>

## サポートするコンパイラー

インテル® MKL がサポートしているコンパイラーは、『[リリースノート](#)』に記述されています。しかし、ライブラリーはほかのコンパイラーでも動作することが確認されています。

インテル® MKL には、インクルード・ファイルのセットが用意されています。関数の列挙値とプロトタイプを指定することで、プログラム開発が単純化されます ( インクルード・ファイルのリストは、[表 A-2](#) を参照 )。適切なインクルード・ファイルを使用せずにアプリケーションからインテル® MKL 関数を呼び出すと、関数が正しく動作しない場合があります。

## 開始前の準備

このセクションでは、Linux OS でインテル® MKL を使用するために必要な情報を説明します。

## インストールの確認

インテル® MKL をインストールした後、ライブラリーが正しくインストールされ、設定されていることを確認します。

1. まず、インストール用に選択したディレクトリーが作成されていることを確認します。インテル® MKL のデフォルトのインストール・ディレクトリーは以下のいずれかになります。

- /opt/intel/mkl/RR.r.y.xxx  
(RR.r はバージョン番号、y はリリース番号、xxx はパッケージ番号。  
例: /opt/intel/mkl/10.2.0.004)
- <インテル® コンパイラー・プロフェッショナル・ディレクトリー>/mkl  
(例: /opt/intel/Compiler/11.1/015/mkl)

2. システムにインテル® MKL の複数のバージョンをインストールしている場合、使用するバージョンを指定してビルドスクリプトを更新します。

3. 以下の 6 つのファイルが tools/environment ディレクトリーにインストールされます。

```
mklvars32.sh
mklvars32.csh
mklvarsem64t.sh
mklvarsem64t.csh
mklvars64.sh
mklvars64.csh
```

現在のユーザーシェルで、これらのファイルを使用して環境変数を設定します。

4. インテル® MKL インストール・ディレクトリーの詳細な構造は、第 3 章を参照してください。

## 環境変数の設定

インテル® MKL Linux® OS 版のインストールが完了すると、tools/environment ディレクトリーにある 3 つのスクリプト mklvars32、mklvarsem64t、および mklvars64 (.sh および .csh) を使用して、ユーザーシェルで環境変数 INCLUDE、MKLROOT、LD\_LIBRARY\_PATH、MANPATH、LIBRARY\_PATH、CPATH、FPATH、および NLSPATH を設定できます。ライブラリーの構成についての詳細は、第 4 章を参照してください。

## Web ベースのリンク・アドバイザーの使用

アプリケーションのリンク行に指定するライブラリーとオプションの選択を支援する、Web ベースのリンク・アドバイザーが利用できるようになりました。

<http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor> (英語) にアクセスしてください。

インテル® MKL の使用方法を選択すると、リンク行が画面に表示されます。選択できる項目は、オペレーティング・システム、アーキテクチャー、コンパイラー、リンクモデル (スタティックまたはダイナミック)、整数の長さ (32 ビットまたは 64 ビット)、インテル® MKL の動作モード (スレッド化または逐次)、その他です。

表示された出力をコピーして、リンク行に貼り付けてください。

インテル® MKL ライブラリーのリンクについては、第 5 章を参照してください。一般的なリンク・ライブラリーは、[表 5-1](#) にリストされています。

## インテル® MKL コードサンプルの使用

インテル® MKL パッケージにはコードサンプルが含まれています。コードサンプルは、インテル® MKL インストール・ディレクトリーの `examples` サブディレクトリーにあります。サンプルを使用して、以下のような項目を確認できます。

- インテル® MKL がシステムで動作しているか
- ライブラリーの呼び出し方法
- ライブラリーのリンク方法

サンプルは、主にインテル® MKL 関数ドメインとプログラミング言語別にサブディレクトリーでグループ化されています。例えば、`examples/spblas` サブディレクトリーにはスパース BLAS のサンプル、`examples/vmlc` サブディレクトリーには VML の C のサンプルが含まれています。サンプルのソースコードは、`sources` サブディレクトリーにあります。

サンプルをコンパイル、ビルド、実行するには、提供されているメイクファイルを使用します。メイクファイルの使用法についての情報は、メイクファイル・ヘッダーを参照してください。

また、以下の項目も参照してください。

[上位ディレクトリー構造](#)

# インテル® マス・カーネル・ライブラリーの構造

## 3

本章は、インテル® マス・カーネル・ライブラリー (インテル® MKL) のディレクトリー構造、アーキテクチャー固有の実装、サポートしているプログラミング・インターフェイス、その他を含む、インテル® MKL の構造について説明します。

インテル® MKL は、バージョン 10.0 からレイヤーモデルを採用しています。この変更により、ライブラリーの構造が合理化され、サイズが減少し、使用法に柔軟性が増しました。

「[レイヤーモデルの概念](#)」も参照してください。

## 上位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® MKL の上位ディレクトリー構造を示しています。

表 3-1 上位ディレクトリー構造

ディレクトリー	内容
<mk1 ディレクトリー>	インテル® MKL メイン・ディレクトリー
<mk1 ディレクトリー>/benchmarks/linpack	LINPACK ベンチマークの共有メモリー (SMP) バージョン
<mk1 ディレクトリー>/benchmarks/mp_linpack	LINPACK ベンチマークの MPI バージョン
<mk1 ディレクトリー>/doc	スタンドアロンのインテル® MKL のドキュメント
<mk1 ディレクトリー>/examples	サンプルのソースコード、データ、およびメイクファイル
<mk1 ディレクトリー>/include	ライブラリー・ルーチンとサンプルのテスト用のインクルード・ファイル
<mk1 ディレクトリー>/include/32	IA-32 アーキテクチャー、インテル® Fortran コンパイラー用 BLAS95 <sup>1</sup> および LAPACK95 <sup>2</sup> .mod ファイル
<mk1 ディレクトリー>/include/64/ilp64	IA-64 アーキテクチャー、インテル® Fortran コンパイラー、ILP64 インターフェイス用 BLAS95 および LAPACK95 .mod ファイル
<mk1 ディレクトリー>/include/64/lp64	IA-64 アーキテクチャー、インテル® Fortran コンパイラー、LP64 インターフェイス用 BLAS95 および LAPACK95 .mod ファイル
<mk1 ディレクトリー>/include/em64t/ilp64	インテル® 64 アーキテクチャー (旧称インテル® EM64T)、インテル® Fortran コンパイラー、ILP64 インターフェイス用 BLAS95 および LAPACK95 .mod ファイル
<mk1 ディレクトリー>/include/em64t/lp64	インテル® 64 アーキテクチャー、インテル® Fortran コンパイラー、LP64 インターフェイス用 BLAS95 および LAPACK95 .mod ファイル
<mk1 ディレクトリー>/interfaces/blas95	BLAS 用 Fortran 95 インターフェイスとライブラリー・ビルド用のメイクファイル

表 3-1 上位ディレクトリー構造 ( 続き )

ディレクトリー	内容
<mk1 ディレクトリー>/interfaces/fftw2x_cdft	インテル® MKL クラスター FFT 用 MPI FFTW 2.x インターフェイス
<mk1 ディレクトリー>/interfaces/fftw2xc	インテル® MKL FFT 用 FFTW 2.x インターフェイス (C インターフェイス)
<mk1 ディレクトリー>/interfaces/fftw2xf	インテル® MKL FFT 用 FFTW 2.x インターフェイス (Fortran インターフェイス)
<mk1 ディレクトリー>/interfaces/fftw3xc	インテル® MKL FFT 用 FFTW 3.x インターフェイス (C インターフェイス)
<mk1 ディレクトリー>/interfaces/fftw3xf	インテル® MKL FFT 用 FFTW 3.x インターフェイス (Fortran インターフェイス)
<mk1 ディレクトリー>/interfaces/lapack95	LAPACK 用 Fortran 95 インターフェイスとライブラリー・ビルド用のメイクファイル
<mk1 ディレクトリー>/lib/32	IA-32 アーキテクチャー用のスタティック・ライブラリーと共有オブジェクト
<mk1 ディレクトリー>/lib/64	IA-64 アーキテクチャー (インテル® Itanium® プロセッサー・ファミリー) 用のスタティック・ライブラリーと共有オブジェクト
<mk1 ディレクトリー>/lib/em64t	インテル® 64 アーキテクチャー用のスタティック・ライブラリーと共有オブジェクト
<mk1 ディレクトリー>/man/ja_JP/man3	インテル® MKL 関数の man ページ (スタンドアロンのインテル® MKL 用)
<mk1 ディレクトリー>/tests	テスト用のソースおよびデータファイル
<mk1 ディレクトリー>/tools/builder	カスタム・ダイナミック・リンク・ライブラリー作成用のツール
<mk1 ディレクトリー>/tools/environment	ユーザーシェルで環境変数を設定するシェルスクリプト
<mk1 ディレクトリー>/tools/plugins/com.intel.mk1.help	WebHelp 形式のインテル® MKL リファレンス・マニュアルと Eclipse* IDE プラグイン。mk1_documentation.htm のコメントを参照。
< インテル® コンパイラー・プロフェッショナル・ディレクトリー>/documentation/ja_JP/mk1	インテル® C++ コンパイラー・プロフェッショナル・エディションまたはインテル® Fortran コンパイラー・プロフェッショナル・エディションに含まれているインテル® MKL のドキュメント。
< インテル® コンパイラー・プロフェッショナル・ディレクトリー>/man/ja_JP/man3	インテル® C++ コンパイラー・プロフェッショナル・エディションまたはインテル® Fortran コンパイラー・プロフェッショナル・エディションに含まれているインテル® MKL の man ページ。

1. Fortran 95 インターフェイス、BLAS
2. Fortran 95 インターフェイス、LAPACK

## レイヤーモデルの概念

インテル® MKL は、バージョン 10.0 からレイヤーモデルを採用しています。

ライブラリーには 4 つのレイヤーがあります。

1. インターフェイス・レイヤー
2. スレッド化レイヤー
3. 計算レイヤー
4. コンパイラー・サポート・ランタイム・ライブラリー。

最初のレイヤーは、インテル® MKL のインターフェイス関連の問題に対応します。例えば、使用する整数型 (32 ビットまたは 64 ビット)、異なる関数値を返すコンパイラーへの対応が含まれます。2 つめのレイヤーは、ライブラリーを、異なるスレッド化コンパイラーで使用される OpenMP\* 実



装や非スレッドモードに対応させます。計算レイヤーはライブラリーの大部分を占め、スペースの節約のために個別の部分に分離されています。これらの部分で各ケースごとに1つのライブラリーを生成すると非常に大量のライブラリーを生成することになるため、インテル® MKL では、各部分をそれらのケースに応じて個別のライブラリーに分割しました。これらのライブラリーは必要に応じて組み合わせることができます。いったんインターフェイス・ライブラリーが選択されると、スレッド化ライブラリーはそのインターフェイスを取得し、計算ライブラリーは最初の2つのレイヤーで選択されたインターフェイスとスレッド化を使用します。

**インターフェイス・レイヤー:** このレイヤーは、コンパイルされたアプリケーションのコードと、ライブラリーのスレッド化および計算部分に対応させます。このレイヤーは以下のインターフェイスと手段を提供します。

- ILP64 および ILP64 インターフェイス  
(詳細は、[「ILP64 プログラミングのサポート」](#) を参照)
- 異なる関数値を返すコンパイラーとの互換性
- Cray\* 形式の名前を使用するアプリケーションの単精度名と倍精度名のマッピング (SP2DP インターフェイス)



**注:** SP2DP インターフェイスは、インテル® 64 または IA-64 アーキテクチャー用の ILP64 インターフェイスを使用するアプリケーションで Cray 形式の名前をサポートしています。SP2DP インターフェイスは、アプリケーションの単精度名 (実数型および複素数型) とインテル MKL BLAS/LAPACK の倍精度名のマッピングを提供します。例えば、BLAS 関数 \*GEMM の場合、関数名は以下のようにマッピングされます。

```
SGEMM -> DGEMM
DGEMM -> DGEMM
CGEMM -> ZGEMM
ZGEMM -> ZGEMM
```

倍精度名は変更されません。

**スレッド化レイヤー:** このレイヤーは以下のインターフェイスと手段を提供します。

- このレイヤーを使用して、スレッド化されたインテル® MKL と異なるスレッド化コンパイラーをリンクします。
- ユーザーは、ライブラリーのスレッド化モードまたは逐次モードでリンクできます。

このレイヤーは、異なる環境 (スレッド化または逐次) やコンパイラー (インテル® コンパイラー、GNU\*, その他) 向けにコンパイルされます。

**計算レイヤー:** このレイヤーはインテル® MKL の中心となる部分です。指定したプロセッサ・アーキテクチャー (IA-32、IA-64 またはインテル® 64) と OS で、インターフェイスやスレッド化レイヤーに関係なく、1つの計算ライブラリーが使用されます。計算レイヤーは、アーキテクチャーの機能を識別することで、実行時にさまざまなアーキテクチャー用に適切なバイナリーコードを選択します。

**コンパイラー・サポート・ランタイム・ライブラリー (RTL):** このレイヤーは RTL サポートを提供します。すべての RTL がインテル® MKL で提供されるとは限りません。インテル® MKL クラスタに関連するものを除いて、インテル® コンパイラーの RTL であるインテルの OpenMP 互換ランタイム・ライブラリー (libiomp) とインテルの OpenMP レガシー・ランタイム・ライブラリー (libguide) のみ提供されます。サードパーティーのスレッド化コンパイラーを使用してスレッド化を行うには、スレッド化レイヤーのライブラリーまたは適切な互換ライブラリーを使用します (詳細は、[「スレッド化ライブラリーのリンク」](#) を参照)。

## ライブラリーの逐次モード

インテル® MKL を逐次 (非スレッド) モードで使用できます。OpenMP 互換ランタイム・ライブラリーまたは OpenMP レガシー ランタイム・ライブラリーは必要なく、OMP\_NUM\_THREADS 環境変数やインテル® MKL の等価な変数には応答しません。このモードでは、インテル® MKL は非スレッドコードを実行しますが、このコードはスレッドセーフなので<sup>1</sup>、OpenMP コードの並列領域で使用できます。インテル® MKL のスレッド化を使用しない特別な理由がある場合のみ、逐次モードでライブラリーを使用してください。この逐次モードは、インテル® MKL を、インテル以外のコンパイラーでスレッド化されたプログラムに使用する場合や、さまざまな理由によりライブラリーの非スレッドバージョンが必要な場合に役立ちます (例えば、一部の MPI の場合)。インテル® MKL を逐次モードで実行するには、スレッド化レイヤーで、リンクするライブラリーとして `*sequential.*` ライブラリーを選択します。

`*sequential.*` ライブラリーは、インテル® MKL ソフトウェアをスレッドセーフにするためにリンク行で指定する、POSIX スレッド・ライブラリー (pthread) に依存することに注意してください。

以下の項目も参照してください。

[詳細なディレクトリー構造](#)

[インテル® MKL 並列処理の使用](#)

[実行環境における競合の回避](#)

[リンクの例](#)

## ILP64 プログラミングのサポート

インテル® MKL ILP64 ライブラリーは、64 ビット整数型 ( $2^{31}-1$  以上の要素を含む大規模な配列のインデックス処理に必要) を使用します。しかし、LP64 ライブラリーは 32 ビット整数型を使用して配列をインデックス処理します。

LP64 インターフェイスと ILP64 インターフェイスは、インターフェイス・レイヤーでサポートされています (詳細は、「[レイヤーモデルの概念](#)」および「[詳細なディレクトリー構造](#)」を参照)。

ILP64 インターフェイスは、以下の 2 つの理由により提供されています。

- 大規模なデータ配列 (要素数  $2^{31}-1$  以上) のサポート
- i8 コンパイラー・オプションを使用して Fortran コードをコンパイルする

用途に応じてインターフェイスを選択してください。“LP64” はインテル® MKL のバージョン 9.1 以前に提供されていたインターフェイスの新しい名前です。以前のインテル® MKL との互換性を保つ場合は、LP64 インターフェイスを選択してください。アプリケーションやライブラリーで大規模なデータ配列の計算にインテル® MKL を使用する場合は、ILP64 インターフェイスを選択してください。

ILP64 インターフェイスと LP64 インターフェイスでは同じインクルード・ディレクトリーが使用されます。

1. LAPACK の古いルーチン (7lacon、7lasq3、および 7lasq4) を除きます。

## LP64/ILP64 用のコンパイル

[表 3-2](#) は、ILP64 および LP64 インターフェイス用のコンパイル方法を示しています。

**表 3-2 ILP64 および LP64 インターフェイス用のコンパイル**

<b>Fortran</b>		
ILP64 用のコンパイル	<code>ifort -i8 -I&lt;mk1 ディレクトリー&gt;/include ...</code>	
LP64 用のコンパイル	<code>ifort -I&lt;mk1 ディレクトリー&gt;/include ...</code>	
<b>C/C++</b>		
ILP64 用のコンパイル	<code>icc -DMKL_ILP64 -I&lt;mk1 ディレクトリー&gt;/include ...</code>	
LP64 用のコンパイル	<code>icc -I&lt;mk1 ディレクトリー&gt;/include ...</code>	



**注:** `-i8` または `-DMKL_ILP64` オプションを使用してコンパイルしたアプリケーションと LP64 ライブラリーをリンクすると、予測できない結果や誤出力が発生する場合があります。

## ILP64 用のコーディング

ILP64 インターフェイスを使用していない場合、既存コードを変更する必要はありません。

ILP64 へ変更したり、ILP64 用に新しいコードを記述する場合は、インテル® MKL 関数とサブルーチンのパラメーターに適切な型を使用してください (「[表 3-3](#)」を参照)。

**表 3-3 整数型**

	<b>Fortran</b>	<b>C/C++</b>
32 ビット整数	<code>INTEGER*4</code> または <code>INTEGER (KIND=4)</code>	<code>int</code>
ILP64/LP64 のユニバーサル整数:	<code>INTEGER</code>	<code>MKL_INT</code>
▪ 64 ビット (ILP64 の場合)	<code>KIND</code> の指定なし	
▪ 32 ビット (その他の場合)		
ILP64/LP64 のユニバーサル整数:	<code>INTEGER*8</code> または <code>INTEGER (KIND=8)</code>	<code>MKL_INT64</code>
▪ 64 ビット整数	<code>INTEGER</code>	<code>MKL_LONG</code>
ILP64/LP64 の FFT インターフェイス整数:		
	<code>KIND</code> の指定なし	

## インテル® MKL インクルード・ファイルの参照

整数パラメーターを使用する関数について、『インテル® MKL リファレンス・マニュアル』では、ILP64 で 64 ビットになるパラメーターと 32 ビットのままのパラメーターは説明されていません。ILP64 インターフェイスの詳細は、インクルード・ファイル、サンプル、およびテストを参照してください。ファイルの場所は、[表 3-1](#) を参照してください。

最初に、[表 A-2](#) にリストされているインクルード・ファイルを参照してください。

Fortran インターフェイスのみをサポートする一部の関数ドメイン ([表 A-1](#) を参照) でも、インクルード・ディレクトリーに C/C++ 用のヘッダーファイルが提供されます。これらの `*.h` ファイルを使用すると、C/C++ コードから Fortran バイナリー・インターフェイスが利用可能になります。これらのファイルは、ILP64 の使用法を理解するために使用することもできます。

## 制限

すべてのインテル® MKL 関数ドメインは、以下の例外を除いて ILP64 プログラミングをサポートしています。

- インテル® MKL 用 FFTW インターフェイス:
  - FFTW 2.x ラッパーは ILP64 をサポートしていません。
  - FFTW 3.2 ラッパーは専用の関数セット `plan_guru64` によって ILP64 をサポートしています。
- GMP\* 数学関数は ILP64 をサポートしていません。

## サポートしているアーキテクチャー

インテル® MKL Linux\* OS 版では、3 つのアーキテクチャー固有の実装を提供しています。[表 3-4](#) に、サポートしているアーキテクチャーと各アーキテクチャー固有の実装が含まれているディレクトリーを示します。

**表 3-4 アーキテクチャー固有の実装**

アーキテクチャー	場所
IA-32 または IA-32 互換	<code>&lt;mk1 ディレクトリー&gt;/lib/32</code>
インテル® 64 またはインテル® 64 互換	<code>&lt;mk1 ディレクトリー&gt;/lib/em64t</code>
IA-64	<code>&lt;mk1 ディレクトリー&gt;/lib/64</code>

これらのディレクトリー構造の詳細は、[表 3-5](#)、[表 3-6](#)、および[表 3-7](#) を参照してください。

## 詳細なディレクトリー構造

次の表は、インテル® MKL のアーキテクチャー固有ディレクトリーの構造の詳細をリストしています。interfaces ディレクトリーのメイクファイルを使用してこれらのディレクトリーに生成できるインターフェイス・ライブラリーのリストは、「[言語固有インターフェイスとインテル® MKL の使用](#)」を参照してください。doc ディレクトリーの内容は、「[ドキュメント・ディレクトリーの内容](#)」を参照してください。benchmarks ディレクトリーのサブディレクトリーの内容は、「[LINPACK ベンチマークと MP LINPACK ベンチマーク](#)」を参照してください。インテル® MKL 10.2 では、インテル® MKL バージョン 9.x 以前とのリンク行の互換性のために提供されていたライブラリーが削除されました。

**表 3-5 IA-32 アーキテクチャー固有のディレクトリー lib/32 の構造の詳細**

ファイル	内容
<b>スタティック・ライブラリー</b>	
<b>インターフェイス・レイヤー</b>	
libmkl_blas95.a	インテル® Fortran コンパイラー用 Fortran-95 インターフェイス・ライブラリー (BLAS)
libmkl_gf.a	GNU Fortran コンパイラー用インターフェイス・ライブラリー
libmkl_intel.a	インテル® コンパイラー用インターフェイス・ライブラリー <sup>1</sup>
libmkl_lapack95.a	インテル® Fortran コンパイラー用 Fortran-95 インターフェイス・ライブラリー (LAPACK)
<b>スレッド化レイヤー</b>	
libmkl_gnu_thread.a	GNU Fortran および C コンパイラー用スレッド化ライブラリー
libmkl_intel_thread.a	インテル® コンパイラー用スレッド化ライブラリー
libmkl_pgi_thread.a	PGI* コンパイラー用スレッド化ライブラリー
libmkl_sequential.a	逐次ライブラリー
<b>計算レイヤー</b>	
libmkl_cdft_core.a	FFT のクラスターバージョン
libmkl_core.a	IA-32 アーキテクチャー用カーネル・ライブラリー
libmkl_scalapack_core.a	ScaLAPACK ルーチン
libmkl_solver.a	反復法スパースソルバー、Trust Region ソルバー、および GMP ルーチン
libmkl_solver_sequential.a	反復法スパースソルバー、Trust Region ソルバー、および GMP ルーチンの逐次バージョン
<b>RTL</b>	
libguide.a	スタティック・リンク用インテルの OpenMP レガシー・ランタイム・ライブラリー
libiomp5.a	スタティック・リンク用インテルの OpenMP 互換ランタイム・ライブラリー
libmkl_blacs.a	以下の MPICH バージョンをサポートする BLACS ルーチン。 <ul style="list-style-type: none"> <li>Myricom* MPICH バージョン 1.2.5.10</li> <li>ANL* MPICH バージョン 1.2.5.2</li> </ul>
libmkl_blacs_intelmpi.a	インテル® MPI 2.0/3.x および MPICH2 をサポートする BLACS ルーチン
libmkl_blacs_intelmpi20.a	lib/32/libmkl_blacs_intelmpi.a へのソフトリンク
libmkl_blacs_openmpi.a	OpenMPI をサポートする BLACS ルーチン

表 3-5 IA-32 アーキテクチャー固有のディレクトリー lib/32 の構造の詳細 ( 続き )

ファイル	内容
<b>ダイナミック・ライブラリー</b>	
<b>インターフェイス・レイヤー</b>	
libmkl_gf.so	GNU Fortran コンパイラー用インターフェイス・ライブラリー
libmkl_intel.so	インテル® コンパイラー用インターフェイス・ライブラリー <sup>1</sup>
<b>スレッド化レイヤー</b>	
libmkl_gnu_thread.so	GNU Fortran および C コンパイラー用スレッド化ライブラリー
libmkl_intel_thread.so <sup>1</sup>	インテル® コンパイラー用スレッド化ライブラリー
libmkl_pgi_thread.so	PGI コンパイラー用スレッド化ライブラリー
libmkl_sequential.so	逐次ライブラリー
<b>計算レイヤー</b>	
libmkl_core.so	プロセッサ固有カーネル・ライブラリーのダイナミック・ロード用ライブラリー・ディスパッチャー
libmkl_def.so	デフォルト・カーネル・ライブラリー ( インテル® Pentium® プロセッサ、インテル® Pentium® Pro プロセッサ、インテル® Pentium® II プロセッサ、インテル® Pentium® III プロセッサ )
libmkl_lapack.so	LAPACK および DSS/PARDISO ルーチンとドライバ
libmkl_p4.so	インテル® Pentium® 4 プロセッサ用カーネル・ライブラリー
libmkl_p4m.so	インテル® Core™ マイクロアーキテクチャー・ベースのプロセッサ用カーネル・ライブラリー (mkl_p4p.so が対応しているインテル® Core™ Duo プロセッサおよびインテル® Core™ Solo プロセッサを除く )
libmkl_p4m3.so	インテル® Core™ i7 プロセッサ用カーネル・ライブラリー
libmkl_p4p.so	ストリーミング SIMD 拡張命令 3 (SSE3) 対応インテル® Pentium® 4 プロセッサ用カーネル・ライブラリー
libmkl_scalapack_core.so	ScaLAPACK ルーチン
libmkl_vml_def.so	古いインテル® Pentium® プロセッサ用デフォルトカーネルの VML/VSL 部分
libmkl_vml_ia.so	新しいインテル® アーキテクチャー・プロセッサ用 VML/VSL デフォルトカーネル
libmkl_vml_p4.so	インテル® Pentium® 4 プロセッサ用カーネルの VML/VSL 部分
libmkl_vml_p4m.so	インテル® Core™ マイクロアーキテクチャー・ベースのプロセッサ用 VML/VSL
libmkl_vml_p4m2.so	45nm Hi-k インテル® Core™2 プロセッサ・ファミリーおよびインテル® Xeon® プロセッサ・ファミリー用 VML/VSL
libmkl_vml_p4m3.so	インテル® Core™ i7 プロセッサ用 VML/VSL
libmkl_vml_p4p.so	ストリーミング SIMD 拡張命令 3 (SSE3) 対応インテル® Pentium® 4 プロセッサ用 VML/VSL
<b>RTL</b>	
libguide.so	ダイナミック・リンク用インテルの OpenMP レガシー・ランタイム・ライブラリー
libiomp5.so	ダイナミック・リンク用インテルの OpenMP 互換ランタイム・ライブラリー
libmkl_blacs_intelmpi.so	インテル® MPI 2.0/3.x および MPICH2 をサポートする BLACS ルーチン
locale/ja_JP/mkl_msg.cat	インテル® MKL メッセージカタログ ( 英語 )

表 3-5 IA-32 アーキテクチャ固有のディレクトリ lib/32 の構造の詳細 ( 続き )

ファイル	内容
locale/ja_JP/mkl_msg. cat	インテル® MKL メッセージカタログ (日本語)

1. Absoft\* コンパイラーでのリンクにも使用されます。

表 3-6 インテル® 64 アーキテクチャー固有のディレクトリー lib/em64t の構造の詳細

ファイル	内容
<b>スタティック・ライブラリー</b>	
<b>インターフェイス・レイヤー</b>	
libmkl_blas95_ilp64.a	インテル® Fortran コンパイラー用 Fortran-95 インターフェイス・ライブラリー (BLAS)。ILP64 インターフェイスをサポート
libmkl_blas95_lp64.a	インテル® Fortran コンパイラー用 Fortran-95 インターフェイス・ライブラリー (BLAS)。LP64 インターフェイスをサポート
libmkl_gf_ilp64.a	GNU Fortran および Absoft コンパイラー用 ILP64 インターフェイス・ライブラリー
libmkl_gf_lp64.a	GNU Fortran および Absoft コンパイラー用 LP64 インターフェイス・ライブラリー
libmkl_intel_ilp64.a	インテル® コンパイラー用 ILP64 インターフェイス・ライブラリー
libmkl_intel_lp64.a	インテル® コンパイラー用 LP64 インターフェイス・ライブラリー
libmkl_intel_sp2dp.a	インテル® コンパイラー用 SP2DP インターフェイス・ライブラリー
libmkl_lapack95_ilp64.a	インテル® Fortran コンパイラー用 Fortran-95 インターフェイス・ライブラリー (LAPACK)。ILP64 インターフェイスをサポート
libmkl_lapack95_lp64.a	インテル® Fortran コンパイラー用 Fortran-95 インターフェイス・ライブラリー (LAPACK)。LP64 インターフェイスをサポート
<b>スレッド化レイヤー</b>	
libmkl_gnu_thread.a	GNU Fortran および C コンパイラー用スレッド化ライブラリー
libmkl_intel_thread.a	インテル® コンパイラー用スレッド化ライブラリー
libmkl_pgi_thread.a	PGI コンパイラー用スレッド化ライブラリー
libmkl_sequential.a	逐次ライブラリー
<b>計算レイヤー</b>	
libmkl_cdft_core.a	FFT のクラスターバージョン
libmkl_core.a	インテル® 64 アーキテクチャー用カーネル・ライブラリー
libmkl_scalapack_ilp64.a	ILP64 インターフェイスをサポートする ScaLAPACK ルーチン・ライブラリー
libmkl_scalapack_lp64.a	LP64 インターフェイスをサポートする ScaLAPACK ルーチン・ライブラリー
libmkl_solver_ilp64.a	ILP64 インターフェイスをサポートする反復法スパースソルバーおよび Trust Region ソルバー・ルーチン・ライブラリー
libmkl_solver_ilp64_sequential.a	ILP64 インターフェイスをサポートする反復法スパースソルバーおよび Trust Region ソルバー・ルーチン・ライブラリーの逐次バージョン
libmkl_solver_lp64.a	LP64 インターフェイスをサポートする反復法スパースソルバー、Trust Region ソルバー、および GMP ルーチン・ライブラリー
libmkl_solver_lp64_sequential.a	LP64 インターフェイスをサポートする反復法スパースソルバー、Trust Region ソルバー、および GMP ルーチン・ライブラリーの逐次バージョン



表 3-6 インテル® 64 アーキテクチャー固有のディレクトリー lib/em64t の構造の詳細 ( 続き )

ファイル	内容
<b>RTL</b>	
libguide.a	スタティック・リンク用インテルの OpenMP レガシー・ランタイム・ライブラリー
libiomp5.a	スタティック・リンク用インテルの OpenMP 互換ランタイム・ライブラリー
libmkl_blacs_ilp64.a	以下の MPICH バージョンをサポートする BLACS ルーチンの ILP64 バージョン。 <ul style="list-style-type: none"> <li>Myricom* MPICH バージョン 1.2.5.10</li> <li>ANL* MPICH バージョン 1.2.5.2</li> </ul>
libmkl_blacs_intelmpi_ilp64.a	インテル® MPI 2.0/3.x および MPICH2 をサポートする BLACS ルーチンの ILP64 バージョン
libmkl_blacs_intelmpi_lp64.a	インテル® MPI 2.0/3.x、および MPICH2 をサポートする BLACS ルーチンの LP64 バージョン
libmkl_blacs_intelmpi20_ilp64.a	lib/em64t/libmkl_blacs_intelmpi_ilp64.a へのソフトリンク
libmkl_blacs_intelmpi20_lp64.a	lib/em64t/libmkl_blacs_intelmpi_lp64.a へのソフトリンク
libmkl_blacs_lp64.a	以下の MPICH バージョンをサポートする BLACS ルーチンの LP64 バージョン。 <ul style="list-style-type: none"> <li>Myricom* MPICH バージョン 1.2.5.10</li> <li>ANL* MPICH バージョン 1.2.5.2</li> </ul>
libmkl_blacs_openmpi_ilp64.a	OpenMPI をサポートする BLACS ルーチンの ILP64 バージョン
libmkl_blacs_openmpi_lp64.a	OpenMPI をサポートする BLACS ルーチンの LP64 バージョン
libmkl_blacs_sgimpt_ilp64.a	SGI MPT をサポートする BLACS ルーチンの ILP64 バージョン
libmkl_blacs_sgimpt_lp64.a	SGI MPT をサポートする BLACS ルーチンの LP64 バージョン
<b>ダイナミック・ライブラリー</b>	
<b>インターフェイス・レイヤー</b>	
libmkl_gf_ilp64.so	GNU Fortran および Absoft コンパイラー用 ILP64 インターフェイス・ライブラリー
libmkl_gf_lp64.so	GNU Fortran および Absoft コンパイラー用 LP64 インターフェイス・ライブラリー
libmkl_intel_ilp64.so	インテル® コンパイラー用 ILP64 インターフェイス・ライブラリー
libmkl_intel_lp64.so	インテル® コンパイラー用 LP64 インターフェイス・ライブラリー
libmkl_intel_sp2dp.so	インテル® コンパイラー用 SP2DP インターフェイス・ライブラリー
<b>スレッド化レイヤー</b>	
libmkl_gnu_thread.so	GNU Fortran および C コンパイラー用スレッド化ライブラリー
libmkl_intel_thread.so	インテル® コンパイラー用スレッド化ライブラリー
libmkl_pgi_thread.so	PGI コンパイラー用スレッド化ライブラリー
libmkl_sequential.so	逐次ライブラリー

表 3-6 インテル® 64 アーキテクチャー固有のディレクトリー lib/em64t の構造の詳細 ( 続き )

ファイル	内容
<b>計算レイヤー</b>	
libmkl_avx.so	インテル® Advanced Vector Extensions ( インテル® AVX) 用に最適化されたカーネル
libmkl_core.so	プロセッサ固有カーネル・ライブラリーのダイナミック・ロード用ライブラリー・ディスパッチャー
libmkl_def.so	デフォルト・カーネル・ライブラリー
libmkl_mc.so	インテル® Core™ マイクロアーキテクチャー・ベースのプロセッサ用カーネル・ライブラリー
libmkl_mc3.so	インテル® Core™ i7 プロセッサ用カーネル・ライブラリー
libmkl_lapack.so	LAPACK および DSS/PARDISO ルーチンとドライバー
libmkl_scalapack_ilp64.so	ILP64 インターフェイスをサポートする ScaLAPACK ルーチン・ライブラリー
libmkl_scalapack_lp64.so	LP64 インターフェイスをサポートする ScaLAPACK ルーチン・ライブラリー
libmkl_vml_avx.so	インテル® Advanced Vector Extensions ( インテル® AVX) 用に最適化された VML/VSL
libmkl_vml_def.so	デフォルトカーネルの VML/VSL 部分
libmkl_vml_mc.so	インテル® Core™ マイクロアーキテクチャー・ベースのプロセッサ用 VML/VSL
libmkl_vml_mc3.so	インテル® Core™ i7 プロセッサ用 VML/VSL
libmkl_vml_p4n.so	インテル® 64 アーキテクチャー対応インテル® Xeon® プロセッサ用 VML/VSL
libmkl_vml_mc2.so	45nm Hi-k インテル® Core™2 プロセッサ・ファミリーおよびインテル® Xeon® プロセッサ・ファミリー用 VML/VSL
<b>RTL</b>	
libguide.so	ダイナミック・リンク用インテルの OpenMP レガシー・ランタイム・ライブラリー
libiomp5.so	ダイナミック・リンク用インテルの OpenMP 互換ランタイム・ライブラリー
libmkl_intelmpi_ilp64.so	インテル® MPI 2.0/3.x および MPICH2 をサポートする BLACS ルーチンの ILP64 バージョン
libmkl_intelmpi_lp64.so	インテル® MPI 2.0/3.x、および MPICH2 をサポートする BLACS ルーチンの LP64 バージョン
locale/ja_JP/mkl_msg.cat	インテル® MKL メッセージカタログ ( 英語 )
locale/ja_JP/mkl_msg.cat	インテル® MKL メッセージカタログ ( 日本語 )

表 3-7 IA-64 アーキテクチャー固有のディレクトリー lib/64 の構造の詳細

ファイル	内容
<b>スタティック・ライブラリー</b>	
<b>インターフェイス・レイヤー</b>	
libmkl_blas95_ilp64.a	インテル® Fortran コンパイラー用 Fortran-95 インターフェイス・ライブラリー (BLAS)。ILP64 インターフェイスをサポート
libmkl_blas95_lp64.a	インテル® Fortran コンパイラー用 Fortran-95 インターフェイス・ライブラリー (BLAS)。LP64 インターフェイスをサポート
libmkl_intel_ilp64.a	インテル® コンパイラー用 ILP64 インターフェイス・ライブラリー
libmkl_intel_lp64.a	インテル® コンパイラー用 LP64 インターフェイス・ライブラリー

表 3-7 IA-64 アーキテクチャー固有のディレクトリー lib/64 の構造の詳細 ( 続き )

ファイル	内容
libmkl_intel_sp2dp.a	インテル® コンパイラー用 SP2DP インターフェイス・ライブラリー
libmkl_gf_ilp64.a	GNU Fortran コンパイラー用 ILP64 インターフェイス・ライブラリー
libmkl_gf_lp64.a	GNU Fortran コンパイラー用 LP64 インターフェイス・ライブラリー
libmkl_lapack95_ilp64.a	インテル® Fortran コンパイラー用 Fortran-95 インターフェイス・ライブラリー (LAPACK)。ILP64 インターフェイスをサポート
libmkl_lapack95_lp64.a	インテル® Fortran コンパイラー用 Fortran-95 インターフェイス・ライブラリー (LAPACK)。LP64 インターフェイスをサポート
<b>スレッド化レイヤー</b>	
libmkl_intel_thread.a	インテル® コンパイラー用スレッド化ライブラリー
libmkl_gnu_thread.a	GNU Fortran および C コンパイラー用スレッド化ライブラリー
libmkl_sequential.a	逐次ライブラリー
<b>計算レイヤー</b>	
libmkl_cdft_core.a	FFT のクラスターバージョン
libmkl_core.a	IA-64 アーキテクチャー用カーネル・ライブラリー
libmkl_scalapack_ilp64.a	ILP64 インターフェイスをサポートする ScaLAPACK ルーチン・ライブラリー
libmkl_scalapack_lp64.a	LP64 インターフェイスをサポートする ScaLAPACK ルーチン・ライブラリー
libmkl_solver_ilp64.a	ILP64 インターフェイスをサポートする反復法スパースソルバーおよび Trust Region ソルバー・ルーチン・ライブラリー
libmkl_solver_ilp64_sequential.a	ILP64 インターフェイスをサポートする反復法スパースソルバーおよび Trust Region ソルバー・ルーチン・ライブラリーの逐次バージョン
libmkl_solver_lp64.a	LP64 インターフェイスをサポートする反復法スパースソルバー、Trust Region ソルバー、および GMP ルーチン・ライブラリー
libmkl_solver_lp64_sequential.a	LP64 インターフェイスをサポートする反復法スパースソルバー、Trust Region ソルバー、および GMP ルーチン・ライブラリーの逐次バージョン

表 3-7 IA-64 アーキテクチャー固有のディレクトリー lib/64 の構造の詳細 ( 続き )

ファイル	内容
<b>RTL</b>	
libguide.a	スタティック・リンク用インテルの OpenMP レガシー・ランタイム・ライブラリー
libiomp5.a	スタティック・リンク用インテルの OpenMP 互換ランタイム・ライブラリー
libmkl_blacs_ilp64.a	以下の MPICH バージョンをサポートする BLACS ルーチンの ILP64 バージョン。 <ul style="list-style-type: none"> <li>Myricom* MPICH バージョン 1.2.5.10</li> <li>ANL* MPICH バージョン 1.2.5.2</li> </ul>
libmkl_blacs_intelmpi_ilp64.a	インテル® MPI 2.0/3.x および MPICH2 をサポートする BLACS ルーチンの ILP64 バージョン
libmkl_blacs_intelmpi_lp64.a	インテル® MPI 2.0/3.x および MPICH2 をサポートする BLACS ルーチンの LP64 バージョン
libmkl_blacs_intelmpi20_ilp64.a	lib/64/libmkl_blacs_intelmpi_ilp64.a へのソフトリンク
libmkl_blacs_intelmpi20_lp64.a	lib/64/libmkl_blacs_intelmpi_lp64.a へのソフトリンク
libmkl_blacs_lp64.a	以下の MPICH バージョンをサポートする BLACS ルーチンの LP64 バージョン。 <ul style="list-style-type: none"> <li>Myricom* MPICH バージョン 1.2.5.10</li> <li>ANL* MPICH バージョン 1.2.5.2</li> </ul>
libmkl_blacs_openmpi_ilp64.a	OpenMPI をサポートする BLACS ルーチンの ILP64 バージョン
libmkl_blacs_openmpi_lp64.a	OpenMPI をサポートする BLACS ルーチンの LP64 バージョン
libmkl_blacs_sgimpt_ilp64.a	SGI MPT をサポートする BLACS ルーチンの ILP64 バージョン
libmkl_blacs_sgimpt_lp64.a	SGI MPT をサポートする BLACS ルーチンの LP64 バージョン
<b>ダイナミック・ライブラリー</b>	
<b>インターフェイス・レイヤー</b>	
libmkl_gf_ilp64.so	GNU Fortran コンパイラー用 ILP64 インターフェイス・ライブラリー
libmkl_gf_lp64.so	GNU Fortran コンパイラー用 LP64 インターフェイス・ライブラリー
libmkl_intel_ilp64.so	インテル® コンパイラー用 ILP64 インターフェイス・ライブラリー
libmkl_intel_lp64.so	インテル® コンパイラー用 LP64 インターフェイス・ライブラリー
libmkl_intel_sp2dp.so	インテル® コンパイラー用 SP2DP インターフェイス・ライブラリー
<b>スレッド化レイヤー</b>	
libmkl_gnu_thread.so	GNU Fortran および C コンパイラー用スレッド化ライブラリー
libmkl_intel_thread.so	インテル® コンパイラー用スレッド化ライブラリー
libmkl_sequential.so	逐次ライブラリー
<b>計算レイヤー</b>	
libmkl_core.so	プロセッサ固有カーネル・ライブラリーのダイナミック・ロード用ライブラリー・ディスパッチャー
libmkl_i2p.so	IA-64 アーキテクチャー用カーネル・ライブラリー
libmkl_lapack.so	LAPACK および DSS/PARDISO ルーチンとドライバー

表 3-7 IA-64 アーキテクチャー固有のディレクトリー lib/64 の構造の詳細 ( 続き )

ファイル	内容
libmkl_scalapack_ilp64.so	ILP64 インターフェイスをサポートする ScaLAPACK ルーチン・ライブラリー
libmkl_scalapack_lp64.so	LP64 インターフェイスをサポートする ScaLAPACK ルーチン・ライブラリー
libmkl_vml_i2p.so	IA-64 アーキテクチャー用 VML カーネル
<b>RTL</b>	
libguide.so	ダイナミック・リンク用インテルの OpenMP レガシー・ランタイム・ライブラリー
libiomp5.so	ダイナミック・リンク用インテルの OpenMP 互換ランタイム・ライブラリー
libmkl_blacs_intelmpi_ilp64.so	インテル® MPI 2.0/3.x および MPICH2 をサポートする BLACS ルーチンの ILP64 バージョン
libmkl_blacs_intelmpi_lp64.so	インテル® MPI 2.0/3.x および MPICH2 をサポートする BLACS ルーチンの LP64 バージョン
locale/en_US/mkl_msg.cat	インテル® MKL メッセージカタログ ( 英語 )
locale/ja_JP/mkl_msg.cat	インテル® MKL メッセージカタログ ( 日本語 )

## インテル® MKL ドキュメントへのアクセス

このセクションでは、インテル® MKL ドキュメント・ディレクトリーの内容とライブラリーの man ページへのアクセス方法を説明します。

### ドキュメント・ディレクトリーの内容

表 3-8 は、インテル® MKL 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をリストしています。

表 3-8 doc ディレクトリーの内容

ファイル名	内容
Install.txt	インテル® MKL インストール・ガイド
mkl_documentation.htm	インテル® MKL ドキュメントの概要とリンク
mklEULA.txt	インテル® MKL の使用許諾契約書
mklman.pdf	インテル® MKL リファレンス・マニュアル
mklman90_j.pdf	インテル® MKL 9.0 リファレンス・マニュアル ( 日本語 )
mklsupport.txt	テクニカルサポートで使用するパッケージ番号の情報
redist.txt	再配布可能ファイルのリスト
Release_Notes.pdf	インテル® MKL リリースノート
userguide.pdf	インテル® MKL ユーザーズガイド ( 本ドキュメント )

### Man ページへのアクセス

インストール中に、インテル® MKL 関数の man ページがインテル® MKL インストール・ディレクトリーのサブディレクトリー man/ja\_JP/man3 にコピーされます。

コマンドシェルで man コマンドを使用して man ページを表示するには、man ページのディレクトリーを MANPATH 環境変数に追加してください ( 「[環境変数の設定](#)」を参照 ) 。

環境変数を設定した後、次のように man コマンドを入力します。

man < 関数ベース名 >

本リリースでは、< 関数ベース名 > にデータ型、精度、または関数ドメインを示す接頭辞を省略した関数名を入力してください。

例：

- BLAS 関数 ddot の場合、man dot と入力します。
- ScaLAPACK 関数 pzgeql2 の場合、man pgeql2 と入力します。
- FFT 関数 DftiCommitDescriptor の場合、man CommitDescriptor と入力します。



---

**注：**man コマンドの関数名の大文字と小文字は区別されます。

---

# 開発環境の構成

本章は、インテル® マス・カーネル・ライブラリー (インテル® MKL) を使用するための開発環境の構成方法を説明します。

環境変数 INCLUDE、MKLROOT、LD\_LIBRARY\_PATH、MANPATH、LIBRARY\_PATH、CPATH、FPATH、および NLSPATH の設定方法については、第 2 章を参照してください。これらの変数をスタートアップ時に自動的に設定する方法は、[「環境変数の自動設定」](#) セクションを参照してください。

スレッド化用に環境変数を設定する方法についての情報は、[「OpenMP 環境変数を使用したスレッド数の設定」](#) を参照してください。

## 環境変数の自動設定

環境変数 INCLUDE、MKLROOT、LD\_LIBRARY\_PATH、MANPATH、LIBRARY\_PATH、CPATH、FPATH、および NLSPATH をスタートアップ時に自動的に設定するには、シェル・プロファイルに mklvars\*.sh を追加します。これを行うと、ログインするたびに適切なインテル® MKL ディレクトリーのパスが設定されます。

ローカル・ユーザー・アカウントで以下のファイルを編集し、変数をエクスポートする直前のパスの設定セクションに適切なスクリプトを追加してください。

- bash:  
~/.bash\_profile、~/.bash\_login または ~/.profile  
  
# bash の MKL 環境を設定します。  
． <MKL のインストール先の絶対パス>/tools/environment/mklvars<arch>.sh
- sh:  
~/.profile  
  
# sh の MKL 環境を設定します。  
． <MKL のインストール先の絶対パス>/tools/environment/mklvars<arch>.sh
- csh:  
~/.login  
  
# csh の MKL 環境を設定します。  
． <MKL のインストール先の絶対パス>/tools/environment/mklvars<arch>.csh

上記の mklvars<arch> は、mklvars32、mklvarsem64t または mklvars64 に置き換えてください。

スーパーユーザー権限がある場合、同じコマンドを /etc/profile (bash および sh の場合) または /etc/csh.login (csh の場合) のシステムファイルに追加してもかまいません。

ログイン時の問題を回避するため、インテル® MKL をアンインストールする前には、すべてのプロファイル・ファイルから上記のコマンドを削除してください。

## インテル® MKL を Eclipse\* IDE CDT でリンクする場合の構成

このセクションでは、インテル® MKL を Eclipse IDE C/C++ 開発ツール (CDT) 3.x および 4.0 でリンクする場合の構成を説明します。



**ヒント:** インテル® MKL を CDT とリンクすると、Eclipse が提供するコードアシスト機能を利用できます。Eclipse ヘルプのコード/コードアシストの説明を参照してください。

## Eclipse IDE CDT 4.0 でリンクする場合の構成

インテル® MKL を Eclipse CDT 4.0 でリンクするには、以下の手順を実行してください。

1. ツールチェーン/コンパイラ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、**[C/C++ General] > [Paths and Symbols]** プロパティ・ページの **[Includes]** タブを開いて、インテル® MKL インクルード・パス (`<mk1 ディレクトリー>/include`) を設定します。
2. ツールチェーン/コンパイラ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・ページの **[Library Paths]** タブを開いて、ターゲット・アーキテクチャー用のインテル® MKL ライブラリーのパス (例えば、`<mk1 ディレクトリー>/lib/em64t`) を設定します。
3. 一部のビルドでは、**[C/C++ Build] > [Settings]** プロパティ・ページの **[Tool Settings]** タブを開いて、アプリケーションとリンクするインテル® MKL ライブラリーの名前 (例えば、`mk1_solver_lp64` および `mk1_core`) を指定します。ライブラリー・ファイルの名前ではなく、ライブラリー名を指定するコンパイラーでは、“lib” 接頭辞と “a” 拡張子は省略してください。ライブラリーの選択方法は、「[リンクする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 IDE CDT 3.x の構成

インテル® MKL を Eclipse CDT 3.x でリンクするには、以下の手順を実行してください。

- Standard Make プロジェクトの場合：
  1. **[C/C++ Include Paths and Symbols]** プロパティ・ページを開いて、インテル® MKL インクルード・パス (`<mk1 ディレクトリー>/include`) を設定します。
  2. **[C/C++ Project Paths]** プロパティ・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® MKL ライブラリーを設定します (例えば、`<mk1 ディレクトリー>/lib/em64t/libmkl_intel_lp64.a`、`<mk1 ディレクトリー>/lib/em64t/libmkl_intel_thread.a`、および `<mk1 ディレクトリー>/lib/em64t/libmkl_core.a`)。ライブラリーの選択方法は、「[リンクする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、特定のビルド用に設定を指定できます。次の操作を行ってください。
  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® MKL インクルード・パス (`<mk1 ディレクトリー>/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・アーキテクチャー用のインテル® MKL ライブラリーのパス (例えば、`<mk1 ディレクトリー>/lib/em64t`) 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® MKL ライブラリーの名前 (例えば、`mk1_solver_lp64` および `mk1_core`) を指定します。ライブラリー・ファイルの名前ではなく、ライブラリー名を指定するコンパイラーでは、“lib” 接頭辞と “a” 拡張子は省略してください。ライブラリーの選択方法は、「[リンクするライブラリーの選択](#)」を参照してください。

## Out-of-Core (OOC) DSS/PARDISO\* ソルバーの構成

OOC DSS/PARDISO ソルバーの構成ファイルを使用するときは、ファイルのパス行の最大長が 1000 文字であることに注意してください。

詳細は、『*インテル® MKL リファレンス・マニュアル*』の「スパース・ソルバー・ルーチン」の章を参照してください。

# アプリケーションと インテル® マス・カーネル・ ライブラリーのリンク

## 5

本章は、アプリケーションとインテル® マス・カーネル・ライブラリー (インテル® MKL) Linux® OS 版のリンクについて説明します。アプリケーションとリンクするライブラリーの情報、およびリンク例が含まれます。また、カスタム共有オブジェクトの構築についても説明します。

レイヤー・リンク・モデルを使用するインテル® MKL をリンクするには、インターフェイス・レイヤーから1つのライブラリー、スレッド化レイヤーから1つのライブラリー、計算レイヤー・ライブラリーからライブラリーを選択します。必要な場合は、ランタイム・ライブラリーを追加します。[表 5-1](#) に、インテル® MKL でリンクする一般的なライブラリーのセットのリストを示します。

表 5-1 一般的なリンク・ライブラリー

	インターフェイス・ レイヤー	スレッド化レイヤー	計算レイヤー	RTL
<b>IA-32</b> アーキテクチャー、 スタティック・ リンク	libmkl_intel.a	libmkl_intel_ thread.a	libmkl_core.a	libiomp5.so
<b>IA-32</b> アーキテクチャー、 ダイナミック・ リンク	libmkl_intel.s o	libmkl_intel_ thread.so	libmkl_core.so	libiomp5.so
インテル® <b>64</b> および <b>IA-64</b> アーキテクチャー、 スタティック・ リンク	libmkl_intel_ lp64.a	libmkl_intel_ thread_lp64.a	libmkl_core.a	libiomp5.so
インテル® <b>64</b> および <b>IA-64</b> アーキテクチャー、 ダイナミック・ リンク	libmkl_intel_ lp64.so	libmkl_intel_ thread_lp64.so	libmkl_core.so	libiomp5.so

上記のライブラリーの例外および代替情報については、「[リンクするライブラリーの選択](#)」を参照してください。

「[リンク行のライブラリーのリスト](#)」も参照してください。

## リンク行のライブラリーのリスト

インテル® MKL ライブラリーをリンクするには、以下のようにリンク行にパスとライブラリーを指定します。



**注:** 以下の構文はダイナミック・リンク用のものです。スタティック・リンクの場合は、“-l”で始まるライブラリーをライブラリー・ファイルのパスに置換してください。例えば、-lmkl\_coreを\$MKL\_PATH/libmkl\_core.a (\$MKL\_PATHは適切なユーザー定義環境変数)に置き換えます。「[リンクの例](#)」セクションの例を参照してください。

#### < リンクするファイル >

```
-L<MKL パス> -I<MKL インクルード>
[-I<MKL インクルード>/{32|em64t|{ilp64|lp64}|64/{ilp64|lp64}}]
[-lmkl_blas{95|95_ilp64|95_lp64}]
[-lmkl_lapack{95|95_ilp64|95_lp64}]
[< クラスター・コンポーネント >]
-lmkl_{intel|intel_ilp64|intel_lp64|intel_sp2dp|gf|gf_ilp64|gf_lp64}
-lmkl_{intel_thread|gnu_thread|pgi_thread|sequential}
[-lmkl_solver_{lp64|ilp64}]
[-lmkl_lapack] -lmkl_core
{-liomp5|-lguide} [-lpthread] [-lm]
```

この構文の使用法の詳細およびインテル® MKL の使用法に応じた推奨ライブラリーについては、「[リンクするライブラリーの選択](#)」を参照してください。

< クラスター・コンポーネント > ライブラリーのリンクについては、「[インテル® マス・カーネル・ライブラリー・クラスター・ソフトウェアの使用](#)」を参照してください。

スタティック・リンクでは、クラスター・コンポーネント、インターフェイス、スレッド化、および計算ライブラリーをグループ化シンボルで囲みます (例えば、-Wl,--start-group \$MKL\_PATH/libmkl\_cdft\_core.a \$MKL\_PATH/libmkl\_blacs\_intelmpi\_ilp64.a \$MKL\_PATH/libmkl\_intel\_ilp64.a \$MKL\_PATH/libmkl\_intel\_thread.a \$MKL\_PATH/libmkl\_core.a -Wl,--end-group)。「[リンクの例](#)」セクションの例を参照してください。

リンク行でライブラリーをリストする順序は非常に重要です (グループ化シンボルで囲まれたライブラリーを除く)。

## リンクするライブラリーの選択

このセクションでは、インテル® MKL の使用法に応じて、リンクするライブラリーを推奨します。リンクの詳細は、サブセクションで説明します。

[Fortran 95 インターフェイス・ライブラリーのリンク](#)

[スレッド化ライブラリーのリンク](#)

[計算ライブラリーのリンク](#)

[コンパイラー・サポート RTL のリンク](#)

[システム・ライブラリーのリンク](#)

[リンクの例](#)

## Fortran 95 インターフェイス・ライブラリーのリンク

libmkl\_blas95.a、libmkl\_blas95\_lp64.a、libmkl\_blas95\_ilp64.a、libmkl\_lapack95.a、libmkl\_lapack95\_lp64.a、および libmkl\_lapack95\_ilp64.a ライブラリーには、コンパイラーに依存する、BLAS と LAPACK 用の Fortran 95 インターフェイスが

それぞれ含まれています。インテル® MKL パッケージでは、これらのライブラリーは、インテル® Fortran コンパイラー用に事前に構築されています。異なるコンパイラーを使用する場合は、インターフェイスを使用する前に、これらのライブラリーを構築してください。(ライブラリーの構築についての詳細とソースコードが配布されている理由については、「[Fortran 95 インターフェイス、LAPACK および BLAS](#)」および「[コンパイラー依存の関数と Fortran 90 モジュール](#)」を参照してください。)

## スレッド化ライブラリーのリンク

インテル® MKL がサポートしているいくつかのコンパイラーは、スレッド化に OpenMP\* テクノロジーを使用しています。インテル® MKL は、バージョン 10.0 から、これらのコンパイラーが提供する OpenMP の実装をサポートしています。これらのコンパイラーでコンパイルされた OpenMP を使用するアプリケーションが、インテル® コンパイラーでスレッド化されたインテル® MKL の以前のバージョンを呼び出すと、スレッド化ライブラリーに互換性がないためにパフォーマンス問題やエラーが発生することがあります。インテル® MKL 10.0 では、この問題に対応するべく、スレッド化レイヤーとコンパイラー・サポート・ランタイム・ライブラリー (RTL) が提供されました。

**スレッド化レイヤー:** ライブラリーの内部構造のために、スレッド化はわずかなコードで表現されます。このコードは異なるコンパイラー (Linux の場合、インテル® コンパイラー、gnu および PGI\* コンパイラー) でコンパイルされます。適切なスレッド化レイヤーでリンクする必要があります。

**RTL:** このレイヤーには、インテル® コンパイラーのランタイム・ライブラリー (インテルの OpenMP 互換ランタイム・ライブラリー libiomp およびインテルの OpenMP レガシー・ランタイム・ライブラリー libguide) が含まれます。互換ライブラリー libiomp は libguide の拡張で、追加のスレッド化ライブラリー (Linux の場合 GNU) をサポートします。つまり、GNU コンパイラーを使用してスレッド化されたプログラムは、インテル® MKL および libiomp と安全にリンクし、効率的かつ効果的に実行することができます。このため、libguide ではなく libiomp を使用することを推奨します。

[表 5-2](#) は、使用されているスレッド化コンパイラーと状況別に、インテル® MKL を使用した場合に選択するスレッド化ライブラリーと RTL を示したものです (スタティックのみ)。

表 5-2 スレッド化ライブラリーの選択

コンパイラー	マルチ スレッド?	スレッド化レイヤー	RTL を推奨	内容
Intel	問題になり ません	libmkl_intel_thread.a	libiomp5.so	
PGI	○	libmkl_pgi_thread.a または libmkl_sequential.a	PGI が提供	libmkl_sequential.a を使用すると、 インテル® MKL 呼び出しからスレッド化が削 除されます。
PGI	X	libmkl_intel_thread.a	libiomp5.so	
PGI	X	libmkl_pgi_thread.a	PGI が提供	
PGI	X	libmkl_sequential.a	なし	
gnu	○	libmkl_gnu_thread.a	libiomp5.so また は GNU OpenMP ランタ イム・ライブラリー	libiomp5 は優れた スケーリング・パ フォーマンスを提供し ます。
gnu	○	libmkl_sequential.a	なし	
gnu	X	libmkl_intel_thread.a	libiomp5.so	
その他	○	libmkl_sequential.a	なし	
その他	X	libmkl_intel_thread.a	libiomp5.so	

## 計算ライブラリーのリンク

一般的に、レイヤー・リンク・モデルでは、アプリケーションを 1 つの計算レイヤー・ライブラリーとリンクする必要があります。しかし、特定のインテル® MKL 関数ドメインでは複数の計算リンク・ライブラリーが必要です。

インテル® MKL 関数ドメイン別に、リンク行で指定する必要がある計算レイヤー・ライブラリーを [表 5-3](#) にリストします。ScaLAPACK とクラスター FFT のリンクについては、「[ScaLAPACK および クラスター FFT とのリンク](#)」も参照してください。

**表 5-3 リンクする計算ライブラリー、関数ドメイン別**

関数ドメイン	IA-32 アーキテクチャー		インテル® 64 または IA-64 アーキテクチャー	
	スタティック	ダイナミック	スタティック	ダイナミック
BLAS、 CBLAS、 スパース BLAS、 LAPACK、 VML、VSL、 FFT、 三角変換関数、 ポアソン・ ライブラリー	libmkl_core.a	libmkl_core.so	libmkl_core.a	libmkl_core.so
反復法 スパース ソルバー、 Trust Region ソルバー、 および GMP ルーチン	libmkl_solver.a または libmkl_solver_ sequential.a libmkl_core.a	n/a	以下を参照	n/a
反復法 スパース ソルバー、 Trust Region ソルバー、 および GMP ルーチン、 LP64 インター フェイス	n/a	n/a	libmkl_solver_ lp64.a または libmkl_solver_ lp64_ sequential.a  libmkl_core.a	n/a
反復法 スパース ソルバー、 Trust Region ソルバー、 および GMP ルーチン、 ILP64 インター フェイス	n/a	n/a	libmkl_solver_ ilp64.a または libmkl_solver_ ilp64_ sequential.a  libmkl_core.a	n/a
直接法 スパース ソルバー / PARDISO ソルバー	libmkl_core.a	libmkl_lapack. so libmkl_core.so	libmkl_core.a	libmkl_lapack. so libmkl_core.so
ScaLAPACK <sup>1</sup>	libmkl_scalapack _core.a libmkl_core.a	libmkl_scalapack _core.so libmkl_lapack. so libmkl_core.so	以下を参照	以下を参照

表 5-3 リンクする計算ライブラリー、関数ドメイン別 ( 続き )

関数ドメイン	IA-32 アーキテクチャー		インテル® 64 または IA-64 アーキテクチャー	
	スタティック	ダイナミック	スタティック	ダイナミック
ScaLAPACK、 LP64 インター フェイス <sup>1</sup>	n/a	n/a	libmkl_scalapack _lp64.a libmkl_core.a	libmkl_scalapack _lp64.so libmkl_lapack. so libmkl_core.so
ScaLAPACK、 ILP64 インター フェイス <sup>1</sup>	n/a	n/a	libmkl_scalapack _ilp64.a libmkl_lapack. so libmkl_core.a	libmkl_scalapack_ ilp64.so libmkl_lapack. so libmkl_core.so
クラスター フーリエ変換 関数 <sup>1</sup>	libmkl_cdft_ core.a libmkl_core.a	n/a	libmkl_cdft_ core.a libmkl_core.a	n/a

1. 使用されている MPI に対応する BLACS ルーチンのライブラリーも追加する。詳細は、「[ScaLAPACK およびクラスター FFT とのリンク](#)」を参照。

以下の項目も参照してください。

[コンパイラー・サポート RTL のリンク](#)

## コンパイラ・サポート RTL のリンク

インテルの OpenMP 互換ランタイム・ライブラリー libiomp およびインテルの OpenMP レガシー・ランタイム・ライブラリー libguide は、ダイナミックにリンクすることを強く推奨します。OpenMP ランタイム・ライブラリーのスタティック・リンクは、ライブラリーの複数のコピーにソフトウェアがリンクされるため、推奨しません。リンクすると、パフォーマンス問題 (多すぎるスレッド数) が発生し、複数のコピーが初期化されるときに正当性問題が発生します。

ほかのライブラリーがスタティックにリンクされている場合でも、libiomp および libguide はダイナミックにリンクすることを推奨します。

libiomp/libguide をスタティックにリンクする場合、リンクする libiomp/libguide のバージョンは使用するコンパイラによって異なります。

- インテル® コンパイラを使用する場合、コンパイラに含まれているバージョンの libiomp/libguide を (-openmp オプションを使用して) リンクします。
- インテル® コンパイラを使用しない場合、インテル® MKL に含まれているバージョンの libiomp/libguide をリンクします。

libiomp/libguide のダイナミック・バージョンをリンクする場合 (推奨)、つまり libiomp5.so または libguide.so を使用する場合、LD\_LIBRARY\_PATH が正しく定義されていることを確認してください。詳細は、「[環境変数の設定](#)」を参照してください。

## システム・ライブラリーのリンク

インテル® MKL FFT、三角変換、またはポアソン、ラプラス、およびヘルムホルツ・ソルバー・ルーチンを使用するには、リンク行に "-lm" を追加して数学サポート・システム・ライブラリーをリンクします。

Linux では、libiomp/libguide はどちらもマルチスレッド用の標準 pthread ライブラリーに依存します。libiomp/libguide が必要な場合、リンク行の最後に -lpthread を追加する必要があります (ライブラリーをリストする順序は非常に重要です)。

## リンクの例

このセクションでは、IA-32、インテル® 64、および IA-64 アーキテクチャー・ベースのシステムでインテル® コンパイラを使用してリンクする場合の例を紹介します。これらの例では、<MKL パス> および <MKL インクルード> プレースホルダーは、ユーザーが定義した環境変数 \$MKLPATH および \$MKLINCLUDE に置換されています。第 9 章の ScaLAPACK およびクラスター FFT とのリンクの例も参照してください。

以下の例では、.f Fortran ソースファイルを使用しています。C/C++ ユーザーは、ソースファイルを .cpp (C++) または .c (C) に、ifort リンカーを icc に変更してください。

正しいリンク行を設定するために、Web ベースのリンク・アドバイザーが用意されています。<http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor> (英語) にアクセスしてください。

### IA-32 アーキテクチャー・ベースのシステムでのリンク

1. myprog.f とインテル® MKL の並列バージョンのスタティック・リンク。

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
-Wl,--start-group $MKLPATH/libmkl_intel.a
$MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a -Wl,--end-group
-liomp5 -lpthread
```
2. myprog.f とインテル® MKL の並列バージョンのダイナミック・リンク。

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
-lmkl_intel -lmkl_intel_thread -lmkl_core -liomp5 -lpthread
```
3. myprog.f とインテル® MKL の逐次バージョンのスタティック・リンク。



- ```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
-Wl,--start-group $MKLPATH/libmkl_intel.a
$MKLPATH/libmkl_sequential.a $MKLPATH/libmkl_core.a -Wl,--end-group
-lpthread
```
4. myprog.f とインテル® MKL の逐次バージョンのダイナミック・リンク。  

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
-lmkl_intel -lmkl_sequential -lmkl_core -lpthread
```
  5. myprog.f、Fortran 95 LAPACK インターフェイス<sup>1</sup>、インテル® MKL の並列バージョンのスタティック・リンク。  

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -I$MKLINCLUDE/32
-lmkl_lapack95
-Wl,--start-group $MKLPATH/libmkl_intel.a
$MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a -Wl,--end-group
-liomp5 -lpthread
```
  6. myprog.f、Fortran 95 BLAS インターフェイス<sup>1</sup>、インテル® MKL の並列バージョンのスタティック・リンク。  

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -I$MKLINCLUDE/32 -lmkl_blas95
-Wl,--start-group $MKLPATH/libmkl_intel.a
$MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a -Wl,--end-group
-liomp5 -lpthread
```
  7. myprog.f、反復法スパースソルバーの並列バージョン、インテル® MKL の並列バージョンのスタティック・リンク。  

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -lmkl_solver
-Wl,--start-group $MKLPATH/libmkl_intel.a
$MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a -Wl,--end-group
-liomp5 -lpthread
```
  8. myprog.f、反復法スパースソルバーの逐次バージョン、インテル® MKL の逐次バージョンのスタティック・リンク。  

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -lmkl_solver_sequential
-Wl,--start-group $MKLPATH/libmkl_intel.a
$MKLPATH/libmkl_sequential.a $MKLPATH/libmkl_core.a -Wl,--end-group
-lpthread
```

---

1. Fortran 95 LAPACK および BLAS インターフェイス・ライブラリーの構築方法については、「[Fortran 95 インターフェイス、LAPACK および BLAS](#)」を参照してください。

## インテル® 64 および IA-64 アーキテクチャー・ベースのシステムでのリンク

1. myprog.f と LP64 インターフェイスをサポートしているインテル® MKL の並列バージョンのスタティック・リンク。  

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
-Wl,--start-group $MKLPATH/libmkl_intel_lp64.a
$MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a -Wl,--end-group
-liomp5 -lpthread
```
2. myprog.f と LP64 インターフェイスをサポートしているインテル® MKL の並列バージョンのダイナミック・リンク。  

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
-lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread
```
3. myprog.f と LP64 インターフェイスをサポートしているインテル® MKL の逐次バージョンのスタティック・リンク。  

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
-Wl,--start-group $MKLPATH/libmkl_intel_lp64.a
$MKLPATH/libmkl_sequential.a $MKLPATH/libmkl_core.a -Wl,--end-group
-lpthread
```
4. myprog.f と LP64 インターフェイスをサポートしているインテル® MKL の逐次バージョンのダイナミック・リンク。  

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
-lmkl_intel_lp64 -lmkl_sequential -lmkl_core -lpthread
```
5. myprog.f と ILP64 インターフェイスをサポートしているインテル® MKL の並列バージョンのスタティック・リンク。  

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
-Wl,--start-group $MKLPATH/libmkl_intel_ilp64.a
$MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a -Wl,--end-group
-liomp5 -lpthread
```
6. myprog.f と ILP64 インターフェイスをサポートしているインテル® MKL の並列バージョンのダイナミック・リンク。  

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
-lmkl_intel_ilp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread
```
7. myprog.f、Fortran 95 LAPACK インターフェイス<sup>1</sup>、LP64 インターフェイスをサポートしているインテル® MKL の並列バージョンのスタティック・リンク。  

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -I$MKLINCLUDE/em64t/lp64
-lmkl_lapack95_lp64
-Wl,--start-group $MKLPATH/libmkl_intel_lp64.a
$MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a -Wl,--end-group
-liomp5 -lpthread
```
8. myprog.f、Fortran 95 BLAS インターフェイス<sup>1</sup>、LP64 インターフェイスをサポートしているインテル® MKL の並列バージョンのスタティック・リンク。  

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -I$MKLINCLUDE/em64t/lp64
-lmkl_blas95_lp64
-Wl,--start-group $MKLPATH/libmkl_intel_lp64.a
$MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a -Wl,--end-group
-liomp5 -lpthread
```
9. myprog.f、反復法スパースソルバーの並列バージョン、LP64 インターフェイスをサポートしているインテル® MKL の並列バージョンのスタティック・リンク。

1. Fortran 95 LAPACK および BLAS インターフェイス・ライブラリーの構築方法については、「[Fortran 95 インターフェイス、LAPACK および BLAS](#)」を参照してください。

- ```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -lmkl_solver_lp64
-Wl,--start-group $MKLPATH/libmkl_intel_lp64.a
$MKLPATH/libmkl_intel_thread.a $MKLPATH/libmkl_core.a -Wl,--end-group
-liomp5 -lpthread
```
10. myprog.f、反復法スパースソルバーの逐次バージョン、LP64 インターフェイスをサポートしているインテル® MKL の逐次バージョンのスタティック・リンク。
- ```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -lmkl_solver_lp64_sequential
-Wl,--start-group $MKLPATH/libmkl_intel_lp64
$MKLPATH/libmkl_sequential.a $MKLPATH/libmkl_core.a -Wl,--end-group
-lpthread
```

## カスタム共有オブジェクトの構築

カスタム共有オブジェクトを使用すると、特定の問題を解くために必要な関数のコレクションをインテル® MKL ライブラリーから減らすことができるため、ディスク容量が節約できます。また、独自のダイナミック・ライブラリーを構築できます。

## インテル® MKL カスタム共有オブジェクト・ビルダー

カスタム共有オブジェクト・ビルダーを使用すると、選択した関数を含むダイナミック・ライブラリー (共有オブジェクト) を作成できます。ビルダーは、tools/builder ディレクトリーにあります。ビルダーには、メイクファイルおよび定義ファイル (関数のリスト) が含まれています。メイクファイルには、ia32、ipf、および em64t の 3 つのターゲットがあります。ia32 は IA-32 アーキテクチャー対応プロセッサ、ipf は IA-64 アーキテクチャー対応プロセッサ、em64t はインテル® 64 アーキテクチャー対応プロセッサに使用します。



**注:** インテル® MKL スタティック・ライブラリーのオブジェクトは位置独立コード (PIC) であり、スタティック・ライブラリーでは一般的ではありません。カスタム共有オブジェクト・ビルダーは、スタティック・ライブラリーのオブジェクト・ファイルを使用して、インテル® MKL 関数のサブセットから共有オブジェクトを作成できます。

## メイクファイル・パラメーターの指定

メイクファイルには以下のパラメーター (マクロ) があります。

`export=user_list`

共有オブジェクトに含まれるエン트리ポイント関数のリストを含むファイルの完全な名前を指定します。デフォルト名は `user_list` (拡張子なし) です。

`name=mkl_custom`

作成するライブラリーの名前を指定します。デフォルトでは、作成するライブラリーの名前は `mkl_custom.so` です。

`xerbla=user_xerbla.o`

ユーザーのエラーハンドラーを含むオブジェクト・ファイルの名前を指定します。このエラーハンドラーは、ライブラリーに追加された後、デフォルトのインテル® MKL エラーハンドラー `xerbla` の代わりに使用されます。このパラメーターを省略すると、標準インテル® MKL エラーハンドラー `xerbla` が使用されます。独自のエラーハンドラーの開発方法は、『インテル® MKL リファレンス・マニュアル』の `xerbla` 関数の説明を参照してください。

`MKLROOT=<mkl ディレクトリー>`

カスタム共用オブジェクトの構築に使用するインテル® MKL ライブラリーの場所を指定します。デフォルトでは、現在のインテル® MKL リリースのインストール・ディレクトリーが使用されます。

すべてのパラメーターはオプションです。

最も単純なケースでのコマンドは `make ia32` になります。残りのパラメーターの値はデフォルトになります。この場合、IA-32 アーキテクチャー対応プロセッサ用の `mkl_custom.so` ライブラリー作成されます。`user_list` ファイルから関数のリストが取得され、標準インテル® MKL エラーハンドラー `xerbla` が使用されます。

以下は、より複雑な場合の例です。

```
make ia32 export=my_func_list.txt name=mkl_small xerbla=my_xerbla.o
```

この場合、IA-32 アーキテクチャー対応プロセッサ用の `mkl_small.so` ライブラリー作成されます。`my_func_list.txt` ファイルから関数のリストが取得され、ユーザーのエラーハンドラー `my_xerbla.o` が使用されます。

インテル® 64 または IA-64 アーキテクチャー対応プロセッサの場合も処理はほぼ同様です。

## 関数のリストの指定

インターフェイスに応じて `user_list` ファイルのエントリーポイントを調整します。例えば、Fortran 関数の場合、接尾辞として名前に下線文字 `"_"` を追加します。

```
dgemm_  
ddot_  
dgetrf_
```

選択した関数に複数のプロセッサ固有のバージョンがある場合、すべてのバージョンがカスタム・ライブラリーに追加され、ディスパッチャーによって管理されます。

# パフォーマンスとメモリーの 管理

## 6

本章は、インテル® マス・カーネル・ライブラリー (インテル® MKL) を使用して最適なパフォーマンスを得るための方法を説明します。最初に、スレッド化 (「[インテル® MKL 並列処理の使用](#)」を参照) について説明した後、ライブラリーのパフォーマンスを向上するためのコーディング・テクニックとハードウェア構成を示します。また、インテル® MKL のメモリー管理と、ライブラリーで使用するメモリー関数を再定義する方法についても説明します。

## インテル® MKL 並列処理の使用

インテル® MKL は、さまざまな場所でスレッド化されています。

- 直接法スパースソルバー。
- LAPACK
  - 線形方程式、計算ルーチン:
    - 因数分解: \*getrf、\*gbtrf、\*potrf、\*pptrf、\*sytrf、\*hetrf、\*sptrf、\*hptrf
    - 解法: \*gbtrs、\*gttrs、\*pptrs、\*pbtrs、\*pttrs、\*sytrs、\*spttrs、\*hptrs、\*tptrs、\*tbtrs
  - 直交因数分解、計算ルーチン:
    - \*geqrf、\*ormqr、\*unmqr、\*ormlq、\*unmlq、\*ormql、\*unmql、\*ormrq、\*unmrq
  - 特異値分解、計算ルーチン: \*gebrd、\*bdsqr
  - 対称固有値問題、計算ルーチン:
    - \*sytrd、\*hetrd、\*sptd、\*hptd、\*steqr、\*stedc

スレッド化された LAPACK または BLAS ルーチンをベースとする多くの LAPACK ルーチン (\*gesv、\*posv、\*gels、\*gesvd、\*syev、\*heev、その他) は、並列処理を効率的に利用します。

- レベル 1 およびレベル 2 BLAS 関数:
  - レベル 1 BLAS: \*axpy、\*copy、\*swap、ddot/sdot、drot/srot
  - レベル 2 BLAS: \*gemv、\*trmv、dsyr/ssyr、dsyr2/ssyr2、dsymv/ssymvこれらの関数は、以下の場合にのみスレッド化されます。
  - インテル® 64 アーキテクチャー
  - インテル® Core™2 Duo プロセッサおよびインテル® Core™ i7 プロセッサ
- すべてのレベル 3 BLAS およびすべてのスパース BLAS ルーチン (レベル 2 三角ソルバーを除く)。
- VML。
- FFT。

インテル® MKL はマルチスレッド・プログラミング用に設計されているため、スレッドセーフです。すべてのインテル® MKL 関数<sup>1</sup> は、複数のスレッドで同時に実行されている場合でも正常に動作します。データの共有部分へのアクセスが 1 つのスレッドに制限されている場合でも、スレッド化されたインテル® MKL のコードは、複数のスレッドから同じ共有データにアクセスできます。スレッドセーフなので、複数のスレッドからインテル® MKL を呼び出しても関数のインスタンスが干渉することを心配する必要はありません。

ライブラリーは、OpenMP\* スレッド化ソフトウェアを使用します。スレッド数は、OMP\_NUM\_THREADS 環境変数で設定します。スレッド数を設定する手法は複数あることに注意してください。インテル® MKL 10.0 よりも前のリリースでは、OMP\_NUM\_THREADS 環境変数 (詳細は、「[OpenMP 環境変数を使用したスレッド数の設定](#)」を参照) または等価な OpenMP ランタイム関数呼び出し (「[ランタイムのスレッド数の変更](#)」セクションを参照) を使用することができました。インテル® MKL バージョン 10.0 から、MKL\_NUM\_THREADS のような OpenMP とは独立した変数と等価なインテル® MKL 関数をスレッド管理に使用できるようになりました (詳細は、「[新しいスレッド化コントロールの使用](#)」を参照)。インテル® MKL 変数は常に最初に検査され、次に OpenMP 変数が検査されます。どちらの変数も使用されていない場合、OpenMP ソフトウェアはデフォルトのスレッド数を選択します。これは、インテル® MKL バージョン 9.x 以前からの変更点です。



---

**注:** インテル® MKL 10.0 以降は、OpenMP ソフトウェアがデフォルトのスレッド数を決定します。インテルの OpenMP ライブラリーでは、デフォルトのスレッド数はシステムの論理プロセッサの数と同じです。

---

より高いパフォーマンスを得るには、「[スレッド数を設定する手法](#)」で説明されているように、スレッド数を実際のプロセッサ数または物理コア数に設定してください。

以下の項目も参照してください。

[マルチコア・パフォーマンスの管理](#)

1. LAPACK の古いルーチン (zlacon、zlasq3、および zlasq4) を除きます。

## スレッド数を設定する手法

異なる手法を使用してインテル® MKL で使用するスレッド数を変更することができます。

- OpenMP またはインテル® MKL 環境変数のいずれかを設定します。
  - OMP\_NUM\_THREADS
  - MKL\_NUM\_THREADS
  - MKL\_DOMAIN\_NUM\_THREADS
- OpenMP またはインテル® MKL 関数のいずれかを呼び出します。
  - `omp_set_num_threads()`
  - `mkl_set_num_threads()`
  - `mkl_domain_set_num_threads()`

手法を選択する場合、以下の規則を考慮してください。

- インテル® MKL スレッド化コントロールは OpenMP 手法よりも優先されます。
- 関数呼び出しは環境変数よりも優先されます。ただし、OpenMP サブルーチン `omp_set_num_threads()` の呼び出しは例外で、インテル® MKL 環境変数 (MKL\_NUM\_THREADS など) が優先されます。
- 環境変数の読み取りはインテル® MKL を最初に呼び出したときに一度しか行われないため、ランタイム動作の変更には使用できません。

## 実行環境における競合の回避

特定の状況で、インテル® MKL でのスレッドの使用が問題となる競合が発生することがあります。このセクションでは最初に、なぜこの問題が存在するかについて簡単に説明します。

OpenMP 宣言子を使用してプログラムをスレッド化し、インテル® コンパイラーを使用してプログラムをコンパイルした場合、そのプログラムとインテル® MKL はどちらも同じスレッド化ライブラリーを使用します。インテル® MKL はプログラムに並列領域が存在するかどうかを判断し、存在する場合、ユーザーが MKL\_DYNAMIC で指定しない限り、その処理を複数のスレッド上では行いません (詳細は「[新しいスレッド化コントロールの使用](#)」を参照)。しかし、インテル® MKL が並列領域を認識できるのは、スレッド化プログラムとインテル® MKL が同じスレッド化を使用している場合のみです。ユーザーのプログラムがほかの手法でスレッド化されている場合、インテル® MKL はマルチスレッド・モードで動作しますが、リソースの浪費により適切なパフォーマンスが得られません。

使用するスレッド化モデル別に、推奨する競合の回避方法を説明します。

表 6-1 スレッド化モデル別の実行環境における競合の回避方法

| スレッド化モデル                                               | 説明                                                                                                                                           |
|--------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| OS スレッド (Linux® OS の場合 pthreads) を使用してプログラムをスレッド化する場合。 | 複数のスレッドがインテル® MKL を呼び出し、呼び出した関数がスレッド化されている場合、インテル® MKL のスレッド化をオフにする必要があります。利用可能な任意の手法を使用してスレッド数を設定します (「 <a href="#">スレッド数を設定する手法</a> 」を参照)。 |

表 6-1 スレッド化モデル別の実行環境における競合の回避方法 ( 続き )

| スレッド化モデル                                                                   | 説明                                                                                                                                                                                                                                                                                                                                                               |
|----------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OpenMP 宣言子またはプラグマを使用してプログラムをスレッド化し、インテル以外のコンパイラーを使用してプログラムをコンパイルする場合。      | 環境変数 OMP_NUM_THREADS の設定がコンパイラーのスレッド化ライブラリーと libiomp(libguide) の両方に影響を与えるため、問題になる可能性がより高くなります。この場合、インテル® MKL と使用する OpenMP コンパイラーのレイヤーが一致するように、スレッド化ライブラリーを選択してください ( 詳細は、「 <a href="#">リンクの例</a> 」を参照 )。選択できない場合、インテル® MKL を逐次モードで使用してください。この際、適切なスレッド化ライブラリー (libmkl_sequential.a または libmkl_sequential.so) をリンクする必要があります (「 <a href="#">上位ディレクトリー構造</a> 」を参照)。 |
| 例えば、マルチ CPU システムで、各プロセッサをノードとして扱い、コミュニケーションに MPI を使用する複数の並列プログラムが動作している場合。 | 各プロセッサで個別の MPI プロセスが実行されていても、スレッド化ソフトウェアはシステムに複数のプロセッサが存在することを認識します。この場合、解決方法の 1 つは、利用可能な任意の手法を使用してスレッド数を設定することです (「 <a href="#">スレッド数を設定する手法</a> 」を参照)。ハイブリッド (OpenMP + MPI) モードの場合の別の解決方法については、「 <a href="#">Intel® Optimized MP LINPACK Benchmark for Clusters</a> 」を参照してください。                                                                                |

以下の項目も参照してください。

[コンパイラー・サポート RTL のリンク](#)

## OpenMP 環境変数を使用したスレッド数の設定

OMP\_NUM\_THREADS 環境変数を使用してスレッド数を設定することができます。スレッド数を変更するには、プログラムを実行するコマンドシェルで次のように入力します。

export OMP\_NUM\_THREADS=< 使用するスレッド数 > (bash などの特定のシェルの場合)

または

set OMP\_NUM\_THREADS=< 使用するスレッド数 > (csh や tcsh などほかのシェルの場合)

インテル® MKL 環境変数 (例えば、MKL\_NUM\_THREADS) を使用してスレッド数を設定する方法は、「[新しいスレッド化コントロールの使用](#)」を参照してください。

## ランタイムのスレッド数の変更

環境変数を使用してランタイム時にスレッド数を変更することはできません。しかし、プログラムから OpenMP API 関数を呼び出してランタイム時にスレッドの数を変更することはできます。以下のサンプルコードでは、omp\_set\_num\_threads() ルーチンを使用してランタイム時にスレッドの数を変更しています。「[スレッド数を設定する手法](#)」も参照してください。

以下のサンプルは、C と Fortran コードの両方を示しています。C 言語でこのサンプルを実行するには、インテル® コンパイラー・パッケージの omp.h ヘッダーファイルを使用します。インテル® コンパイラーがない場合、C バージョンではなく omp\_set\_num\_threads() に Fortran API を使用してください。

### 例 6-1 スレッド数の変更

```
// ***** C 言語 *****
#include "omp.h"
#include "mkl.h"
#include <stdio.h>
```



## 例 6-1 スレッド数の変更 ( 続き )

```

#define SIZE 1000

void main(int args, char *argv[]){

    double *a, *b, *c;
    a = new double [SIZE*SIZE];
    b = new double [SIZE*SIZE];
    c = new double [SIZE*SIZE];

    double alpha=1, beta=1;
    int m=SIZE, n=SIZE, k=SIZE, lda=SIZE, ldb=SIZE, ldc=SIZE, i=0, j=0;
    char transa='n', transb='n';
    for( i=0; i<SIZE; i++){
        for( j=0; j<SIZE; j++){
            a[i*SIZE+j]= (double) (i+j);
            b[i*SIZE+j]= (double) (i*j);
            c[i*SIZE+j]= (double) 0;
        }
    }
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
                m, n, k, alpha, a, lda, b, ldb, beta, c, ldc);
    printf("row\ta\tc\n");
    for ( i=0;i<10;i++){
        printf("%d:\t%f\t%f\n", i, a[i*SIZE], c[i*SIZE]);
    }

    omp_set_num_threads(1);

    for( i=0; i<SIZE; i++){
        for( j=0; j<SIZE; j++){
            a[i*SIZE+j]= (double) (i+j);
            b[i*SIZE+j]= (double) (i*j);
            c[i*SIZE+j]= (double) 0;
        }
    }
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
                m, n, k, alpha, a, lda, b, ldb, beta, c, ldc);

    printf("row\ta\tc\n");
    for ( i=0;i<10;i++){
        printf("%d:\t%f\t%f\n", i, a[i*SIZE], c[i*SIZE]);
    }
    omp_set_num_threads(2);
    for( i=0; i<SIZE; i++){
        for( j=0; j<SIZE; j++){
            a[i*SIZE+j]= (double) (i+j);
            b[i*SIZE+j]= (double) (i*j);
            c[i*SIZE+j]= (double) 0;
        }
    }
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
                m, n, k, alpha, a, lda, b, ldb, beta, c, ldc);

    printf("row\ta\tc\n");
    for ( i=0;i<10;i++){
        printf("%d:\t%f\t%f\n", i, a[i*SIZE],
c[i*SIZE]);
    }

    delete [] a;
    delete [] b;
    delete [] c;
}

// ***** Fortran 言語 *****
PROGRAM DGEMM_DIFF_THREADS

```

**例 6-1 スレッド数の変更 ( 続き )**

---

```

INTEGER      N, I, J
PARAMETER    (N=1000)

REAL*8       A(N,N), B(N,N), C(N,N)
REAL*8       ALPHA, BETA

INTEGER*8     MKL_MALLOC
integer       ALLOC_SIZE

integer       NTHRS

ALLOC_SIZE = 8*N*N
A_PTR = MKL_MALLOC(ALLOC_SIZE,128)
B_PTR = MKL_MALLOC(ALLOC_SIZE,128)
C_PTR = MKL_MALLOC(ALLOC_SIZE,128)

ALPHA = 1.1
BETA = -1.2

DO I=1,N
DO J=1,N
    A(I,J) = I+J
    B(I,J) = I*j
    C(I,J) = 0.0
END DO
END DO

CALL DGEMM('N','N',N,N,N,ALPHA,A,N,B,N,BETA,C,N)

print *, 'Row          A          C'
DO i=1,10
    write(*, '(I4,F20.8,F20.8)') I, A(1,I), C(1,I)
END DO

CALL OMP_SET_NUM_THREADS(1);

DO I=1,N
DO J=1,N
    A(I,J) = I+J
    B(I,J) = I*j
    C(I,J) = 0.0
END DO
END DO

CALL DGEMM('N','N',N,N,N,ALPHA,A,N,B,N,BETA,C,N)

print *, 'Row          A          C'
DO i=1,10
    write(*, '(I4,F20.8,F20.8)') I, A(1,I), C(1,I)
END DO

CALL OMP_SET_NUM_THREADS(2);

DO I=1,N
DO J=1,N
    A(I,J) = I+J
    B(I,J) = I*j
    C(I,J) = 0.0
END DO
END DO

CALL DGEMM('N','N',N,N,N,ALPHA,A,N,B,N,BETA,C,N)

print *, 'Row          A          C'
DO i=1,10
    write(*, '(I4,F20.8,F20.8)') I, A(1,I), C(1,I)
END DO

STOP

```

---

## 例 6-1 スレッド数の変更 (続き)

END

## 新しいスレッド化コントロールの使用

インテル® MKL に、新しいオプションのスレッド化コントロール (新しい環境変数とサービス関数) が追加されました。これらのコントロールは OpenMP の等価な変数と似ていますが、OpenMP 変数よりも優先されます。これらのコントロールを OpenMP 変数とともに使用することで、インテル® MKL とライブラリーを互いに呼び出さないアプリケーションの部分をスレッド化することができます。

これらのコントロールを使用すると、OpenMP 設定とは無関係にインテル® MKL のスレッド数を指定できます。インテル® MKL は実際には推奨と異なるスレッド数を使用することがありますが、コントロールは呼び出しアプリケーションで使用されている数が利用できない場合、推奨するスレッド数を使用するようにライブラリーに指示します。



**注:** インテル® MKL では、システムリソースなどの特定の理由により、スレッド数を常に変更できるとは限りません。

アプリケーションでインテル® MKL スレッド化コントロールを使用するかどうかは任意です。コントロールを使用しない場合、ライブラリーは、デフォルトのスレッド数が異なることを除けば、スレッド化に関してはインテル® MKL 9.1 と同じように動作します。

『インテル® MKL リファレンス・マニュアル』の「Fourier Transform Functions」の章の「Number of User Threads」セクションに、インテル® MKL スレッド化コントロールを使用して FFT 計算のスレッド数を設定する方法が説明されています。

表 6-2 は、スレッド化コントロール用のインテル® MKL 環境変数とサービス関数、および等価な OMP 環境変数の一覧です。

表 6-2 スレッド化コントロール用の環境変数

| 環境変数                   | サービス関数                     | 内容                               | 等価な OpenMP 環境変数 |
|------------------------|----------------------------|----------------------------------|-----------------|
| MKL_NUM_THREADS        | mkl_set_num_threads        | 使用するスレッド数を示します。                  | OMP_NUM_THREADS |
| MKL_DOMAIN_NUM_THREADS | mkl_domain_set_num_threads | 特定の関数ドメイン用のスレッド数を示します。           |                 |
| MKL_DYNAMIC            | mkl_set_dynamic            | インテル® MKL がスレッド数を動的に変更できるようにします。 | OMP_DYNAMIC     |



**注:** 関数はそれぞれの環境変数よりも優先されます。特に、アプリケーションで、インテル® MKL が指定されたスレッド数を使用し、アプリケーションのユーザーが環境変数を使用してスレッド数を変更しないようにするには、`mkl_set_num_threads()` を呼び出してスレッド数を設定します。この設定は、環境変数の設定よりも優先されます。

次の例は、インテル® MKL 関数 `mkl_set_num_threads()` を使用してインテル® MKL 9.x のデフォルトの動作 (1 つのスレッドで実行) を行う方法を示しています。

#### 例 6-2 スレッド数を 1 に設定

```
// ***** C 言語 *****
#include <omp.h>
#include <mkl.h>

...

mkl_set_num_threads ( 1 );

// ***** Fortran 言語 *****
...

call mkl_set_num_threads( 1 )
```

この後のセクションでは、スレッド化コントロール用のインテル® MKL 環境変数について説明します。スレッド化コントロール関数、パラメーター、呼び出し構文、コードの例は、『インテル® MKL リファレンス・マニュアル』を参照してください。

### MKL\_DYNAMIC

MKL\_DYNAMIC のデフォルト値は、OMP\_DYNAMIC のデフォルト値が FALSE の場合でも、TRUE です。

MKL\_DYNAMIC が TRUE の場合、インテル® MKL は常に、ユーザーが指定した最大値までの範囲で、最良とみなすスレッド数を選択します。

MKL\_DYNAMIC が FALSE の場合、インテル® MKL は、ユーザーが指定したスレッド数を使用します。ただし、MKL\_DYNAMIC=FALSE を設定しても、例えば、システムリソースの不足など、要求されたスレッド数を選択できない場合、インテル® MKL はそのスレッド数を使用しません。さらに、ライブラリーは問題を調査し、推奨値と異なるスレッド数を使用することがあります。例えば、8 つのスレッドでサイズ 1 の行列 - 行列乗算を行おうとすると、このイベントで 8 つのスレッドを使用することは実用的でないため、ライブラリーは代わりに 1 スレッドのみ使用します。

インテル® MKL が並列領域で呼び出された場合、デフォルトでは 1 つのスレッドのみを使用することにも注意してください。ライブラリーで入れ子の並列処理を使用し、並列領域内のスレッドをインテル® MKL が使用しているものと同じ OpenMP コンパイラーでコンパイルする場合、MKL\_DYNAMIC を FALSE にして、スレッド数を手動で設定してください。

一般に、ライブラリーが並列セクションからすでに呼び出されていて入れ子の並列処理が望ましい場合など、インテル® MKL が検出できない状況でのみ、MKL\_DYNAMIC を FALSE に設定してください。

MKL\_DYNAMIC が TRUE の場合、特に以下のようなケースでは最適なスレッド数が選択されます。

- 要求されたスレッド数が (ハイパースレッディングなどにより) 物理コアの数を超えていて、MKL\_DYNAMIC がデフォルト値の TRUE から変更されていない場合、インテル® MKL はスレッド数を物理コアの数まで下げます。
- MPI が使用されていて、スレッドセーフ・モードで呼び出されているかどうか判断できない場合 (例えば、MPICH 1.2.x では検出不可) で、MKL\_DYNAMIC がデフォルト値の TRUE から変更されていない場合、インテル® MKL は 1 つのスレッドを実行します。

### MKL\_DOMAIN\_NUM\_THREADS

MKL\_DOMAIN\_NUM\_THREADS には、文字列値 `<MKL 環境文字列>` を以下の形式で指定します。

```
<MKL 環境文字列> ::= <MKL ドメイン環境文字列> { <区切り文字>
    <MKL ドメイン環境文字列> }
<区切り文字> ::= [ <スペース記号>* ] ( <スペース記号> | <カンマ記号> |
    <セミコロン記号> | <コロン記号> ) [ <スペース記号>* ]
```

<MKL ドメイン環境文字列> ::= <MKL ドメイン環境名> <使用法> <スレッド数>  
 <MKL ドメイン環境名> ::= MKL\_ALL | MKL\_BLAS | MKL\_FFT | MKL\_VML  
 <使用法> ::= [ <スペース記号>\* ] ( <スペース記号> | <等号記号> | <カンマ記号> ) [ <スペース記号>\* ]  
 <スレッド数> ::= <正の整数>  
 <正の整数> ::= <10 進数> | <8 進数> | <16 進数>

上記の構文で、MKL\_BLAS は BLAS 関数ドメイン、MKL\_FFT は非クラスター FFT、MKL\_VML はベクトル・マス・ライブラリーを示します。

例:

```
MKL_ALL 2 : MKL_BLAS 1 : MKL_FFT 4
MKL_ALL=2 : MKL_BLAS=1 : MKL_FFT=4
MKL_ALL=2, MKL_BLAS=1, MKL_FFT=4
MKL_ALL=2; MKL_BLAS=1; MKL_FFT=4
MKL_ALL = 2 MKL_BLAS 1 , MKL_FFT 4
MKL_ALL,2: MKL_BLAS 1, MKL_FFT,4 .
```

グローバル変数 MKL\_ALL、MKL\_BLAS、MKL\_FFT、MKL\_VML、およびインテル® MKL スレッド化コントロール用のインターフェイスは、mkl.h ヘッダーファイルに記述されています。

[表 6-3](#) は、MKL\_DOMAIN\_NUM\_THREADS の値がどのように解釈されるかを示しています。

**表 6-3 MKL\_DOMAIN\_NUM\_THREADS の値の解釈**

| MKL_DOMAIN_NUM_THREADS の値 | 解釈                                                                                                                       |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------|
| MKL_ALL=4                 | インテル® MKL のすべての部分で 4 つのスレッドを使用するように推奨します。実際のスレッド数は、MKL_DYNAMIC の設定やシステムリソースの状況に応じて異なります。この設定は、MKL_NUM_THREADS = 4 と等価です。 |
| MKL_ALL=1, MKL_BLAS=4     | BLAS で 4 つのスレッドを使用することを除いて、インテル® MKL の残りの部分で 1 つのスレッドを使用するように推奨します。                                                      |
| MKL_VML = 2               | VML で 2 つのスレッドを使用するように推奨します。設定は、インテル® MKL のほかの部分には影響しません。                                                                |



**注:** ドメイン固有の設定は、ほかの設定よりも優先されます。例えば、MKL\_DOMAIN\_NUM\_THREADS が "MKL\_BLAS=4" に設定された場合、MKL\_NUM\_THREADS の設定に関係なく、BLAS で 4 つのスレッドを使用するように推奨します。関数呼び出し "mkl\_domain\_set\_num\_threads ( 4, MKL\_BLAS );" も、mkl\_set\_num\_threads() への呼び出しに関係なく、BLAS で 4 つのスレッドを使用するように推奨します。しかし、"mkl\_domain\_set\_num\_threads (4, MKL\_ALL);" のように、"MKL\_ALL" が入力関数呼び出しは、"mkl\_set\_num\_threads(4)" と等価であるため、後の mkl\_set\_num\_threads 呼び出しよりも優先されることに注意してください。同様に、MKL\_DOMAIN\_NUM\_THREADS が "MKL\_ALL=4" に設定された場合、MKL\_NUM\_THREADS = 2 よりも優先されます。

例えば、MKL\_DOMAIN\_NUM\_THREADS 環境変数では、"MKL\_BLAS=4,MKL\_FFT=2" のように複数の変数を一度に設定することができますが、対応する関数で文字列構文は使用できません。このため、関数呼び出しで同じことを行うには、以下のように複数の呼び出しを行う必要があります。

```
mkl_domain_set_num_threads ( 4, MKL_BLAS );
```

```
mkldomain_set_num_threads ( 2, MKL_FFT );
```

### スレッド化コントロール用の環境変数の設定

スレッド化コントロールに使用する環境変数を設定するには、プログラムを実行するコマンドシェルで、使用しているシェルに応じて、`export` または `set` コマンドを入力します。例えば、`bash` やほかのシェルでは、`export` コマンドを入力します。

```
export <変数名>=<値>
```

次に例を示します。

```
export MKL_NUM_THREADS=4
```

```
export MKL_DOMAIN_NUM_THREADS="MKL_ALL=1, MKL_BLAS=4"
```

```
export MKL_DYNAMIC=FALSE
```

`csh` や `tcsh` などのほかのシェルの場合は、`set` コマンドを使用します。

```
set <変数名>=<値>
```

次に例を示します。

```
set MKL_NUM_THREADS=4
```

```
set MKL_DOMAIN_NUM_THREADS="MKL_ALL=1, MKL_BLAS=4"
```

```
set MKL_DYNAMIC=FALSE
```

## インテル® Advanced Vector Extensions (インテル® AVX) のディスパッチ

インテル® MKL 10.2 では、インテル® AVX 用に最適化されたカーネルが追加されました。インテル® AVX が有効なハードウェア (またはシミュレーション) でインテル® AVX 命令をディスパッチするには、インテル® MKL サービス関数 `mkld_enable_instructions()` を使用します。この関数を使用すると、新しいインテル® AVX 命令のディスパッチが有効になります。すべてのインテル® MKL 関数呼び出しの前に、この関数を呼び出してください。関数の説明は、『インテル® MKL リファレンス・マニュアル』を参照してください。



**注:** この関数が正常に実行されても、新しい命令のディスパッチは保証されません。特定の命令のディスパッチは、ハードウェアでインテル® AVX が有効かどうか、および関数がこの命令をディスパッチするためにすでに最適化されているかどうかによって依存します。この関数を呼び出さない場合、新しい命令はディスパッチされません。

インテル® AVX 命令セットは現在も改良されているため、`mkld_enable_instructions()` の動作は将来のインテル® MKL リリースで変更される可能性があります。関数の動作についてのリリース固有の情報は、『リリースノート』を参照してください。

## パフォーマンスを向上するためのヒントと手法

インテル® MKL を使用して最適なパフォーマンスを得るには、次に説明する推奨事項に従ってください。

## コーディング手法

インテル® MKL を使用して最適なパフォーマンスを得るため、ソースコードでデータが以下のよう  
にアライメントされていることを確認してください。

- ・ 配列が 16 バイト境界でアライメントされている。
- ・ 2 次元配列のリーディング・ディメンジョンの値 ( $n \times \text{element\_size}$ ) が 16 の倍数である。
- ・ 2 次元配列のリーディング・ディメンジョンの値が 2048 の倍数ではない。

### LAPACK 圧縮ルーチン

名前の行列の型と格納位置 (2 つめと 3 つめの文字) が HP、OP、PP、SP、TP、UP のルーチンは、  
圧縮形式で行列を処理します (『インテル® MKL リファレンス・マニュアル』の LAPACK の「ルーチ  
ン命名規則」セクションを参照)。これらの機能は、名前の行列の型と格納位置 (2 つめと 3 つめの  
文字) が HE、OR、PO、SY、TR、UN の非圧縮ルーチンの機能と厳密に等価ですが、パフォーマン  
スは大幅に低くなります。

メモリー制限があまり厳しくない場合は、非圧縮ルーチンを使用してください。この場合、それ  
ぞれの圧縮ルーチンで要求されるメモリーよりも  $N^2/2$  多いメモリーを割り当てる必要がありま  
す。N は問題サイズ (方程式の数) です。

例えば、エキスパート・ドライバーを使用した対称固有値問題を解く時間を短縮するには、次の  
非圧縮ルーチンを使用します。

```
call dsyevx(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol, m, w, z,
ldz, work, lwork, iwork, ifail, info)
```

a は少なくとも  $N^2$  の要素を含む次元  $lda \times n$  です。変更前は次のとおりです。

```
call dspevx(jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m, w, z,
ldz, work, iwork, ifail, info)
```

ap は次元  $N \times (N+1)/2$  です。

### FFT 関数

追加の条件により FFT 関数のパフォーマンスを向上できます。

**IA-32 またはインテル® 64 アーキテクチャー・ベースのアプリケーション:** 2 次元配列の  
最初の要素のアドレスとリーディング・ディメンジョンの値 ( $n \times \text{element\_size}$ ) が、以下の  
キャッシュ・ライン・サイズの倍数である必要があります。

- ・ 32 バイト (インテル® Pentium® III プロセッサの場合)
- ・ 64 バイト (インテル® Pentium® 4 プロセッサおよびインテル® 64 アーキテクチャー対応プロ  
セッサ)

**IA-64 アーキテクチャー・ベースのアプリケーション:** 2 次元配列のリーディング・ディメ  
ンジョンの値 ( $n \times \text{element\_size}$ ) が 2 の累乗ではない。

## ハードウェア構成のヒント

**デュアルコア インテル® Xeon® プロセッサ 5100 番台のシステム:** デュアルコア インテ  
ル® Xeon® プロセッサ 5100 番台のシステムで最良のインテル® MKL パフォーマンスを得るには、  
このプロセッサのハードウェア DPL (ストリーミング・データ) プリフェッチャー機能を有効に  
してください。この機能を有効にするには、適切な BIOS 設定を使用します。詳細は、BIOS のド  
キュメントを参照してください。

**ハイパースレッディング・テクノロジーの使用:** ハイパースレッディング・テクノロジー (HT  
テクノロジー) は、各スレッドが異なる演算を実行している場合、またはプロセッサ上に十分に  
活用されていないリソースがある場合に特に有効です。しかし、インテル® MKL は、このどちらに

もあてはまりません。ライブラリーのスレッド化された部分が利用可能なリソースの大半を使用して効率的に実行され、各スレッドで同一の演算を行っているためです。HT テクノロジーを無効にすると、より高いパフォーマンスを得られることがあります。デフォルトのスレッド数、スレッド数の変更、その他の関連情報は、「[インテル® MKL 並列処理の使用](#)」を参照してください。HT テクノロジーを有効にして実行する場合、物理コアの数よりも少ないスレッド数で実行すると、特にパフォーマンスに影響があります。また、例えば、すべての物理コアに対して2つのスレッドがある場合、スレッド・スケジューラーは一部のコアに2つのスレッドを割り当て、ほかのコアを無視することがあります。インテル® コンパイラーの OpenMP ライブラリーを使用している場合、この状況を回避するために、ユーザーガイドを参照して、スレッド・アフィニティー・インターフェイスの最適な設定を行ってください。インテル® MKL では、`KMP_AFFINITY=granularity=fine,compact,1,0` に設定することを推奨します。

## マルチコア・パフォーマンスの管理

スレッドを処理するコアが変更されないようにすることで、マルチコア・プロセッサのシステムで最良のパフォーマンスを得ることができます。このためには、スレッドにアフィニティー・マスクを設定し、スレッドと CPU コアをバインドします。以下のいずれかのオプションを使用してください。

- OpenMP の機能 ( 利用可能な場合、例えば、インテルの OpenMP ライブラリーを使用して `KMP_AFFINITY` 環境変数を設定することを推奨 )
- システム関数 ( この後のサンプルを参照 )

次のケースについて考えてみます。

- システムにそれぞれ2つのコアがある2つのソケットがある ( 合計 4 コア )。
- インテル® MKL FFT を呼び出す 2 スレッドの並列アプリケーションが 4 スレッドでは高速に実行されるが、2 スレッドのパフォーマンスが非常に不安定

この問題を解決する C のコードサンプルを [例 6-3](#) に示します。このコードサンプルは、システム関数 `sched_setaffinity` を呼び出して異なるソケットのコアにスレッドをバインドします。その後、インテル® MKL FFT 関数が呼び出されます。

インテル® コンパイラーで、以下のコマンドを使用してアプリケーションをコンパイルします。

```
icc test_application.c -openmp
```

`test_application.c` は、アプリケーションのファイル名です。

アプリケーションをビルドします。例えば、次のように環境変数を使用してスレッド数を設定し、ビルドしたアプリケーションを 2 スレッドで実行します。

```
env OMP_NUM_THREADS=2 ./a.out
```

### 例 6-3 インテル® コンパイラーを使用してオペレーティング・システムでアフィニティー・マスクを設定

```
#include <stdio.h>

#define __USE_GNU
// アフィニティー・マスクを設定します。
#include <sched.h>
#include <unistd.h>
#include <omp.h>

int main(void) {
    int NCPUs = sysconf(_SC_NPROCESSORS_CONF);

    printf("%i NCPUs のスレッド・アフィニティーを使用 \n", NCPUs);

#pragma omp parallel default(shared)
    {
        cpu_set_t new_mask;
        cpu_set_t was_mask;
```



**例 6-3 インテル® コンパイラーを使用してオペレーティング・システムでアフィニティー・マスクを設定 (続き)**

```

int tid = omp_get_thread_num();

CPU_ZERO(&new_mask);

// 2 パッケージ x 2 コア / パッケージ x 1 スレッド / コア (4 コア)
CPU_SET(tid==0 ? 0 : 2, &new_mask);

if (sched_getaffinity(0, sizeof(was_mask), &was_mask) == -1) {
    printf("エラー: sched_getaffinity(%d, sizeof(was_mask), &was_mask)\n", tid);
}
if (sched_setaffinity(0, sizeof(new_mask), &new_mask) == -1) {
    printf("エラー: sched_setaffinity(%d, sizeof(new_mask), &new_mask)\n", tid);
}
printf("tid=%d new_mask=%08X was_mask=%08X\n", tid,
        *(unsigned int*)(&new_mask), *(unsigned int*)(&was_mask));
}

// インテル® MKL FFT ルーチンを呼び出します。

return 0;
}

```

上記の例で使用されている `sched_setaffinity` `SetThreadAffinityMask` 関数の詳細については、*Linux プログラマーズ・マニュアル* (man ページ形式) を参照してください。

## 非正規化数の演算

IEEE 754-2008 規格「An IEEE Standard for Binary Floating-Point Arithmetic」では、浮動小数点形式で最も小さな正規化数よりも小さな非ゼロの数を、*denormal* (または *subnormal*) 数として定義しています。本書では、どちらも非正規化数として訳しています。非正規化オペランドと結果は、通常ハードウェアで直接処理されるのではなく、ソフトウェアで処理されるため、非正規化数の浮動小数点演算は正規化オペランドよりも遅くなります。このソフトウェア処理により、インテル® MKL 関数で非正規化数を使用する場合、正規化された浮動小数点数を使用する場合よりも実行が遅くなります。

このパフォーマンス問題に対応する方法として、MXCSR 浮動小数点制御レジスターの適切なビットフィールドを設定する方法があります。FTZ は非正規化数をゼロにフラッシュし、DAZ はメモリーからロードされた非正規化数をゼロに置換します。コンパイラーで FTZ および DAZ を制御するオプションが用意されているかどうかは、コンパイラーのドキュメントを参照してください。これらのコンパイラー・オプションを使用すると精度に影響する可能性があることに注意してください。

## FFT 最適化基数

データベクトルの長さが最適化基数の累乗に因数分解できる場合、インテル® MKL FFT のパフォーマンスを向上させることができます。

インテル® MKL では、最適化基数は 2、3、5、7、および 11 です。

## インテル® MKL メモリー管理の使用

インテル® MKL には、ライブラリー関数によって使用されるメモリーバッファを管理するメモリー管理ソフトウェアが用意されています。アプリケーションが特定の関数 (レベル 3 BLAS または FFT) を呼び出すときにライブラリーが割り当てる新しいバッファは、プログラムを終了するまで解放されません。メモリー管理ソフトウェアによって割り当てられたメモリー容量を取得するには、`mk1_mem_stat()` 関数を呼び出します。プログラムでメモリーを解放する必要がある場

合は、`mkl_free_buffers()` 関数を呼び出します。メモリーバッファが必要なライブラリー関数に別の呼び出しが行われると、メモリー・マネージャーはバッファを再び割り当てます。このバッファは、プログラムを終了するか、プログラムがメモリーを解放するまで割り当てられたままです。

この動作により、パフォーマンスが向上します。しかし、一部のツールではこの動作をメモリーリークとして報告することがあります。`mkl_free_buffers()` 関数を呼び出す以外に、環境変数を設定してプログラムのメモリーを解放することもできます。

メモリー管理ソフトウェアはデフォルトでオンになっているため、プログラムを終了するまでレベル 3 BLAS と FFT を呼び出して割り当てられたメモリーは解放されません。メモリー管理ソフトウェアの動作を無効にするには、`MKL_DISABLE_FAST_MM` 環境変数に任意の値を設定してください。メモリーは呼び出しごとに割り当てられ、呼び出しの後に解放されます。この機能を無効にすると、特に問題サイズの小さな、レベル 3 BLAS などのルーチンのパフォーマンスが低下します。

しかし、これらの方法を使用してメモリーを解放しても、メモリーリークが報告されなくなるとは限りません。実際、ライブラリーを複数回呼び出す場合、各呼び出しごとに新しいメモリーの割り当てが必要になり、報告される数は増えることもあります。上記の方法で解放されなかったメモリーは、プログラムの終了時にシステムによって解放されます。

## メモリー関数の再定義

インテル® MKL の C/C++ ユーザーは、ライブラリーがデフォルトで使用するメモリー関数を独自の関数に置換できます。置換を行うには、*メモリー関数名の変更機能*を使用します。

### メモリー関数名の変更

デフォルトでは、インテル® MKL メモリー管理は、標準 C ランタイムメモリー関数を使用してメモリーの割り当てと解放を行います。これらの関数は、メモリー関数名の変更機能を使用して置換することができます。

インテル® MKL は、アプリケーション・レベルで確認可能な、ポインター `i_malloc`、`i_free`、`i_calloc`、`i_realloc` を使用してメモリー関数にアクセスしています。これらのポインターは、最初は標準 C ランタイムメモリー関数 `malloc`、`free`、`calloc`、`realloc` のアドレスをそれぞれ保持しています。これらのポインターの値がアプリケーションのメモリー管理関数のアドレスを指すように、プログラムで再定義できます。

ポインターのリダイレクトは、独自のメモリー管理関数を使用するための唯一の正しい方法です。ポインターのリダイレクトを行わずに独自のメモリー関数を呼び出すと、2 つのメモリー管理パッケージでメモリーが管理されることになり、予測できないメモリー問題が発生します。

### メモリー関数の再定義方法

メモリー関数を再定義するには、以下の操作を行います。

1. `i_malloc.h` ヘッダーファイルをコードにインクルードします。  
ヘッダーファイルには、アプリケーション開発者がメモリー割り当て関数を置換するために必要な宣言がすべて含まれています。このヘッダーファイルでは、この機能をサポートするインテル® ライブラリーでメモリー割り当てを置換する方法も示しています。
2. インテル® MKL 関数への最初の呼び出しの前に、ポインター `i_malloc`、`i_free`、`i_calloc`、`i_realloc` の値を再定義します。

**例 6-4   メモリー関数の再定義**

---

```
#include "i_malloc.h"

. . .
i_malloc  = my_malloc;
i_calloc  = my_calloc;
i_realloc = my_realloc;
i_free    = my_free;
. . .
// ここでインテル® MKL 関数を呼び出すことができます。
```

---

# 言語固有の使用法オプション

## 7

インテル® マス・カーネル・ライブラリー (インテル® MKL) は、Fortran と C/C++ プログラミングを  
広範囲にサポートしています。しかし、すべての関数ドメインで Fortran と C インターフェイスの  
両方をサポートしているとは限りません (付録 A の「表 A-1」を参照)。例えば、LAPACK には C イン  
ターフェイスはありません。混在言語プログラミングを使用して、C からこれらのドメインを含  
む関数を呼び出すことは可能です。

Fortran 95 環境で、Fortran をサポートする LAPACK や BLAS を使用する場合でも、最初に、インテ  
ル® MKL で提供されているソースコードからコンパイラ固有のインターフェイス・ライブラリー  
とモジュールをビルドする作業が必要です。

本章は、主に混在言語プログラミングと言語固有のインターフェイスの使用について説明します。  
Fortran インターフェイスのみが提供されている関数ドメインで C 言語環境のインテル® MKL を使  
用する方法と、言語固有のインターフェイス、特に Fortran 95 インターフェイスを LAPACK および  
BLAS で使用する方法を説明します。また、コンパイラ依存の関数についても説明し、  
Fortran 90 のモジュールがソースで提供されている理由について明らかにします。別のセクション  
では、Java® からインテル® MKL 関数を呼び出すサンプルを実行する過程を、順を追って説明しま  
す。

## 言語固有インターフェイスとインテル® MKL の使用

interfaces ディレクトリーにあるメイクファイルを使用して、以下のインターフェイス・ライブラ  
リーとモジュールを生成できます。

表 7-1 インターフェイス・ライブラリーとモジュール

| ファイル名                                     | 内容                                                                              |
|-------------------------------------------|---------------------------------------------------------------------------------|
| ライブラリー (インテル® MKL のアーキテクチャー固有のディレクトリーに生成) |                                                                                 |
| libmkl_blas95.a <sup>1</sup>              | IA-32 アーキテクチャー用の BLAS (BLAS95) 用 Fortran 95 ラッパー。                               |
| libmkl_blas95_ilp64.a <sup>1</sup>        | LP64 インターフェイスをサポートする BLAS (BLAS95) 用 Fortran 95 ラッパー。                           |
| libmkl_blas95_lp64.a <sup>1</sup>         | ILP64 インターフェイスをサポートする BLAS (BLAS95) 用 Fortran 95 ラッパー。                          |
| libmkl_lapack95.a <sup>1</sup>            | IA-32 アーキテクチャー用の LAPACK (LAPACK95) 用 Fortran 95 ラッパー。                           |
| libmkl_lapack95_lp64.a <sup>1</sup>       | LP64 インターフェイスをサポートする LAPACK (LAPACK95) 用 Fortran 95 ラッパー。                       |
| libmkl_lapack95_ilp64.a <sup>1</sup>      | ILP64 インターフェイスをサポートする LAPACK (LAPACK95) 用 Fortran 95 ラッパー。                      |
| libfftw2xc_intel.a <sup>1</sup>           | インテル® MKL FFT を呼び出す FFTW バージョン 2.x 用イン<br>ターフェイス (インテル® コンパイラ用 C インターフェイ<br>ス)。 |
| libfftw2xc_gnu.a                          | インテル® MKL FFT を呼び出す FFTW バージョン 2.x 用イン<br>ターフェイス (GNU® コンパイラ用 C インターフェイス)。      |

表 7-1 インターフェイス・ライブラリーとモジュール ( 続き )

| ファイル名                                                                        | 内容                                                                               |
|------------------------------------------------------------------------------|----------------------------------------------------------------------------------|
| libfftw2xf_intel.a                                                           | インテル® MKL FFT を呼び出す FFTW バージョン 2.x 用インターフェイス ( インテル® コンパイラー用 Fortran インターフェイス )。 |
| libfftw2xf_gnu.a                                                             | インテル® MKL FFT を呼び出す FFTW バージョン 2.x 用インターフェイス (GNU コンパイラー用 Fortran インターフェイス)。     |
| libfftw3xc_intel.a <sup>2</sup>                                              | インテル® MKL FFT を呼び出す FFTW バージョン 3.x 用インターフェイス ( インテル® コンパイラー用 C インターフェイス )。       |
| libfftw3xc_gnu.a                                                             | インテル® MKL FFT を呼び出す FFTW バージョン 3.x 用インターフェイス (GNU コンパイラー用 C インターフェイス)。           |
| libfftw3xf_intel.a <sup>2</sup>                                              | インテル® MKL FFT を呼び出す FFTW バージョン 3.x 用インターフェイス ( インテル® コンパイラー用 Fortran インターフェイス )。 |
| libfftw3xf_gnu.a                                                             | インテル® MKL FFT を呼び出す FFTW バージョン 3.x 用インターフェイス (GNU コンパイラー用 Fortran インターフェイス)。     |
| libfftw2x_cdf_t_SINGLE.a                                                     | インテル® MKL クラスター FFT を呼び出す MPI FFTW バージョン 2.x 用単精度インターフェイス (C インターフェイス)。          |
| libfftw2x_cdf_t_DOUBLE.a                                                     | インテル® MKL クラスター FFT を呼び出す MPI FFTW バージョン 2.x 用倍精度インターフェイス (C インターフェイス)。          |
| <b>モジュール ( インテル® MKL インクルード・ディレクトリーのアーキテクチャー / インターフェイス固有のサブディレクトリーに生成 )</b> |                                                                                  |
| blas95.mod <sup>1</sup>                                                      | BLAS (BLAS95) 用 Fortran 95 インターフェイス・モジュール。                                       |
| lapack95.mod <sup>1</sup>                                                    | LAPACK (LAPACK95) 用 Fortran 95 インターフェイス・モジュール。                                   |
| f95_precision.mod <sup>1</sup>                                               | BLAS95 および LAPACK95 用精度パラメーターの Fortran 95 定義。                                    |
| mk195_blas.mod <sup>1</sup>                                                  | BLAS (BLAS95) 用 Fortran 95 インターフェイス・モジュール、blas95.mod と同一。将来のリリースで削除予定。           |
| mk195_lapack.mod <sup>1</sup>                                                | LAPACK (LAPACK95) 用 Fortran 95 インターフェイス・モジュール、lapack95.mod と同一。将来のリリースで削除予定。     |
| mk195_precision.mod <sup>1</sup>                                             | BLAS95 および LAPACK95 用精度パラメーターの Fortran 95 定義。f95_precision.mod と同一。将来のリリースで削除予定。 |

1. インテル® Fortran コンパイラー用に事前構築

2. FFTW3 インターフェイスはインテル® MKL と統合されます。ビルド方法を定義するオプションおよびラッパーとスタンダード・ライブラリーの配置場所は、<mk1 ディレクトリー>/interfaces/fftw3x\*/makefile を参照してください。

ライブラリーとモジュールの生成方法の例については、この後の「[Fortran 95 インターフェイス、LAPACK および BLAS](#)」を参照してください。

FFTW ラッパーの詳細は、『インテル® MKL リファレンス・マニュアル』の付録 G を参照してください。

## Fortran 95 インターフェイス、LAPACK および BLAS

Fortran 95 インターフェイスはコンパイラーに依存します。インテル® MKL には、インテル® Fortran コンパイラーを使用して事前に構築されたインターフェイス・ライブラリーとモジュールが用意されています。さらに、Fortran 95 インターフェイスとラッパーがソースで提供されています。( 詳細は、「[コンパイラー依存の関数と Fortran 90 モジュール](#)」を参照 )。異なるコンパイラーを使用している場合、そのコンパイラーを使用して適切なライブラリーとモジュールをビルドし、ライブラリーをユーザーのライブラリーとしてリンクしてください。

1. `<mkl ディレクトリー>/interfaces/blas95` または `<mkl ディレクトリー>/interfaces/lapack95` ディレクトリーに移動します。
2. 以下のいずれかのコマンドを入力します。

|                                                                                    |                      |
|------------------------------------------------------------------------------------|----------------------|
| <code>make lib32 INSTALL_DIR=&lt;user_dir&gt;</code>                               | IA-32 アーキテクチャーの場合    |
| <code>make libem64t [interface=lp64 ilp64]<br/>INSTALL_DIR=&lt;user_dir&gt;</code> | インテル® 64 アーキテクチャーの場合 |
| <code>make lib64 [interface=lp64 ilp64]<br/>INSTALL_DIR=&lt;user_dir&gt;</code>    | IA-64 アーキテクチャーの場合    |



**注:** パラメーター `INSTALL_DIR` は必須です。

コマンドを実行すると、ライブラリーとモジュールがビルドされ、ライブラリーは `<user_dir>/lib/<arch>` ディレクトリーに、`.mod` ファイルは `<user_dir>/include/<arch>[/({lp64 | ilp64})]` ディレクトリーにインストールされます。`<arch>` は、`{32, em64t, 64}` のいずれかです。

デフォルトでは、`ifort` コンパイラーが選択されています。`make` の `FC=<コンパイラー>` パラメーターを使用してコンパイラーのコマンド名を変更することもできます。

例えば、次のコマンドを実行すると、ライブラリーとモジュールがビルドされ、`<user_pgf95_dir>` のサブディレクトリーにインストールされます。

```
make libem64t FC=pgf95 INSTALL_DIR=<user_pgf95_dir> interface=lp64
```

ライブラリーをビルドせずにインターフェイスを使用する別の方法もあります。

ビルド用ディレクトリーからライブラリーを削除するには、以下のコマンドを使用します。

|                                                           |                      |
|-----------------------------------------------------------|----------------------|
| <code>make clean32 INSTALL_DIR=&lt;user_dir&gt;</code>    | IA-32 アーキテクチャーの場合    |
| <code>make cleanem64t INSTALL_DIR=&lt;user_dir&gt;</code> | インテル® 64 アーキテクチャーの場合 |
| <code>make clean64 INSTALL_DIR=&lt;user_dir&gt;</code>    | IA-64 アーキテクチャーの場合    |
| <code>make clean INSTALL_DIR=&lt;user_dir&gt;</code>      | すべてのアーキテクチャーの場合。     |



**注:** 上記のビルド/クリーンコマンドで `INSTALL_DIR=../..` または `INSTALL_DIR=<mkl_directory>` に設定すると、インテル® MKL 事前構築 Fortran 95 ライブラリーおよびモジュールを置換または削除できます。この処理は管理者権限がある場合に可能ですが、必要な場合を除いて行わないでください。

## コンパイラー依存の関数と Fortran 90 モジュール

コンパイラーがそのランタイム・ライブラリー (RTL) で解決されるオブジェクト・コード関数呼び出しを行うと常に、コンパイラー依存の関数が使用されます。適切な RTL なしでこれらのコードをリンクすると、未定義シンボルになります。インテル® MKL は、RTL の依存関係を最小限に抑えるように設計されています。

依存関係が発生する場合、サポートする RTL がインテル® MKL とともに提供されます。インテル® MKL クラスター・ソフトウェアに関連するものを除くと、このような RTL の唯一の例は、インテル® コンパイラーでコンパイルされる OpenMP\* コード用の `libiomp` および `libguide` です。`libiomp` と `libguide` は、インテル® MKL でスレッド化されたコードをサポートしています。

RTL 依存関係が発生する可能性がある場合、関数はソースコードで提供されます。アプリケーションで使用するコンパイラでコードをコンパイルしてください。

特に、Fortran 90 モジュールは、RTL をサポートするコンパイラ固有のコード生成が必要になるため、インテル® MKL はこれらのモジュールをソースコードで提供しています。

## 混在言語プログラミングとインテル® MKL

付録 A は、各インテル® MKL 関数ドメインでサポートされているプログラミング言語の一覧です。しかし、インテル® MKL ルーチンを異なる言語環境から呼び出すこともできます。このセクションでは、混在言語プログラミングを使用してこの呼び出しを行う方法を説明します。

## LAPACK、BLAS、および CBLAS ルーチンの C 言語環境からの呼び出し

すべてのインテル® MKL 関数ドメインで C と Fortran 環境の両方をサポートしているとは限りません。インテル® MKL Fortran 形式の関数を C/C++ 環境で使用するには、この後で説明されている LAPACK と BLAS の特定の規則に従う必要があります。



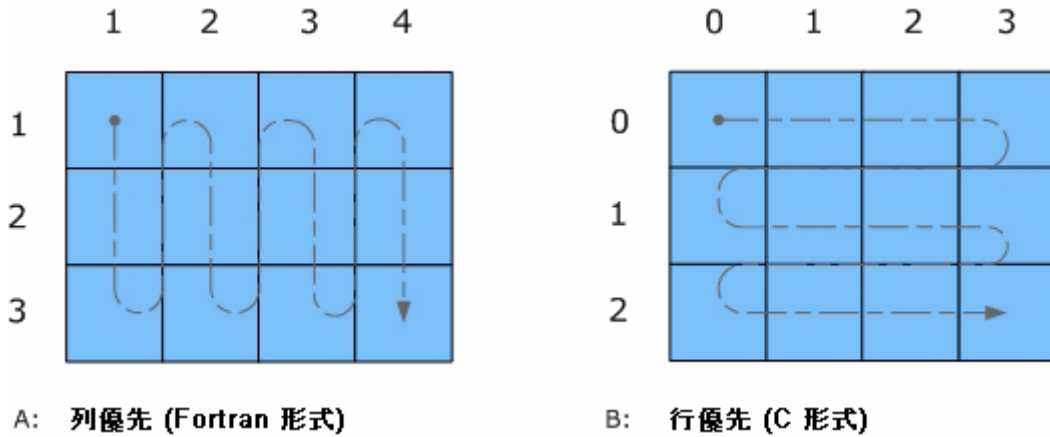
**警告:** BLAS95/LAPACK95 を C/C++ から呼び出さないでください。この呼び出しを行うには、Fortran 90 型の遅延形状配列のディスクリプターを処理する知識が必要です。また、BLAS95/LAPACK95 ルーチンには Fortran RTL へのリンクが含まれています。

### LAPACK

LAPACK ルーチンは Fortran 形式のため、C 言語プログラムから呼び出す場合、以下の Fortran 形式の呼び出し規則に従う必要があります。

- 変数を **値** ではなく **アドレス** で渡します。  
[例 7-2](#) および [例 7-3](#) の関数呼び出しを参照してください
- データを Fortran 形式、つまり行優先ではなく列優先で格納します。  
C で採用されている行優先で、配列が格納されているメモリーを全検索すると、最後の配列インデックスが最も速く変更され、最初の配列インデックスが最も遅く変更されます。  
Fortran 形式の列優先では、最後のインデックスが最も遅く変更され、最初のインデックスが最も速く変更されます ([図 7-1](#) の 2D 配列を参照)。

図 7-1 列優先と行優先



例えば、サイズ  $m \times n$  の 2 次元行列 A が 1 次元配列 B に格納されている場合、行列の要素は以下のようにアクセスされます。

C の場合:  $A[i][j] = B[i \cdot n + j]$  ( $i=0, \dots, m-1, j=0, \dots, n-1$ )

Fortran の場合:  $A(i, j) = B(j \cdot m + i)$  ( $i=1, \dots, m, j=1, \dots, n$ )

LAPACK ルーチンを C から呼び出す場合、Fortran 言語は大文字と小文字を区別しないことに注意してください。LAPACK ルーチンの名前には大文字と小文字の両方を使用できます (末尾の下線の有無を含む)。例えば、`dgetrf`、`DGETRF`、`dgetrf_`、`DGETRF_` は等価です。

## BLAS

BLAS ルーチンは Fortran 形式のルーチンです。BLAS ルーチンを C 言語プログラムから呼び出す場合、Fortran 形式の呼び出し規則に従う必要があります。

- 変数を値ではなくアドレスで渡します。
- データを Fortran 形式、つまり行優先ではなく列優先で格納します。

これらの規則の詳細は、「[LAPACK](#)」セクションを参照してください。BLAS ルーチンを C から呼び出す方法は、「[例 7-2](#)」を参照してください。

BLAS ルーチンを C から呼び出す場合、Fortran 言語は大文字と小文字を区別しないことに注意してください。BLAS ルーチンの名前には大文字と小文字の両方を使用できます (末尾の下線の有無を含む)。例えば、`dgemm`、`DGEMM`、`dgemm_`、`DGEMM_` は等価です。

## CBLAS

BLAS ルーチンを C 言語プログラムから呼び出す代わりに、CBLAS インターフェイスを使用することができます。

CBLAS は、BLAS ルーチンの C 形式のインターフェイスです。通常の C 形式の呼び出しを使用して CBLAS ルーチンを呼び出すことができます。CBLAS インターフェイスを使用する場合、ヘッダーファイル `mk1.h` によりすべての関数の列挙値とプロトタイプが指定されるため、プログラム開発は単純化されます。ヘッダーはプログラムが C++ コンパイラでコンパイルされているかどうかを判断し、コンパイルされている場合、インクルード・ファイルは C++ コンパイル用に設定されます。[例 7-3](#) は、CBLAS インターフェイスの使用例です。



## C/C++ での複素数型の使用

インテル® Fortran コンパイラーのアプリケーション構築ドキュメントで説明しているように、C/C++ では Fortran 型 COMPLEX(4) および COMPLEX(8) を直接実装していません。しかし、等価の構造を記述することはできます。型 COMPLEX(4) は、2つの4バイト浮動小数点からなります。最初の浮動小数点は実数コンポーネントで、2つめの浮動小数点は虚数コンポーネントです。型 COMPLEX(8) は、2つの8バイト浮動小数点を含むことを除けば、COMPLEX(4) と同じです。

インテル® MKL では、複素数型 MKL\_Complex8 および MKL\_Complex16 を提供しています。これらの構造は、Fortran 複素数型 COMPLEX(4) および COMPLEX(8) と等価です。これらの型は mkl\_types.h ヘッダーファイルで定義されています。これらの型を使用して、複素数データを定義できます。mkl\_types.h ヘッダーファイルをインクルードする前に、独自の型で型を再定義することも可能です。型を定義するときに必要なのは、Fortran 複素数レイアウトとの互換性を保つことです。つまり、複素数型は実部と虚部の値を含む実数のペアでなければなりません。

例えば、C++ コードで以下の定義を使用できます。

```
#define MKL_Complex8 std::complex<float>
```

および

```
#define MKL_Complex16 std::complex<double> .
```

詳細は、「[例 7-2](#)」を参照してください。コマンドラインでこれらの型を定義することもできます。  
-DMKL\_Complex8="std::complex<float>"  
-DMKL\_Complex16="std::complex<double>"

## C/C++ コードで複素数を返す BLAS 関数の呼び出し

C から複素数を返す BLAS 関数への呼び出しを制御している場合は注意が必要です。BLAS には Fortran 関数が含まれ、複素数の戻り値の制御は C と Fortran で異なるため、問題が発生します。しかし、Fortran では、通常関数呼び出しに加えて、関数が C プログラムから呼び出されたときに複素数の戻り値を返すように、関数をサブルーチンとして呼び出すことができます。Fortran 関数がサブルーチンとして呼び出された場合、戻り値は呼び出しシーケンスで最初のパラメーターになります。C プログラマーは、この違いに注意してください。

以下に呼び出し方法の違いを示します。

通常の Fortran 関数呼び出し: `result = cdotc( n, x, 1, y, 1 )`

関数をサブルーチンとして呼び出す場合:

```
call cdotc( result, n, x, 1, y, 1)
```

C から関数を呼び出す場合 ( 隠しパラメーターが表示されている点に注意 ):

```
cdotc( &result, &n, x, &one, y, &one )
```



**注:** インテル® MKL では、Fortran 形式の ( 大文字と小文字を区別しない ) BLAS で大文字と小文字の両方のエンタリーポイントを使用できます ( 末尾の下線の有無を含む )。このため、`cdotc`、`CDOTC`、`cdotc_`、`CDOTC_` はすべて等価です。

上記の例は、C/C++ アプリケーションから複素数を返すレベル 1 BLAS 関数を呼び出す方法の 1 つです。最も簡単な方法は、CBLAS を使用することです。例えば、以下のように CBLAS インターフェイスを使用して、同じ関数を呼び出すことができます。

```
cblas_cdotu( n, x, 1, y, 1, &result )
```



**注:** この場合、複素数は引数リストの最後になります。

以下に、Fortran 形式の BLAS インターフェイスと CBLAS (C 言語) インターフェイスを C/C++ から使用する方法を示します。

次の例は、C プログラムから複素レベル 1 BLAS 関数 `zdotc()` を呼び出す方法を表しています。この関数は、2 つの倍精度複素ベクトルのドット積を計算します。

この例では、複素数型のドット積が構造体 `c` に返されます。

#### 例 7-1 複素レベル 1 BLAS 関数の C からの呼び出し

```
#include "mkl.h"
#define N 5
void main()
{
    MKL_int n = N, inca = 1, incb = 1, i;
    MKL_Complex16 a[N], b[N], c;
    for( i = 0; i < n; i++ ){
        a[i].re = (double)i; a[i].im = (double)i * 2.0;
        b[i].re = (double)(n - i); b[i].im = (double)i * 2.0;
    }
    zdotc( &c, &n, a, &inca, b, &incb );
    printf( " 複素ドット積: ( %6.2f, %6.2f)\n", c.re, c.im );
}
```

以下は C++ 実装です。

#### 例 7-2 複素レベル 1 BLAS 関数の C++ からの呼び出し

```
#include <complex>
#include <iostream>
#define MKL_Complex16 std::complex<double>
#include "mkl.h"

#define N 5

int main()
{
    int n, inca = 1, incb = 1, i;
    std::complex<double> a[N], b[N], c;
    n = N;
```

**例 7-2 複素レベル 1 BLAS 関数の C++ からの呼び出し ( 続き )**

```

for( i = 0; i < n; i++ ){
    a[i] = std::complex<double>(i,i*2.0);
    b[i] = std::complex<double>(n-i,i*2.0);
}
zdotc(&c, &n, a, &inca, b, &incb );
std::cout << " 複素ドット積 : " << c << std::endl;
return 0;
}

```

以下のサンプルは CBLAS を使用しています。

**例 7-3 BLAS を C から直接呼び出す代わりに CBLAS インターフェイスを使用**

```

#include "mkl.h"
typedef struct{ double re; double im; } complex16;

extern "C" void cblas_zdotc_sub ( const int , const complex16 *,
                                const int , const complex16 *, const int, const complex16*);

#define N 5

void main()
{

    int n, inca = 1, incb = 1, i;

    complex16 a[N], b[N], c;
    n = N;
    for( i = 0; i < n; i++ ){

        a[i].re = (double)i; a[i].im = (double)i * 2.0;
        b[i].re = (double)(n - i); b[i].im = (double)i * 2.0;
    }
    cblas_zdotc_sub(n, a, inca, b, incb,&c );
    printf( " 複素ドット積 : ( %6.2f, %6.2f)\n", c.re, c.im );
}

```

**Boost uBLAS 行列 - 行列乗算のサポート**

uBLAS を使用する場合、Boost uBLAS 関数の代わりにインテル® MKL の関数を使って、C++ で BLAS 行列 - 行列乗算を実行できます。uBLAS は Boost C++ オープンソース・ライブラリーに含まれているライブラリーで、密行列、圧縮行列、およびスパース行列で BLAS の機能を利用できます。ライブラリーは式を関数の引数として渡すために式テンプレート方式を使用しており、1 つのパスで ( 一時行列を使用しないで ) ベクトルと行列式を評価できます。uBLAS には 2 つのモードがあります。

- デバッグ (セーフ) モード (デフォルト)  
型と適合性のチェックが行われます。
- リリース (高速) モード  
NDEBUG プリプロセッサ・シンボルにより有効になります。

Boost uBLAS のドキュメントは、<http://www.boost.org/> ( 英語 ) から入手できます。

インテル® MKL には、uBLAS *dense* 行列・行列乗算をインテル® MKL `gemm` 呼び出しに置換するオーバーロードされた `prod()` 関数が用意されています。これらの関数は uBLAS 式テンプレートではなく一時行列を使用しますが、行列のサイズが小さくない (50 以上) 場合はパフォーマンスの向上が期待できます。

関数を使用するためにソースコードを変更する必要はありません。関数の呼び出し方法は以下のとおりです。

- コードに (インテル® MKL インクルード・ディレクトリーから) ヘッダーファイル `mkl_boost_ublas_matrix_prod.hpp` をインクルードします。
- 適切なインテル® MKL ライブラリーをリンク行に追加します (「[アプリケーションとインテル® マス・カーネル・ライブラリーのリンク](#)」を参照)。

以下の式が置換されます。

```
prod( m1, m2 )
prod( trans(m1), m2 )
prod( trans(conj(m1)), m2 )
prod( conj(trans(m1)), m2 )
prod( m1, trans(m2) )
prod( trans(m1), trans(m2) )
prod( trans(conj(m1)), trans(m2) )
prod( conj(trans(m1)), trans(m2) )
prod( m1, trans(conj(m2)) )
prod( trans(m1), trans(conj(m2)) )
prod( trans(conj(m1)), trans(conj(m2)) )
prod( conj(trans(m1)), trans(conj(m2)) )
prod( m1, conj(trans(m2)) )
prod( trans(m1), conj(trans(m2)) )
prod( trans(conj(m1)), conj(trans(m2)) )
prod( conj(trans(m1)), conj(trans(m2)) )
```

これらの式は、リリースモードでのみ (NDEBUG プリプロセッサ・シンボルが定義されている場合のみ) 置換されます。サポートしている uBLAS のバージョンは、Boost 1.34.1、1.35.0、1.36.0、および 1.37.0 です。<http://www.boost.org/> から入手できます。

<mkl ディレクトリー>/examples/ublas/source/sylvester.cpp ファイルのコードサンプルは、シルベスター方程式の特別なケースを解くインテル® MKL uBLAS ヘッダーファイルの使用例です。

インテル® MKL ublas サンプルを実行するには、BOOST\_ROOT パラメーターを make コマンドで指定します。例えば、Boost バージョン 1.37.0 を使用する場合は、次のように指定します。

```
make lib32 BOOST_ROOT=<パス>/boost_1_37_0
```

## インテル® MKL 関数の Java アプリケーションからの呼び出し

このセクションでは、インテル® MKL パッケージで提供されるサンプルとライブラリー関数を Java から呼び出す方法を説明します。

### インテル® MKL の Java サンプル

Java は、開発元である米 Sun Microsystems 社によって、WORA (Write Once Run Anywhere: 一度書けばどこでも実行可能) 言語と位置付けられています。インテル® MKL は、大部分のデスクトップ PC とノートブック PC、多くのワークステーションとサーバーをカバーする、広範囲なオペレーティング・システムとプロセッサ用にさまざまなエディションを提供することで、WORA 哲学を部分的にサポートし、Java アプリケーションの高速化を支援します。

インテル® MKL には以下のディレクトリーにさまざまな Java のサンプルが含まれています。

<mk1 ディレクトリー>/examples/java

以下のインテル® MKL 関数用のサンプルが提供されています。

- CBLAS の ?gemm、?gemv、および ?dot ファミリー
- 非クラスター FFT 関数の完全なセット
- 1 次元の畳み込み / 相関関数 ESSL<sup>1</sup> 形式の関数
- ユーザー定義のものとファイル・サブルーチンを除く VSL 乱数生成器 (RNG)
- GetErrorCallBack、SetErrorCallBack、および ClearErrorCallBack を除く VML 関数

サンプルのソースは以下のディレクトリーにあります。

<mk1 ディレクトリー>/examples/java/examples

サンプルは Java で記述されています。サンプルでは、以下の種類のデータを使用しています。

- 1 次元および 2 次元データシーケンス
- データの実数型と複素数型
- 単精度と倍精度

ただし、サンプルで使用されているラッパーは以下のことを行いません。

- 巨大な配列 (要素が 2 億以上) の使用
- ネイティブメモリー中の配列の処理
- 関数パラメーターの正当性の確認
- パフォーマンスの最適化

インテル® MKL とバインドするため、サンプルは Java Native Interface (JNI 開発者フレームワーク) を使用しています。JNI のドキュメントは、<http://java.sun.com/javase/6/docs/ja/technotes/guides/jni/> から入手できます。

Java のサンプルには、バインドを行う JNI ラッパーも含まれています。ラッパーはサンプルに依存しません。独自の Java アプリケーションで使用することもできます。CBLAS、FFT、VML、VSL RNG、および ESSL 形式の畳み込み / 相関関数のラッパーは互いに依存しません。

ラッパーをビルドするには、サンプルを実行してください (詳細は、「[サンプルの実行](#)」を参照)。メイクファイルを実行すると、ラッパーのバイナリーがビルドされます。メイクファイルを実行した後、サンプルを実行して、ラッパーが正しくビルドされたかどうかを確認できます。サンプルを実行すると、

<mk1 ディレクトリー>/examples/java に以下のディレクトリーが作成されます。

- docs
- include

1. IBM ESSL (Engineering Scientific Subroutine Library)。

- `classes`
- `bin`
- `_results`

`docs`、`include`、`classes`、および `bin` ディレクトリーには、ラッパーのバイナリーとドキュメントが含まれます。`_results` ディレクトリーには、テスト結果が含まれます。

Java プログラマーにとっては、ラッパーは次のような Java クラスに見えます。

- `com.intel.mkl.CBLAS`
- `com.intel.mkl.DFTI`
- `com.intel.mkl.ESSL`
- `com.intel.mkl.VML`
- `com.intel.mkl.VSL`

特定のラッパーとサンプルのクラスのドキュメントは、サンプルのビルドおよび実行中に Java ソースから生成されます。ドキュメントを参照するには、(ビルドスクリプトを実行すると作成される) `docs` ディレクトリーにある次のファイルを開いてください。

<mk1 ディレクトリー>/examples/java/docs/index.html

CBLAS、VML、VSL RNG、および FFT 用の Java ラッパーは、基本となるネイティブ関数に直接対応するインターフェイスを確立します。機能とパラメーターについては、『インテル® MKL リファレンス・マニュアル』を参照してください。ESSL 形式の関数用のインターフェイスは、`com.intel.mkl.ESSL` クラス用に生成されたドキュメントで説明されています。

各ラッパーは、Java のインターフェイス部分と C で記述された JNI スタブで構成されています。ソースは以下のディレクトリーにあります。

<mk1 ディレクトリー>/examples/java/wrappers

CBLAS と VML 用のラッパーの Java と C 部分はどちらも標準的なアプローチを採用しているため、追加の CBLAS 関数をカバーするために使用できます。

FFT 用のラッパーは、FFT ディスクリプター・オブジェクトのライフサイクルをサポートする必要があるため、より複雑です。単一フーリエ変換を計算するには、アプリケーションはネイティブ FFT ディスクリプターの同じコピーを使用して FFT ソフトウェアを複数回呼び出す必要があります。ラッパーは、仮想マシンが Java バイトコードを実行する間、ネイティブ・ディスクリプターを保持するハンドラークラスを提供します。

VSL RNG 用のラッパーは、FFT 用のラッパーと似ています。ラッパーは、ストリームステートのネイティブ・ディスクリプターを保持するハンドラークラスを提供します。

畳み込み / 相関関数用のラッパーは、“タスク・ディスクリプター”と同様のライフサイクルを仮定し、VSL インターフェイスの難易度を緩和します。ラッパーは、1 次元でより単純な、これらの関数の ESSL 形式の関数を使用します。JNI スタブは、C で記述された ESSL 形式のラッパーにインテル® MKL 関数をラップし、ネイティブメソッドへの単一呼び出しにタスク・ディスクリプターのライフサイクルを“パック”します。

ラッパーは、JNI 仕様 1.1 および 5.0 を満たしているため、新しい Java のすべての実装で動作します。

サンプルとラッパーの Java 部分は、「*The Java Language Specification (First Edition)*」で説明されている Java 言語用に記述され、1990 年代後半に登場したインナークラスの機能が拡張されています。この言語バージョンのレベルは、Sun の JDK (Java 開発キット) のすべてのバージョンと、バージョン 1.1.5 以降の互換性のある実装をサポートしています。

C 言語レベルは、インテル® MKL インターフェイスと JNI ヘッダーファイルに必要な、整数と浮動小数点データ型に関する追加の仮定を含む“標準 C” (C89) です。つまり、ネイティブ `float` および `double` データ型は、JNI `jfloat` および `jdouble` データ型とそれぞれ同じである必要があります。また、ネイティブ `int` データ型は、4 バイト長でなければなりません。

## サンプルの実行

Java のサンプルは、インテル® MKL でサポートされている C/C++ コンパイラーをすべてサポートしています。メイクファイルを実行するには、Linux® OS ディストリビューションで提供される make ユーティリティーが必要です。

Java のサンプルを実行するには、Java コードのコンパイルと実行に JDK 開発者ツールキットが必要です。Java 実装はコンピューターにインストールされているか、ネットワーク経由で利用可能でなければなりません。JDK は、各ベンダーの Web サイトからダウンロードできます。

サンプルは、JDK のすべてのバージョンで動作するように作成されていますが、以下の Java 実装でのみテストされています。

- Sun Microsystems 社(<http://sun.com>)の J2SE® SDK 1.4.2、JDK 5.0 および 6.0
- BEA Systems 社(<http://bea.com>)の JRockit® JDK 1.4.2 および 5.0



---

**注：** Sun Microsystems 社の実装は、IA-32 およびインテル® 64 アーキテクチャー対応プロセッサのみをサポートしています。BEA Systems 社の JRockit JDK は、インテル® Itanium® プロセッサもサポートしています。

---

コンピューターに JRE (Java ランタイム環境) がインストールされている場合でも、以下のツールをサポートする JDK 開発者ツールキットが必要です。

- java
- javac
- javah
- javadoc

これらのツールをサンプルのメイクファイルで利用できるようにするには、例えば、bash シェルを使用して、JAVA\_HOME 環境変数を設定し、PATH 環境変数に JDK バイナリーのディレクトリーを追加する必要があります。

```
export JAVA_HOME=/home/<user name>/jdk1.5.0_09
export PATH=${JAVA_HOME}/bin:${PATH}
```

JDK\_HOME 環境変数に値が割り当てられている場合は、この環境変数をクリアしてください。

```
unset JDK_HOME
```

サンプルを開始するには、インテル® MKL の Java サンプルが含まれているディレクトリーにあるメイクファイルを使用します。

```
make {so32|soem64t|so64|lib32|libem64t|lib64} [function=...] [compiler=...]
```

ターゲット (so32) を省略して make コマンドを実行すると、ターゲットとパラメーターを説明するヘルプ情報が表示されます。

サンプルのリストは、Java サンプルのディレクトリーにある `examples.lst` ファイルを参照してください。

## 既知の制限事項

次の 3 種類の制限があります。

- 機能
- パフォーマンス
- 既知の問題

**機能:** インテル® MKL の Java サンプルのように、ラッパーを使用して Java 環境から呼び出された場合、一部のインテル® MKL 関数が正常に動作しない可能性があります。「[インテル® MKL の Java サンプル](#)」セクションにリストされている、CBLAS、FFT、VML、VSL RNG、および畳み込み / 相関関数は、Java 環境でテストされています。このため、独自の Java アプリケーションでは、これらの CBLAS、FFT、VML、VSL RNG、および畳み込み / 相関関数用の Java ラッパーを使用してください。

**パフォーマンス:** インテル® MKL 関数はピュア Java で記述された同様の関数よりも高速です。ただし、これらのラッパーの目的は、最大限のパフォーマンスを提供することではなく、コードサンプルを提供することです。このため、Java アプリケーションから呼び出されたインテル® MKL 関数は、C/C++ または Fortran で記述されたプログラムから呼び出された同じ関数よりも遅くなります。

**既知の問題:** インテル® MKL には (『リリースノート』に示されている) 既知の問題があります。また、JDK の異なるバージョンでは互換性がないものがあります。サンプルおよびラッパーには、サンプルを動作させるために、これらの問題の回避策が用意されています。サンプルおよびラッパーのソースコードに記述されている、回避策についてのコメントを参照してください。



# コーディングのヒント

## 8

本章は、インテル® マス・カーネル・ライブラリー (インテル® MKL) を使用したプログラミングについて、数値計算の安定性など、特定の用途を満たすために役立つコーディングのヒントを紹介します。一般的な言語固有のプログラミング・オプションは第7章を、パフォーマンスとメモリー管理に関連するコーディングのヒントは第6章を参照してください。

## 数値計算安定性のためのデータの整列

線形代数ルーチン (LAPACK, BLAS) がビットごとに同一の入力データに適用され、配列のアライメントが異なるか、計算が異なるプラットフォーム上や異なるスレッド数で行われる場合、出力はビットごとに同一ではないため、適切な誤差範囲に収まらないことがあります。バージョンによってルーチンの実装が異なる場合があるため、インテル® MKL のバージョンは出力の数値計算安定性に影響します。指定されたインテル® MKL バージョンで、以下の条件がすべて満たされる場合、出力はビットごとに同一になります。

- 出力が同じプラットフォームで得られる
- 入力がビットごとに同一である
- 入力配列が 16 バイト境界で同一にアライメントされている
- インテル® MKL が逐次モードで実行されている

ユーザーの管理下にある最初の 2 つの条件とは異なり、配列はデフォルトではアライメントされていません。例えば、`malloc` を使用して動的に割り当てられた配列は、16 バイトではなく 8 バイト境界でアライメントされます。数値的に同じ出力が必要な場合、以下のように、`mkl_malloc()` を使用して正しくアライメントされたワークスペースを取得してください。

**例 8-1 16 バイト境界でアドレスをアライメント**

---

```
// ***** C 言語 *****  
...  
#include <stdlib.h>  
...  
void *darray;  
int workspace;  
...  
// 16 ビット境界でアライメントされたワークスペースを割り当てる  
darray = mkl_malloc( sizeof(double)*workspace, 16 );  
...  
// MKL を使用してプログラムを呼び出す  
mkl_app( darray );  
...  
// ワークスペースを解放する  
mkl_free( darray );  
  
! ***** Fortran 言語 *****  
...  
double precision darray  
pointer (p_wrk,darray(1))  
integer workspace  
...  
! 16 ビット境界でアライメントされたワークスペースを割り当てる  
p_wrk = mkl_malloc( 8*workspace, 16 )  
...  
! MKL を使用してプログラムを呼び出す  
call mkl_app( darray )  
...  
! ワークスペースを解放する  
call mkl_free(p_wrk)
```

---

# インテル® マス・カーネル・ ライブラリー・クラスター・ ソフトウェアの使用

## 9

本章は、インテル® マス・カーネル・ライブラリー (インテル® MKL) ScaLAPACK とクラスター FFT の使用法について説明します。C と Fortran 固有のリンク例を含む、関数ドメインとアプリケーションのリンク方法を主に示します。また、サポートしている MPI の情報についても説明します。

インテル® MKL のディレクトリー構造の詳細と doc ディレクトリーに含まれるドキュメントについては、第 3 章を参照してください。

クラスター用の MP LINPACK ベンチマークに関する情報は、第 11 章を参照してください。

インテル® MKL ScaLAPACK およびクラスター FFT は、『インテル® MKL リリースノート』に記述されている MPI 実装をサポートしています。

ScaLAPACK を呼び出すプログラムをリンクするには、まず MPI アプリケーションのリンク方法を知っておく必要があります。

通常は、正しい MPI ヘッダーファイルを使用する *mpi* スクリプト *mpicc* または *mpif77* (C または Fortran77 スクリプト) を使用します。例えば、*/opt/mpich* にインストールした MPICH を使用している場合、通常、*/opt/mpich/bin/mpicc* と */opt/mpich/bin/mpif77* がコンパイラー・スクリプト、*/opt/mpich/lib/libmpich.a* がインストールに使用したライブラリーです。

## ScaLAPACK およびクラスター FFT とのリンク

インテル® MKL ScaLAPACK とクラスター FFT の両方またはいずれか一方とリンクするには、以下の一般的な形式を使用します。

```
<<MPI>> リンカースクリプト >> リンクするファイル > \
-L<MKL パス> [-Wl,--start-group] <MKL クラスター・ライブラリー> \
<BLACS> <MKL コア・ライブラリー> [-Wl,--end-group]
```

説明:

<MPI> は、MPI 実装 (MPICH、インテル® MPI 2.x/3.x、...) のいずれか 1 つです。

<MKL クラスター・ライブラリー> は、[表 3-5](#)、[表 3-6](#) および [表 3-7](#) にリストされている、該当アーキテクチャー用の ScaLAPACK またはクラスター FFT ライブラリーのいずれか 1 つです。例えば、IA-32 アーキテクチャーの場合、*-lmkl\_scalapack\_core* または *-lmkl\_cdft\_core* のいずれかです。

<BLACS> は、[表 3-5](#)、[表 3-6](#) および [表 3-7](#) にリストされている、該当アーキテクチャー用の BLACS ライブラリーの 1 つです。例えば、IA-32 アーキテクチャーの場合、*-lmkl\_blacs*、*-lmkl\_blacs\_intelmpi*、*-lmkl\_blacs\_openmpi* のいずれか 1 つです。

<MKL コア・ライブラリー> は、ScaLAPACK の場合は <MKL LAPACK & MKL カーネル・ライブラリー>、クラスター FFT の場合は <MKL カーネル・ライブラリー> です。

<MKL カーネル・ライブラリー> は、「[リンク行のライブラリーのリスト](#)」に記述されているように、スレッド化をサポートするためにリンクされる、プロセッサ最適化カーネル、スレッド化ライブラリー、およびシステム・ライブラリーです。

<MKL LAPACK & カーネル・ライブラリー> は、LAPACK ライブラリーと <MKL カーネル・ライブラリー> です。

スタティック・リンクの場合は、グループ化シンボル `-Wl,--start-group` および `-Wl,--end-group` が必要です。

例えば、インテル® MPI 3.x で ScaLAPACK の LP64 インターフェイスをスタティックに使用し、コアあたりの MPI プロセスが 1 つの場合 (つまり、スレッド化を行わない場合)、次のリンカーオプションを指定します。

```
-L$MKL_PATH -I$MKL_INCLUDE -Wl,--start-group
$MKL_PATH/libmkl_scalapack_lp64.a $MKL_PATH/libmkl_blacs_intelmpi_lp64.a
$MKL_PATH/libmkl_intel_lp64.a $MKL_PATH/libmkl_sequential.a
$MKL_PATH/libmkl_core.a -static_mpi -Wl,--end-group -lpthread -lm
```

詳細は、「[ScaLAPACK およびクラスター FFT とのリンク例](#)」を参照してください。

<<MPI> リンカースクリプト> および <BLACS> ライブラリーは、MPI のバージョンに対応していなければなりません。例えば、インテル® MPI 3.x の場合、<インテル® MPI 3.x リンカースクリプト> および `libmkl_blacs_intelmpi` ライブラリーを使用します。

インテル® MKL ライブラリーのリンクについては、「[アプリケーションとインテル® マス・カーネル・ライブラリーのリンク](#)」を参照してください。

## スレッド数の設定

OpenMP® ソフトウェアは、`OMP_NUM_THREADS` 環境変数を使用します。インテル® MKL には、`MKL_NUM_THREADS` や `MKL_DOMAIN_NUM_THREADS` のような、スレッド数を設定するほかの方法も用意されています (「[新しいスレッド化コントロールの使用](#)」を参照)。関連する環境変数がすべてのノードにおいて同じで正しい値になっていることを確認してください。また、インテル® MKL バージョン 10.0 以降、デフォルトのスレッド数は 1 に設定されません。コンパイラーと一緒に使用している OpenMP ライブラリーに応じて、デフォルトのスレッド数が設定されます。インテル® コンパイラー・ベースのスレッド化レイヤー (`libmkl_intel_thread.a`) では、この値は OS の CPU の数です。例えば、ノードあたりの MPI ランクの数とノードあたりのスレッド数の両方が 1 よりも大きい場合、スレッド数が過剰に指定されないように注意してください。

`OMP_NUM_THREADS` のような環境変数を設定する最良の方法は、ログイン環境です。mpirun はすべてのノードでデフォルトシェルを開始することに注意してください。このため、ヘッドノードでこの値を変更してから実行 (SMP システムで動作) しても、すべてのノードで変数の変更は反映されません。`.bashrc` で、以下のように先頭に行を追加してください。

```
OMP_NUM_THREADS=1; export OMP_NUM_THREADS
```

MPICH を使用してノードごとに複数の CPU を実行することは可能ですが、そのためには MPICH を構築する必要があります。特定の MPICH アプリケーションはスレッド化環境では正常に動作しない場合があることに注意してください (『リリースノート』の「既知の制限事項」セクションを参照)。MPICH で 1 よりも大きなスレッド数を設定したときに問題が発生した場合は、まずスレッド数を 1 に設定して問題が解消するかどうか確認してください。

## 共有ライブラリーの使用

すべての必要な共有ライブラリーはランタイム時にすべてのノードで見えてする必要があります。この状況を実現する 1 つの方法は、`.bashrc` ファイルで `LD_LIBRARY_PATH` 環境変数を使用してこれらのライブラリーを指すことです。インテル® MKL が 1 つのノードにのみインストールされている場合、インテル® MKL アプリケーションをビルドするとき静的にリンクしてください。

インテル® コンパイラーまたは GNU コンパイラーは、インテル® MKL を使用するプログラムをコンパイルすることができます。MPI 実装とコンパイラーが正しく一致していることを確認してください。

## ScaLAPACK テスト

NetLib ScaLAPACK テストのビルド方法は次のとおりです。

- IA-32 アーキテクチャーの場合、リンクコマンドに `libmkl_scalapack_core.a` を追加します。
- IA-64 およびインテル® 64 アーキテクチャーの場合、目的のインターフェイスに応じて、`libmkl_scalapack_lp64.a` または `libmkl_scalapack_ilp64.a` を追加します。

## ScaLAPACK およびクラスター FFT とのリンク例

インテル® MKL のアーキテクチャー固有のディレクトリー構造とリンクするクラスター・ライブラリーの名前の詳細は、「[詳細なディレクトリー構造](#)」を参照してください。

## C アプリケーションのリンク例

以下の条件が満たされていると仮定します。

- MPICH2 1.0.7 以降が `/opt/mpich` にインストールされている。
- `$MKL_PATH` が `<mkl ディレクトリー>/lib/32` を含むユーザー定義変数である。
- インテル® C++ コンパイラー 9.1 以降を使用しており、メインモジュールが C である。

**IA-32 アーキテクチャー・ベースのシステムのクラスターで ScaLAPACK をリンクするには：**

以下のコマンドを使用します。

```
/opt/mpich/bin/mpicc <リンクするユーザーファイル> \
-L$MKL_PATH \
-lmkl_scalapack_core \
-lmkl_blacs_intelmpi \
-lmkl_lapack \
-lmkl_intel -lmkl_intel_thread -lmkl_lapack -lmkl_core \
-liomp5 -lpthread
```

**IA-32 アーキテクチャー・ベースのシステムのクラスターでクラスター FFT をリンクするには：**

以下のコマンドを使用します。

```
/opt/mpich/bin/mpicc <リンクするユーザーファイル> \
$MKL_PATH/libmkl_cdft_core.a \
$MKL_PATH/libmkl_blacs_intelmpi.a \
$MKL_PATH/libmkl_intel.a \
$MKL_PATH/libmkl_intel_thread.a \
$MKL_PATH/libmkl_core.a \
-liomp5 -lpthread
```

## Fortran アプリケーションのリンク例

以下の条件が満たされていると仮定します。

- インテル® MPI 3.0 が `/opt/intel/mpi/3.0` にインストールされている。
- `$MKL_PATH` が `<mkl ディレクトリー>/lib/64` を含むユーザー定義変数である。
- インテル® Fortran コンパイラー 9.1 以降を使用しており、メインモジュールが Fortran である。

### IA-64 アーキテクチャー・ベースのシステムのクラスターで ScaLAPACK をリンクするには :

以下のコマンドを使用します。

```
/opt/intel/mpi/3.0/bin/mpiifort < リンクするユーザーファイル> \
-L$MKLPATH \
-lmkl_scalapack_lp64 \
-lmkl_blacs_intelmpi_lp64 \
-lmkl_lapack \
-lmkl_intel_lp64 -lmkl_intel_thread -lmkl_lapack -lmkl_core \
-liomp5 -lpthread
```

### IA-64 アーキテクチャー・ベースのシステムのクラスターでクラスター FFT をリンクするには :

以下のコマンドを使用します。

```
/opt/intel/mpi/3.0/bin/mpiifort < リンクするユーザーファイル> \
$MKLPATH/libmkl_cdft_core.a \
$MKLPATH/libmkl_blacs_intelmpi_ilp64.a \
$MKLPATH/libmkl_intel_ilp64.a \
$MKLPATH/libmkl_intel_thread.a \
$MKLPATH/libmkl_core.a \
-liomp5 -lpthread
```

ScaLAPACK とリンクされたバイナリーは、ほかの MPI アプリケーションと同じ方法で動作します ( 詳細は、MPI 実装に含まれているドキュメントを参照してください )。例えば、スクリプト `mpirun` は MPICH 2 および OpenMPI の場合に使用され、MPI プロセスの数は `-np` で設定されます。MPICH 2.0 およびすべてのインテル® MPI の場合、アプリケーションを実行する前にデーモンを開始する必要があります。実行にはスクリプト `mpiexec` を使用します。

他のリンク例は、インテル製品のサポート Web サイト

<http://www.intel.com/software/products/support/> ( 英語 ) を参照してください。

# Eclipse\* IDE での プログラミング支援機能

# 10

本章は、ソフトウェア・エンジニアが Eclipse\* IDE で作業するときに利用できるインテル® MKL の機能を説明します。

以下の機能が Eclipse IDE でのプログラミングを支援するために用意されています。

- IDE 内でインテル® MKL リファレンス・マニュアルを表示
- Eclipse Help でインテルの Web サイトを検索
- Eclipse C/C++ Development Tools (CDT) の状況依存ヘルプ
- Eclipse CDT の Code/Content Assist

最初の 3 つの機能は、Eclipse Help 用のインテル® MKL プラグインにより提供されます (インストール後のプラグインの場所は、[表 3-1](#) を参照)。プラグインを使用するには、Eclipse ディレクトリーの plugins フォルダーにプラグインを配置してください。

最後の機能は、Eclipse CDT 固有の機能です。Eclipse ヘルプの「Code Assist」セクションを参照してください。

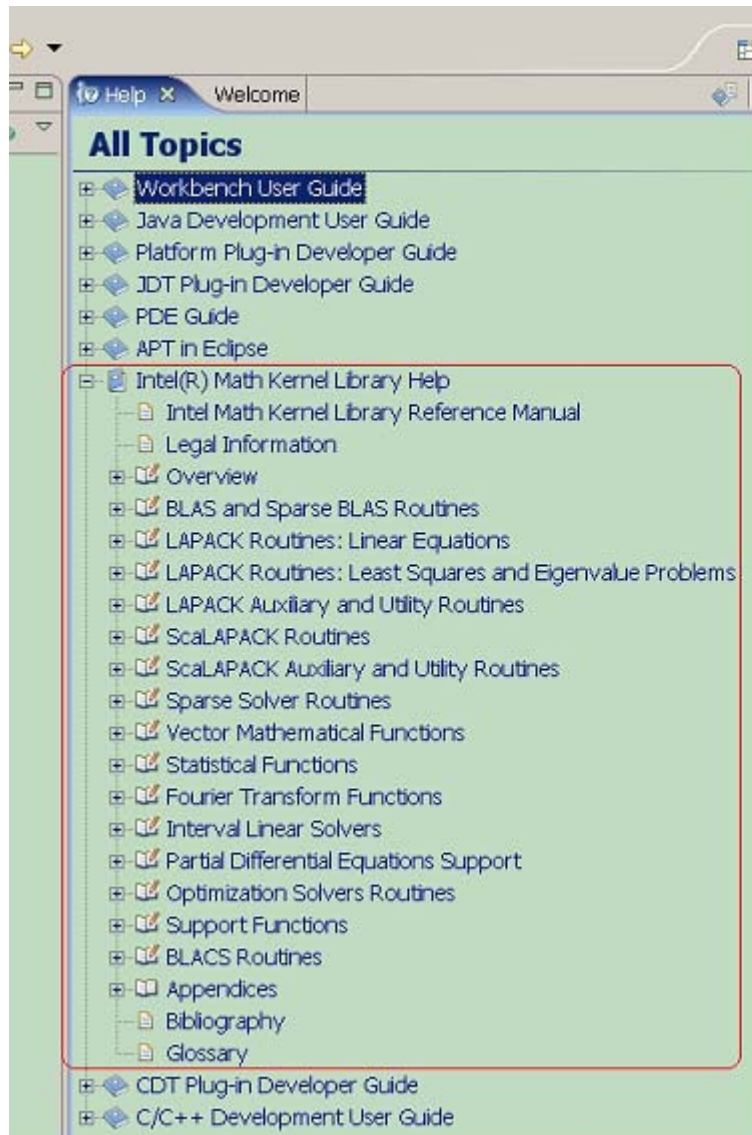
## Eclipse IDE 内でインテル® MKL リファレンス・マニュアルを表示

Eclipse でリファレンス・マニュアルを表示するには、次の手順を行います。

1. メニューから **[Help] > [Help Contents]** を選択します。
2. **[Help]** タブの **[All Topics]** で、**[Intel(R) Math Kernel Library Help]** をクリックします。
3. **[Help]** ツリーを展開して、**[Intel MKL Reference Manual]** をクリックします ([図 10-1](#) を参照)。

インテル® MKL ヘルプ索引が Eclipse で利用できます。リファレンス・マニュアルは Eclipse Help の検索に含まれます。

図 10-1 Eclipse IDE のインテル® MKL ヘルプ



## Eclipse IDE からインテルの Web サイトを検索

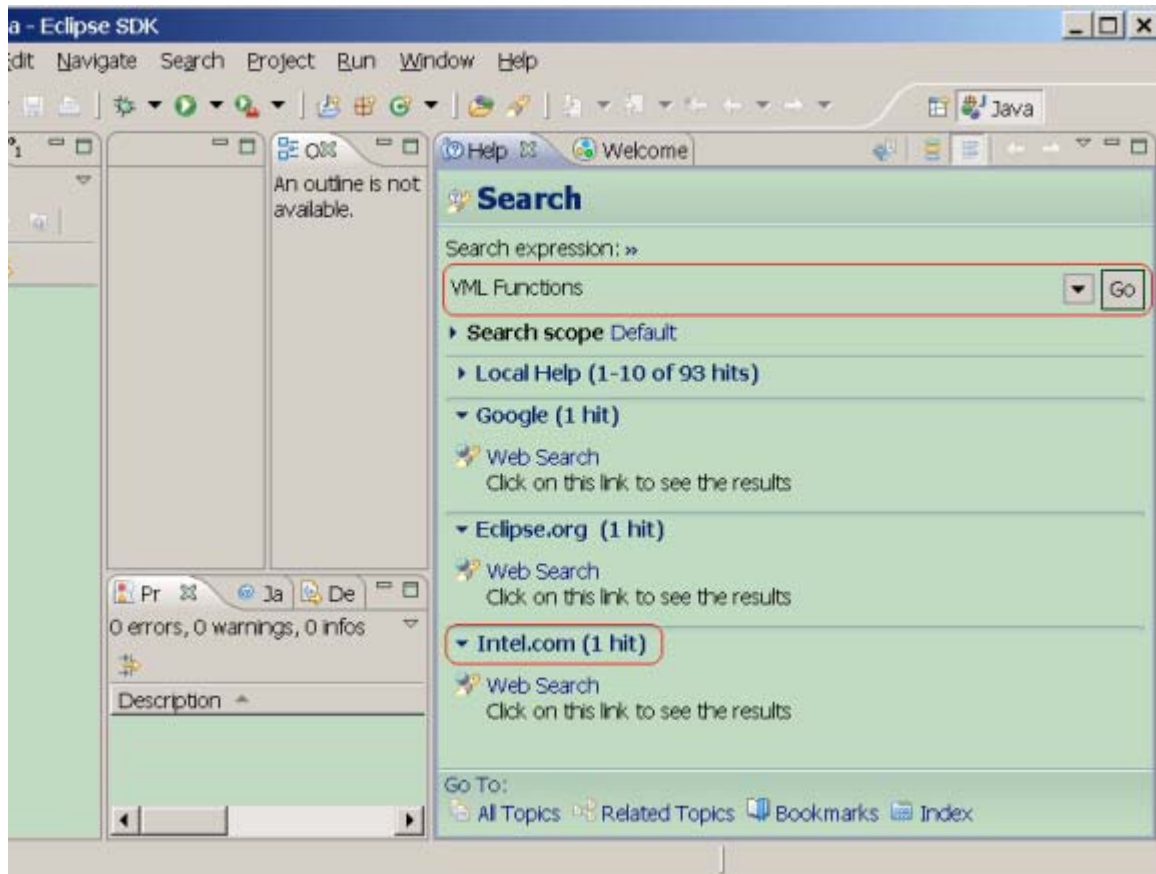
インテル® MKL プラグインにより、Eclipse Help で <http://www.intel.com> が検索されるようになります。Eclipse Help ペインで検索を行うと、サイトで見つけた数が別々のリンクで表示されます。

図 10-2 は、Eclipse Help で "VML Functions" を検索した結果です。ここで、**1 hit** は各サイトでエントリーが 1 つ見つかったことを意味します。

"Intel.com (1 hit)" をクリックすると、インテルの Web サイトで見つけた項目のリストが表示されます。



図 10-2 Eclipse IDE Help の検索でインテルの Web サイトに見つかった数



## Eclipse IDE CDT での状況依存ヘルプの使用

Eclipse CDT エディターで状況依存ヘルプ (Infopop ウィンドウおよび F1 ヘルプ) を表示することができます。

### Infopop ウィンドウ

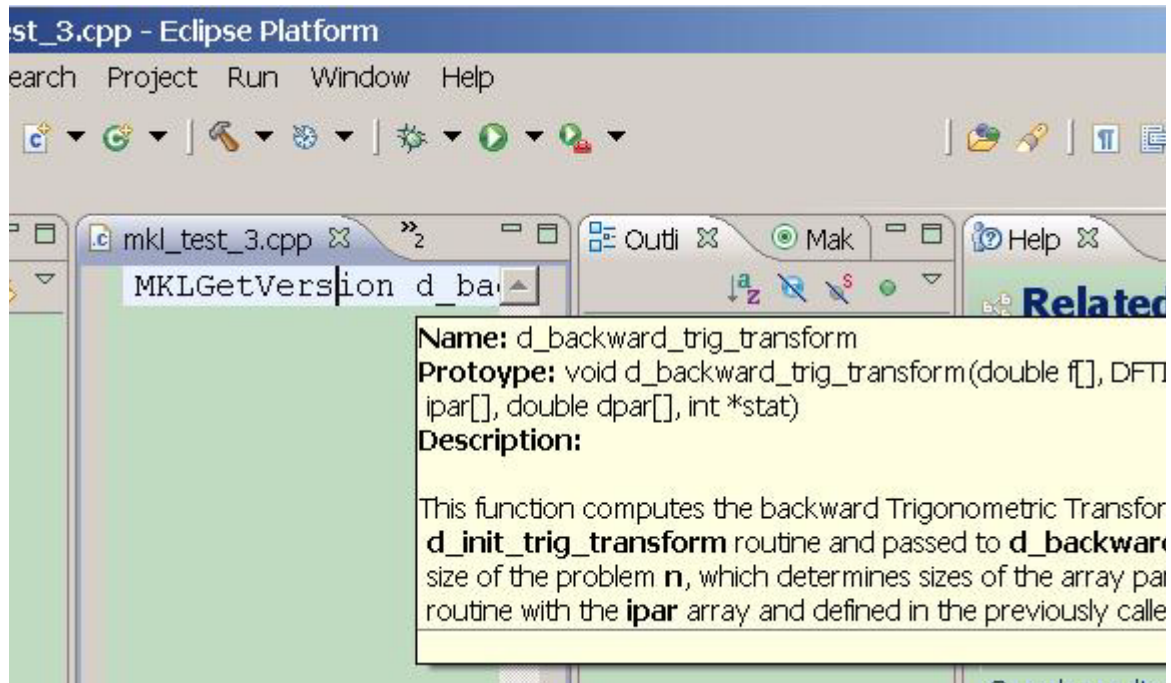
Infopop ウィンドウは、C の関数を説明するポップアップ・ウィンドウです。



**注:** 現在のリリースでは、この機能は VML 関数でのみ利用できます。

エディターで入力した名前のインテル® MKL 関数の説明を表示するには、マウスのポインターを関数名の上に移動します。

図 10-3 Infopop ウィンドウに表示されたインテル® MKL 関数の説明



## F1 ヘルプ

F1 ヘルプは、キーワードに関連するドキュメントのトピックのリストを表示します。

エディター・ウィンドウで入力した名前のインテル® MKL 関数の F1 ヘルプを表示するには、次の手順を行います。

1. マウスのポインターを関数名の上に移動します。
2. F1 を押すか、名前をダブルクリックします。  
2つのリストが表示されます。
  - 製品ドキュメントの関連トピックへのリンクのリストが、[See also] の [Related Topics] ページに表示されます。インテル® MKL ヘルプ索引との関連性が設定されます (図 10-4 を参照)。通常、各関数のリストに 1 つのリンクが表示されます。
  - 関数名の検索結果のリストが [Dynamic Help] の [Related Topics] ページに表示されます (図 10-5 を参照)。
3. リンクをクリックすると、該当するヘルプトピックが表示されます。

図 10-4 Eclipse IDE の F1 ヘルプ

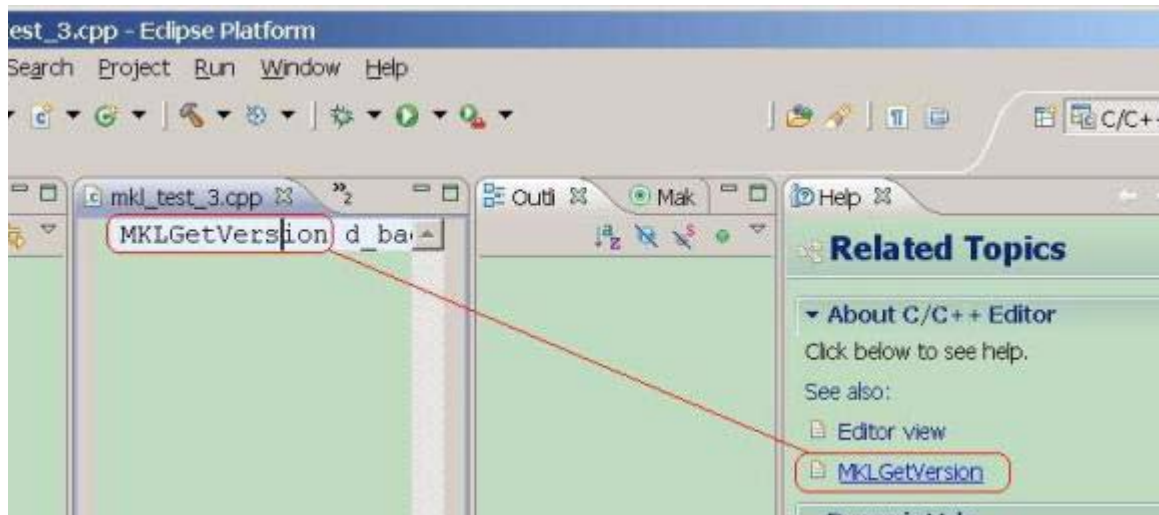
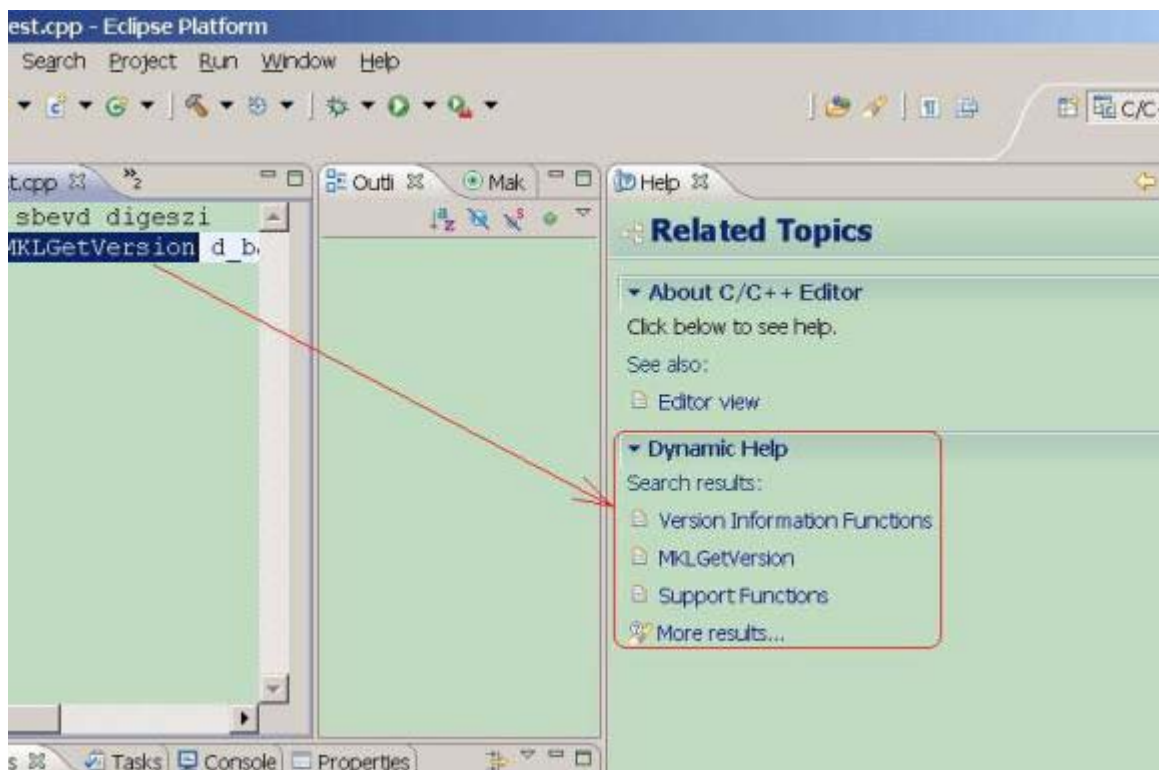


図 10-5 Eclipse IDE CDT での F1 ヘルプの検索



# LINPACK ベンチマークと MP LINPACK ベンチマーク

11

本章は、Intel® Optimized LINPACK Benchmark for the Linux\* OS (共有メモリーシステム用) と Intel® Optimized MP LINPACK Benchmark for Clusters (分散メモリーシステム用) について説明します。

## Intel® Optimized LINPACK Benchmark for Linux OS

Intel® Optimized LINPACK Benchmark は、LINPACK 1000 ベンチマークを一般化したものです。このベンチマークは、稠密な (real\*8) 連立線形方程式 ( $Ax=b$ ) を解き、因数分解して解くためにかかった時間を測定し、時間をパフォーマンス比率に変換して結果の精度をテストします。一般化により、1000 を超える方程式 ( $M$ ) を解くことができます。正確な結果が得られるように、部分的なピボット演算を使用しています。

このベンチマークは、コンパイルされたコードのみを対象とするベンチマークであるため、LINPACK 100 のパフォーマンスのレポートとして使用しないでください。これは単一プラットフォーム上で実行する共有メモリー (SMP) 実装です。このベンチマークと以下の項目を混同しないでください。

- MP LINPACK - 同じベンチマークの分散メモリーバージョン。
- LINPACK - LAPACK ライブラリーで拡張されたライブラリー。

インテルでは、HPL (High Performance Linpack) ベンチマークよりも簡単にインテル® プロセッサ・ベースのシステムで高い LINPACK ベンチマーク結果が得られる LINPACK ベンチマークの最適化バージョンを提供しています。SMP マシンのベンチマークには、このパッケージを使用してください。

このソフトウェアの詳細は、<http://www.intel.com/software/products/> (英語) を参照してください。

## 内容

Intel® Optimized LINPACK Benchmark for Linux には、以下のファイルが含まれています。ファイルは、インテル® MKL ディレクトリーの `/benchmarks/linpack/` サブディレクトリーにあります (「[表 3-1](#)」を参照)。

表 11-1 LINPACK Benchmark の内容

| ./benchmarks/linpack/ |                                                                                                    |
|-----------------------|----------------------------------------------------------------------------------------------------|
| linpack_itanium       | インテル® Itanium® プロセッサ・ベースのシステム用 64 ビット・プログラム                                                        |
| linpack_xeon32        | ストリーミング SIMD 拡張命令 3 (SSE3) 対応 / 非対応インテル® Xeon® プロセッサまたはインテル® Xeon® プロセッサ MP ベースのシステム用 32 ビット・プログラム |
| linpack_xeon64        | インテル® 64 アーキテクチャー対応インテル® Xeon® プロセッサ・ベースのシステム用 64 ビット・プログラム                                        |

表 11-1 LINPACK Benchmark の内容 ( 続き )

| ./benchmarks/linpack/ |                                                                                         |
|-----------------------|-----------------------------------------------------------------------------------------|
| runme_itanium         | linpack_itanium 用に事前に定義された問題セットを実行するためのサンプル・シェル・スクリプト。OMP_NUM_THREADS は 8 プロセッサに設定されます。 |
| runme_xeon32          | linpack_xeon32 用に事前に定義された問題セットを実行するためのサンプル・シェル・スクリプト。OMP_NUM_THREADS は 2 プロセッサに設定されます。  |
| runme_xeon64          | linpack_xeon64 用に事前に定義された問題セットを実行するためのサンプル・シェル・スクリプト。OMP_NUM_THREADS は 4 プロセッサに設定されます。  |
| lininput_itanium      | runme_itanium スクリプト用に事前に定義された問題の入力ファイル。                                                 |
| lininput_xeon32       | runme_xeon32 スクリプト用に事前に定義された問題の入力ファイル。                                                  |
| lininput_xeon64       | runme_xeon64 スクリプト用に事前に定義された問題の入力ファイル。                                                  |
| lin_itanium.txt       | runme_itanium スクリプトを実行した結果。                                                             |
| lin_xeon32.txt        | runme_xeon32 スクリプトを実行した結果。                                                              |
| lin_xeon64.txt        | runme_xeon64 スクリプトを実行した結果。                                                              |
| help.lpk              | 標準ヘルプファイル。                                                                              |
| xhelp.lpk             | 拡張ヘルプファイル。                                                                              |

## ソフトウェアの実行

指定したシステムで事前に定義されたサンプル問題サイズの結果を得るには、次のいずれかのコマンドを入力します。

```
./runme_itanium
./runme_xeon32
./runme_xeon64
```

ほかの問題サイズでソフトウェアを実行する方法は、プログラムに含まれている拡張ヘルプを参照してください。拡張ヘルプは、以下のように `-e` オプションを指定してプログラムを実行すると表示されます。

```
./xlinpack_itanium -e
./xlinpack_xeon32 -e
./xlinpack_xeon64 -e
```

データ入力ファイル `lininput_itanium`、`lininput_xeon32`、および `lininput_xeon64` は、単なる例として提供されています。プロセッサ数やメモリー容量が異なるシステムでは入力ファイルを変更する必要があります。入力ファイルを変更する適切な方法は、拡張ヘルプを参照してください。

各入力ファイルでは、少なくとも以下のメモリー容量が必要です。

```
lininput_itanium    16GB
lininput_xeon32     2GB
lininput_xeon64     16GB
```

システムのメモリー容量が上記のデータ入力ファイルに必要なメモリー容量よりも少ない場合、拡張ヘルプで説明されているように、既存のデータ入力ファイルを編集するか、新しいデータ入力ファイルを作成する必要があります。

各サンプルスクリプトでは、`OMP_NUM_THREADS` 環境変数を使用してターゲットのプロセッサ数を設定します。異なる物理プロセッサ数でパフォーマンスを最適化するには、該当する行を適切な値に変更してください。スレッド数を設定しないで Intel® Optimized LINPACK Benchmark を実



行すると、OS に従ってデフォルトのコア数が設定されます。この環境変数の設定は、`runme` \* サンプルスクリプトで行われています。設定が使用している環境と一致しない場合、スクリプトを編集してください。

## 既知の制限事項

Intel® Optimized LINPACK Benchmark for Linux には、以下の既知の制限があります。

- Intel® Optimized LINPACK Benchmark は、複数のプロセッサを使用して効率的にスレッド化されます。このため、ハイパースレッディング・テクノロジー対応のマルチプロセッサ・システムで最適なパフォーマンスを得るには、オペレーティング・システムが物理プロセッサにのみスレッドを割り当てられるように、ハイパースレッディング・テクノロジーを無効にしてください。
- 不完全なデータ入力ファイルが指定されると、バイナリはハングアップするか失敗します。正しいデータ入力ファイルの作成方法は、データ入力ファイルのサンプルまたは拡張ヘルプを参照してください。

## Intel® Optimized MP LINPACK Benchmark for Clusters

Intel® Optimized MP LINPACK Benchmark for Clusters は、テネシー大学ノックスビル校 (UTK) の Innovative Computing Laboratories (ICL) が提供している HPL 2.0 をベースに修正、追加したものです。この Intel® Optimized MP LINPACK Benchmark for Clusters は、Top 500 (<http://www.top500.org> を参照) の実行に使用することができます。ベンチマークを使用するには、HPL ディストリビューションと使用法について熟知する必要があります。Intel® Optimized MP LINPACK Benchmark for Clusters は、HPL をより便利に使用できるように、追加の拡張とバグフィックスが行われています。パフォーマンスを向上する Intel® MPI 設定についても説明します。

`./benchmarks/mp_linpack` ディレクトリには、長時間の実行における検索時間を最小限に抑えるための手法が加えられています。

Intel® Optimized MP LINPACK Benchmark for Clusters は、HPL コードによる Massively Parallel MP LINPACK (LINPACK の超並列対応版) ベンチマークの実装です。このベンチマークは、ランダムで稠密な ( $\text{real}^*8$ ) 連立線形方程式 ( $Ax=b$ ) を解き、因数分解して解くためにかかった時間を測定し、時間をパフォーマンス比率に変換して結果の精度をテストします。メモリーに収まる任意のサイズ ( $N$ ) の連立方程式を解くことができます。ベンチマークは、正確な結果が得られるように、完全な行ピボット演算を使用しています。

Intel® Optimized MP LINPACK Benchmark for Clusters は、分散メモリーマシンで使用します。共有メモリーマシンでは、Intel® Optimized LINPACK Benchmark を使用してください。

インテルでは、HPL ベンチマークよりも簡単にインテル® プロセッサ・ベースのシステムで高い LINPACK ベンチマーク結果が得られる LINPACK ベンチマークの最適化バージョンを提供しています。クラスターのベンチマークには、Intel® Optimized MP LINPACK Benchmark を使用してください。事前構築バイナリを使用するには、クラスターに Intel® MPI 3.x をインストールする必要があります。Intel® MPI のランタイムバージョンは、[www.intel.com/software/products/cluster](http://www.intel.com/software/products/cluster) からダウンロードできます。

このパッケージには、テネシー大学ノックスビル校の Innovative Computing Laboratories (ICL) で開発されたソフトウェアが含まれていますが、これはテネシー大学や ICL が本製品を推奨あるいは販促していることを意味するわけではありません。HPL 2.0 は特定の条件の下で再配布することができますが、このパッケージは Intel® MKL の使用許諾契約書に基づきます。

Intel® MKL の新しいバージョンでは、ハイブリッド・ビルドと呼ばれる新しい機能が MP LINPACK に追加されました。以前のバージョンも引き続きサポートしています。「ハイブリッド」とは、OpenMP\*/MPI を組み合わせた並列処理を活用するために追加された特別な最適化のことを指します。ノードあたり 1 つの MPI プロセスを使用して、OpenMP によるさらに高度な並列処理を行うには、ハイブリッド・ビルドを使用してください。一般に、ハイブリッド・ビルドはコアあたりの MPI プロセスの数が 1 未満の場合に適しています。MPI で並列処理を行い、コアあたり 1 つの MPI プロセスを使用する場合は、非ハイブリッド・ビルドを使用してください。

特定の事前構築ハイブリッド・ライブラリーに加えて、インテル® MKL では、OpenMP の最適化を活用する事前構築ハイブリッド・ライブラリーをいくつか提供します。



**注:** 事前構築ハイブリッド・ライブラリーは、インテル® MPI でのみ使用できません。

インテル® MPI 以外の MPI バージョンを使用する場合は、提供されている MP LINPACK ソースを使用してください。ソースからハイブリッド・モードで利用できる非ハイブリッド・バージョンをビルドすることもできますが、その場合、ハイブリッド・バージョンに追加される最適化の一部は含まれません。

提供されるソース・コード・メイクファイルのデフォルトは、非ハイブリッド・ビルドです。場合によっては、ハイブリッド・モードの利用が必要なことがあります。バージョンを選択できる場合、非ハイブリッド・コードのほうが高速です。ただし、将来のリリースで変更される可能性があります。非ハイブリッド・コードをハイブリッド・モードで利用するには、インテル® MKL BLAS のスレッド化バージョンを使用して、スレッドセーフな MPI とリンクし、MPI\_init\_thread() 関数を呼び出して、MPI がスレッドセーフである必要があることを示します。

インテル® MKL は、インテル® MPI ライブラリーに対してダイナミックにリンクされた事前構築バイナリーも提供しています。



**注:** スタティックにリンクされた事前構築バイナリーとダイナミックにリンクされた事前構築バイナリーのパフォーマンスは異なります。パフォーマンスは、使用するインテル® MPI のバージョンに依存します。インテル® MPI の特定のバージョンに対してスタティックにリンクしたバイナリーをビルドすることができます。

## 内容

Intel® Optimized MP LINPACK Benchmark for Clusters (MP LINPACK Benchmark) には、HPL 2.0 ディストリビューションとその修正が含まれています。ファイルの一覧は、[表 11-2](#) を参照してください。ファイルは、インテル® MKL ディレクトリーの ./benchmarks/mp\_linpack/ サブディレクトリーにあります (「[表 3-1](#)」を参照)。

**表 11-2 MP LINPACK Benchmark の内容**

| ./benchmarks/mp_linpack/   |                                                                                                            |
|----------------------------|------------------------------------------------------------------------------------------------------------|
| testing/ptest/HPL_pctest.c | HPL 2.0 コードに ASYUGO2_DISPLAY (詳細は、「 <a href="#">新機能</a> 」セクションを参照) で DGEMM 情報がキャプチャーされた場合に情報を表示する修正を加えたもの。 |
| src/blas/HPL_dgemm.c       | HPL 2.0 コードに ASYUGO2_DISPLAY で指定された場合に DGEMM 情報をキャプチャーする修正を加えたもの。                                          |
| src/grid/HPL_grid_init.c   | HPL 2.0 コードに HPL 2.0 がない追加のグリッド試験を行う修正を加えたもの。                                                              |
| src/pgesv/HPL_pdgesvK2.c   | HPL 2.0 コードに ASYUGO および ENDEARLY の修正を加えたもの。                                                                |
| src/pgesv/HPL_pdgesv0.c    | HPL 2.0 コードに ASYUGO、ASYUGO2、および ENDEARLY の修正を加えたもの。                                                        |

表 11-2 MP LINPACK Benchmark の内容 ( 続き )

|                                                       |                                                                                 |
|-------------------------------------------------------|---------------------------------------------------------------------------------|
| ./benchmarks/mp_linpack/                              |                                                                                 |
| testing/ptest/HPL.dat                                 | HPL 2.0 のサンプル HPL.dat を修正したもの。                                                  |
| Make.ia32                                             | (新規) IA-32 アーキテクチャー対応プロセッサ・ベースの Linux システム用のサンプル・アーキテクチャー・メイクファイル。              |
| Make.em64t                                            | (新規) インテル®64 アーキテクチャー対応プロセッサ・ベースの Linux システム用のサンプル・アーキテクチャー・メイクファイル。            |
| Make.ipf                                              | (新規) IA-64 アーキテクチャー対応プロセッサ・ベースの Linux システム用のサンプル・アーキテクチャー・メイクファイル。              |
| HPL.dat                                               | testing/ptest/HPL.dat のコピー。                                                     |
| 次の 6 つのファイルは、簡単なパフォーマンス・テストに利用できる、事前に構築された実行可能ファイルです。 |                                                                                 |
| bin_intel/ia32/xhpl_ia32                              | (新規) IA-32 アーキテクチャーの Linux OS 用の事前構築ハイブリッド・バイナリー。インテル®MPI 3.2 に対してスタティックにリンク。   |
| bin_intel/ia32/xhpl_ia32_dynamic                      | (新規) IA-32 アーキテクチャーの Linux OS 用の事前構築ハイブリッド・バイナリー。インテル®MPI 3.2 に対してダイナミックにリンク。   |
| bin_intel/em64t/xhpl_em64t                            | (新規) インテル®64 アーキテクチャーの Linux OS 用の事前構築ハイブリッド・バイナリー。インテル®MPI 3.2 に対してスタティックにリンク。 |
| bin_intel/em64t/xhpl_em64t_dynamic                    | (新規) インテル®64 アーキテクチャーの Linux OS 用の事前構築ハイブリッド・バイナリー。インテル®MPI 3.2 に対してダイナミックにリンク。 |
| bin_intel/ipf/xhpl_ipf                                | (新規) IA-64 アーキテクチャーの Linux OS 用の事前構築ハイブリッド・バイナリー。インテル®MPI 3.2 に対してスタティックにリンク。   |
| bin_intel/ipf/xhpl_ipf_dynamic                        | (新規) IA-64 アーキテクチャーの Linux OS 用の事前構築ハイブリッド・バイナリー。インテル®MPI 3.2 に対してダイナミックにリンク。   |
| 次の 6 つのファイルは、事前に構築されたハイブリッドの実行可能ファイルです。               |                                                                                 |
| bin_intel/ia32/xhpl_hybrid_ia32                       | (新規) IA-32 アーキテクチャーの Linux OS 用の事前構築ハイブリッド・バイナリー。インテル®MPI 3.2 に対してスタティックにリンク。   |
| bin_intel/ia32/xhpl_hybrid_ia32_dynamic               | (新規) IA-32 アーキテクチャーの Linux OS 用の事前構築ハイブリッド・バイナリー。インテル®MPI 3.2 に対してダイナミックにリンク。   |
| bin_intel/em64t/xhpl_hybrid_em64t                     | (新規) インテル®64 アーキテクチャーの Linux OS 用の事前構築ハイブリッド・バイナリー。インテル®MPI 3.2 に対してスタティックにリンク。 |
| bin_intel/em64t/xhpl_hybrid_em64t_dynamic             | (新規) インテル®64 アーキテクチャーの Linux OS 用の事前構築ハイブリッド・バイナリー。インテル®MPI 3.2 に対してダイナミックにリンク。 |
| bin_intel/ipf/xhpl_hybrid_ipf                         | (新規) IA-64 アーキテクチャーの Linux OS 用の事前構築ハイブリッド・バイナリー。インテル®MPI 3.2 に対してスタティックにリンク。   |
| bin_intel/ipf/xhpl_hybrid_ipf_dynamic                 | (新規) IA-64 アーキテクチャーの Linux OS 用の事前構築ハイブリッド・バイナリー。インテル®MPI 3.2 に対してダイナミックにリンク。   |



表 11-2 MP LINPACK Benchmark の内容 ( 続き )

|                                            |                                                                                   |
|--------------------------------------------|-----------------------------------------------------------------------------------|
| ./benchmarks/mp_linpack/                   |                                                                                   |
| 次の 3 つのファイルは、事前構築ライブラリーです。                 |                                                                                   |
| lib_hybrid/32/libhpl_hybrid.a              | (新規) IA-32 アーキテクチャー、インテル® MPI 3.2 用の MP LINPACK ハイブリッド・バージョン事前構築ライブラリー。           |
| lib_hybrid/em64t/libhpl_hybrid.a           | (新規) インテル® 64 アーキテクチャー、インテル® MPI 3.2 用の MP LINPACK ハイブリッド・バージョン事前構築ライブラリー。        |
| lib_hybrid/64/libhpl_hybrid.a              | (新規) IA-64 アーキテクチャー、インテル® MPI 3.2 用の MP LINPACK ハイブリッド・バージョン事前構築ライブラリー。           |
| 次の 18 のファイルは、実行スクリプトです。                    |                                                                                   |
| bin_intel/ia32/runme_ia32                  | (新規) IA-32 アーキテクチャー、インテル® MPI 3.2 に対してスタティックにリンクされたピュア MPI バイナリー用のサンプル実行スクリプト。    |
| bin_intel/ia32/runme_ia32_dynamic          | (新規) IA-32 アーキテクチャー、インテル® MPI 3.2 に対してダイナミックにリンクされたピュア MPI バイナリー用のサンプル実行スクリプト。    |
| bin_intel/ia32/HPL_serial.dat              | (新規) IA-32 アーキテクチャー、ピュア MPI バイナリー用の MP LINPACK ベンチマーク入力ファイルのサンプル。                 |
| bin_intel/ia32/runme_hybrid_ia32           | (新規) IA-32 アーキテクチャー、インテル® MPI 3.2 に対してスタティックにリンクされたハイブリッド・バイナリー用のサンプル実行スクリプト。     |
| bin_intel/ia32/runme_hybrid_ia32_dynamic   | (新規) IA-32 アーキテクチャー、インテル® MPI 3.2 に対してダイナミックにリンクされたハイブリッド・バイナリー用のサンプル実行スクリプト。     |
| bin_intel/ia32/HPL_hybrid.dat              | (新規) IA-32 アーキテクチャー、ハイブリッド・バイナリー用の MP LINPACK ベンチマーク入力ファイルのサンプル。                  |
| bin_intel/em64t/runme_em64t                | (新規) インテル® 64 アーキテクチャー、インテル® MPI 3.2 に対してスタティックにリンクされたピュア MPI バイナリー用のサンプル実行スクリプト。 |
| bin_intel/em64t/runme_em64t_dynamic        | (新規) インテル® 64 アーキテクチャー、インテル® MPI 3.2 に対してダイナミックにリンクされたピュア MPI バイナリー用のサンプル実行スクリプト。 |
| bin_intel/em64t/HPL_serial.dat             | (新規) インテル® 64 アーキテクチャー、ピュア MPI バイナリー用の MP LINPACK ベンチマーク入力ファイルのサンプル。              |
| bin_intel/em64t/runme_hybrid_em64t         | (新規) インテル® 64 アーキテクチャー、インテル® MPI 3.2 に対してスタティックにリンクされたハイブリッド・バイナリー用のサンプル実行スクリプト。  |
| bin_intel/em64t/runme_hybrid_em64t_dynamic | (新規) インテル® 64 アーキテクチャー、インテル® MPI 3.2 に対してダイナミックにリンクされたハイブリッド・バイナリー用のサンプル実行スクリプト。  |
| bin_intel/em64t/HPL_hybrid.dat             | (新規) インテル® 64 アーキテクチャー、ハイブリッド・バイナリー用の MP LINPACK ベンチマーク入力ファイルのサンプル。               |
| bin_intel/ipf/runme_ia64                   | (新規) IA-64 アーキテクチャー、インテル® MPI 3.2 に対してスタティックにリンクされたピュア MPI バイナリー用のサンプル実行スクリプト。    |
| bin_intel/ipf/runme_ia64_dynamic           | (新規) IA-64 アーキテクチャー、インテル® MPI 3.2 に対してダイナミックにリンクされたピュア MPI バイナリー用のサンプル実行スクリプト。    |
| bin_intel/ipf/HPL_serial.dat               | (新規) IA-64 アーキテクチャー、ピュア MPI バイナリー用の MP LINPACK ベンチマーク入力ファイルのサンプル。                 |

表 11-2 MP LINPACK Benchmark の内容 ( 続き )

|                                         |                                                                                 |
|-----------------------------------------|---------------------------------------------------------------------------------|
| ./benchmarks/mp_linpack/                |                                                                                 |
| bin_intel/ipf/runme_hybrid_ia64         | ( 新規 ) IA-64 アーキテクチャー、インテル® MPI 3.2 に対してスタティックにリンクされたハイブリッド・バイナリー用のサンプル実行スクリプト。 |
| bin_intel/ipf/runme_hybrid_ia64_dynamic | ( 新規 ) IA-64 アーキテクチャー、インテル® MPI 3.2 に対してダイナミックにリンクされたハイブリッド・バイナリー用のサンプル実行スクリプト。 |
| bin_intel/ipf/HPL_hybrid.dat            | ( 新規 ) IA-64 アーキテクチャー、ハイブリッド・バイナリー用の MP LINPACK ベンチマーク入力ファイルのサンプル。              |
| nodeperf.c                              | ( 新規 ) クラスターの DGEMM 速度をテストするサンプル・ユーティリティ。                                       |

## MP LINPACK の構築

サンプル・アーキテクチャー・メイクファイルがいくつか用意されています。使用している構成に合わせて、これらのファイルを以下のように編集することを推奨します。

- TOPdir を MP LINPACK が含まれているディレクトリーに設定します。
- MPI 変数、MPdir、MPinc、および MPlib を設定します。
- インテル® MKL の場所と使用するファイルの場所を指定します (LAdir、LAinc、LAlib)。
- コンパイラーおよびコンパイラー/リンカーオプションを調整します。
- make コマンドで次のようにバージョン・パラメーターを設定して、ビルドする MP LINPACK のバージョン (ハイブリッドまたは非ハイブリッド) を指定します。  
make arch=em64t version=hybrid install

インテル® 64 アーキテクチャー・ベースの Linux システムなど、一部のサンプルケースでは、メイクファイルには一般的な値が含まれています。しかし、HPL の構築についてよく理解した上で、これらの変数に適切な値を設定する必要があります。

## 新機能

ツールセットは HPL 2.0 ディストリビューションと基本的に同一です。いくつかの変更は、オプションで指定してコンパイルしない限り無効です。以下の新機能があります。

**ASYOUGO:** 実行の進行に伴い、非侵入型のパフォーマンス情報を提供します。出力はわずかで、この情報はパフォーマンスに影響しません。情報を提供しなくても、多くの実行が長時間行われるため、特に役立つ機能です。

**ASYOUGO2:** すべての DGEMM を傍受するため、わずかに侵入する付加的なパフォーマンス情報を提供します。

**ASYOUGO2\_DISPLAY:** 実行内部の有効なすべての DGEMM のパフォーマンスを表示します。

**ENDEARLY:** いくつかのパフォーマンスのヒントを表示し、実行を早く終了します。

**FASTSWAP:** HPL のコードに LAPACK で最適化された DLASWP を挿入します。インテル® Itanium® プロセッサで役立ちます。この機能を使用して試験することで最良の結果を判断できます。

**HYBRID:** MP LINPACK のハイブリッド OpenMP/MPI モードを有効にして、スレッド化されたインテル® MKL と MP LINPACK の事前構築ハイブリッド・ライブラリーを使用できるようにします。



**警告:** このオプションは、インテル® コンパイラーとインテル® MPI ライブラリー・バージョン 3.1 以降を使用している場合にのみ使用してください。コンパイラーはバージョン 10.0 以降を使用することを推奨します。

## クラスタのベンチマーク

クラスタのベンチマークを行うには、以下の手順に従ってください (一部の手順はオプションです)。ステップ 3. と 4. の繰り返しには特に注意してください。クラスタが最高のパフォーマンスを得られる HPL パラメーター (HPL.dat で指定) の検索が繰り返されます。

1. HPL をインストールして、HPL がすべてのノードで機能していることを確認します。

2. ディストリビューションに含まれている `nodeperf.c` を実行し、すべてのノードで DGEMM のパフォーマンスを確認します。

MPI およびインテル® MKL を使用して `nodeperf.c` をコンパイルします。次に例を示します。

```
mpiicc -O3 nodeperf.c -L$MKL_PATH $MKL_PATH/libmkl_intel_lp64.a \
-Wl,--start-group $MKL_PATH/libmkl_sequential.a \
$MKL_PATH/libmkl_core.a -Wl,--end-group -lpthread .
```

すべてのノードで `nodeperf.c` を起動することは、非常に大規模なクラスタでは特に有用です。実際、ほかよりも 5% 遅く実行される特定のノード (例えば、738) にジョブが残ることがあります。MP LINPACK は、最も遅いノードに合わせて実行します。この場合、`nodeperf` を使用すると、悪いノードを見つけるためにクラスタでさまざまな小さな MP LINPACK 実行を行うことなく、潜在的な問題のスポットを素早く識別することができます。HPL を何度か実行した後、ゾンビプロセスが存在し、`nodeperf` が遅いノードを検索することは一般的です。検索はすべてのノードに対して 1 つずつ行われ、DGEMM のパフォーマンスに続けてホスト識別子が報告されます。このため、DGEMM のパフォーマンスが高いほど、ノードの実行が高速であったこととなります。

3. 使用するクラスタに合わせて HPL.dat を編集します。

詳細は、HPL のドキュメントを参照してください。ただし、少なくとも 4 つのノードで試すようにしてください。

4. ASYUGO、ASYUGO2 または ENDEARLY などのコンパイラー・オプションを使用して HPL を実行します (これらのオプションを使用することで、パフォーマンスに対する考察が、HPL から通常得られるよりも早く得ることができます)。

実行するときは、以下の推奨事項に従ってください。

- 検索時間を短縮するため、MP LINPACK (HPL のパッチ済みバージョン) を使用してください。

HPL のパッチ済みバージョンを使用することでパフォーマンスに影響を与えないようにする必要があります。このため、パフォーマンスに影響を与える可能性のある機能は、MP LINPACK では (この後で説明するように) コンパイラー・オプションとして提供されています。「[検索時間を短縮するためのオプション](#)」セクションで説明されている新しいオプションを使用しない場合、これらの変更は無効になります。拡張の主な目的は、ソリューションを見つけるための支援です。

HPL では、多くの異なるパラメーターの検索に長い時間がかかります。MP LINPACK では、最適な数を得ることが目標です。

入力が固定でない場合、大きなパラメーター空間を検索する必要があります。実際、あらゆる入力の全数検索は、強力なクラスタでもかなりの時間がかかります。

この HPL のパッチ済みバージョンは、オプションで実行中のパフォーマンスの情報を出力します。また、指定された場合、実行を終了します。

- `-DENDEARLY -DASYOUGO2` ([「検索時間を短縮するためのオプション」](#) セクションを参照) を使用してコンパイルし、負のしきい値を使用して時間を短縮できます (Top 500 に提出する最終的な実行で負のしきい値を使用しないでください)。HPL 2.0 入力ファイル `HPL.dat` の 13 行でしきい値を設定することができます。
  - 問題の完了まで実行する場合は、`-DASYOUGO` ([「検索時間を短縮するためのオプション」](#) を参照) を使用してください。
5. 迅速なパフォーマンス・フィードバックを使用し、最良のパフォーマンスが得られるまでステップ 3 と 4 を繰り返します。

### 検索時間を短縮するためのオプション

多くのノードで問題の完了まで実行すると、長い時間がかかります。MP LINPACK の検索空間も巨大です。実行する問題のサイズのみでなく、ブロックサイズの数、グリッドレイアウト、ステップの先読み、異なる因数分解方法の使用なども影響します。以前に得られた最良のパフォーマンスよりも、0.01% 遅くなったことを発見するためだけに大きな問題を最後まで実行しても、膨大な時間の浪費になります。

検索時間が短くなる可能性のあるオプションは 3 つあります。

- `-DASYOUGO`
- `-DENDEARLY`
- `-DASYOUGO2`

限界パフォーマンスに影響を与えるため、`-DASYOUGO2` は慎重に使用してください。DGEMM の内部パフォーマンスを参照するには、`-DASYOUGO2` および `-DASYOUGO2_DISPLAY` を使用してコンパイルします。これらのオプションを使用することでパフォーマンスは約 0.2% 損なわれますが、多くの有用な DGEMM 情報が提供されます。

以前の HPL に戻すには、これらのオプションを定義せずに最初から再コンパイルします (`"make arch=<arch> clean_arch_all"` を実行してください)。

**-DASYOUGO:** 実行の進行に伴い、パフォーマンス・データが提供されます。LU 分解が発生するため、パフォーマンスは常に開始時は高く、徐々に低くなります。ASYOUGO パフォーマンス評価は通常 (LU 分解により遅くなるため) 高めに評価されますが、問題が進行するにつれて、より正確になります。ステップの先読みが多いほど、最初の数値は正確でなくなります。ASYOUGO は MP LINPACK が実行する LU 分解を含めて評価しようとするため、実際に達成された DGEMM パフォーマンスを測定する ASYOUGO2 と比較して高めに評価されます。ASYOUGO の出力は ASYOUGO2 が提供する情報のサブセットであることに注意してください。このため、出力の詳細は、`-DASYOUGO2` オプションの説明を参照してください。

**-DENDEARLY:** いくつかのステップの後に問題を終了します。このため、モニターをせずに 10 から 20 程度の HPL をセットアップして実行し、最も速かった HPL のみを完了させることができます。`-DENDEARLY` は `-DASYOUGO` を想定します。両方を定義しても問題はありますが、その必要はありません。問題が早く終了するため、`DENDEARLY` をテストする場合は、`HPL.dat` のしきい値を負の数に設定することを推奨します。問題が早く終了する場合に残差を確認する意味はありません。`-DENDEARLY` を使用する場合、`-DASYOUGO2` を使用してコンパイルしたほうが良い場合もあります。

`-DENDEARLY` の使用に関する注意:

- `-DENDEARLY` は、ブロックサイズで DGEMM の反復を数回行った後、問題を停止します (ブロックサイズが大きいほど、得られる情報も多くなります)。5 または 6 のアップデートのみ出力します (`-DASYOUGO` は問題を完了する前に 46 程度の出力要素を出力します)。
- `-DASYOUGO` と `-DENDEARLY` のパフォーマンスは常に 1 つの速度で開始され、ゆっくり増加した後、終了に向かって速度が落ちます (LU 分解が行われるため)。`-DENDEARLY` は通常、速度が落ちる前に終了します。
- `-DENDEARLY` は、HPL エラーとともに問題を早く終了します。問題は完了していないため、見つからない (誤った) 残差を無視する必要があります。しかし、初期のパフォーマンスは確認できるため、パフォーマンスが良い場合は `-DENDEARLY` を指定しないで問題を最後まで実行します。エラーチェックを回避するには、`HPL.dat` に含まれている HPL のしきい値パラメーターを負の数にしてください。

- DENSEARLY は早く終了するため、HPL は問題が完了したと解釈し、問題が完了したものととして Gflop 評価を計算します。この誤った高い評価は無視してください。
- より大きな問題では、精度はより高くなります。-DENSEARLY が返す最後のアップデートは、問題を完了まで実行したときのアップデートに近くなります。-DENSEARLY は、小さな問題では近似が不十分です。この理由により、ENSEARLY は ASYUGO2 と組み合わせることを推奨します。ASYUGO2 は実際の DGEMM パフォーマンスを報告するため、開始した問題への近似がより近くなります。

インテル® コンパイラーを使用した場合、最もよく知られているインテル® Itanium® プロセッサ用のコンパイルオプションは、次のようになります。

```
-O2 -ipo -ipo_obj -ftz -IPF_fltacc -IPF_fma -unroll -w -tpp2
```

**-DASYUGO2:** 詳細な単一ノードの DGEMM パフォーマンス情報を提供します。すべての DGEMM 呼び出しをキャプチャーして (Fortran BLAS を使用している場合)、データを記録します。このため、ルーチンには侵入型のオーバーヘッドが存在します。非侵入型の -DASYUGO とは異なり、-DASYUGO2 は、パフォーマンスをモニターするため、DGEMM の呼び出しごとに中断します。たとえパフォーマンスへの影響が 0.1% 未満であることがわかっている場合でも、大きな問題ではこのオーバーヘッドに注意する必要があります。

次に、ASYUGO2 出力のサンプルを示します (最初の 3 つの非侵入数は ASYUGO および ENDEARLY の説明を参照してください)。

```
Col=001280 Fract=0.050 Mflops=42454.99 (DT= 9.5 DF= 34.1 DMF=38322.78)
```

問題サイズは  $N=16000$  で、ブロックサイズは 128 でした。10 ブロック、つまり 1280 列を処理した後、出力は画面に送られました。ここで、完了した列の小数は  $1280/16000=0.08$  です。行列分解により、最大 40 の出力結果がさまざまな場所に出力されます: fractions

```
0.005 0.010 0.015 0.020 0.025 0.030 0.035 0.040 0.045 0.050 0.055 0.060 0.065 0.070 0.075
0.080 0.085 0.090 0.095 0.100 0.105 0.110 0.115 0.120 0.125 0.130 0.135 0.140 0.145 0.150
0.155 0.160 0.165 0.170 0.175 0.180 0.185 0.190 0.195 0.200 0.205 0.210 0.215 0.220 0.225
0.230 0.235 0.240 0.245 0.250 0.255 0.260 0.265 0.270 0.275 0.280 0.285 0.290 0.295 0.300
0.305 0.310 0.315 0.320 0.325 0.330 0.335 0.340 0.345 0.350 0.355 0.360 0.365 0.370 0.375
0.380 0.385 0.390 0.395 0.400 0.405 0.410 0.415 0.420 0.425 0.430 0.435 0.440 0.445 0.450
0.455 0.460 0.465 0.470 0.475 0.480 0.485 0.490 0.495 0.515 0.535 0.555 0.575 0.595 0.615
0.635 0.655 0.675 0.695 0.795 0.895.
```

しかし、ここでは比較のために問題サイズが非常に小さくブロック数が非常に大きいため、0.045 の値を出力するとすぐに、列の小数である 0.08 が見つかりました。非常に大きな問題では、小数の数はより正確になります。上記の 112 を超える数は出力できません。このため、アップデートの数はより小さな問題では 112 よりも少なく、より大きな問題では正確に 112 になります。

Mflops は、LU 分解が完了した 1280 列に基づく評価です。しかし、ステップの先読みが行われると、出力が行われるときに作業が実際に完了していない場合があります。しかし、これは同一の実行を比較するためには良い評価です。

括弧で囲まれている 3 つの数は、侵入型 ASYUGO2 のアドインです。DT は、プロセッサ 0 が DGEMM で費やした合計時間 (単位は秒) です。DF は、1 つのプロセッサによって DGEMM で実行された処理の数 (単位は 10 億) です。したがって、プロセッサ 0 の DGEMM のパフォーマンス (Gflops) は常に  $DF/DT$  になります。LU flops の数の代わりに DGEMM flops の数を基本として使用し、DMF を調べることで、実行のパフォーマンスの下限がわかります (Mflops はグローバル LU 時間を使用しますが、HPL のノード (0,0) のみは任意の出力を返すため、DGEMM flops は問題がノード間で平等に分散されているという仮定の下で計算されます)。

上記のパフォーマンス監視ツールを使用して異なる HPL.dat 入力データセットを比較する場合、LU を使用したときのパフォーマンス低下のパターンは、一部の入力データの影響を受けやすいことに注意してください。例えば、非常に小さな問題を実行した場合、初期値から終了値までのパフォーマンス低下は非常に急速です。より大きな問題では、パフォーマンス低下はゆるやかになるため、最初のいくつかのパフォーマンス値を使用して問題サイズの違い (例えば、7000000 と 701000) を評価しても安全です。パフォーマンス低下に影響を与える別の要因は、グリッドの次元 (P および Q) です。大きな問題では、P と Q が値でほぼ等しい場合、最初の数ステップからのパ

パフォーマンス低下が少なくなる傾向があります。ブロードキャスト型のような大量のパラメーターを利用するように変更することで、最終的なパフォーマンスに非常に近いパフォーマンスを最初の数ステップで特定することができます。

これらのツールを使用すると、さまざまな量のデータをテストすることが可能です。

# インテル® マス・カーネル・ ライブラリー言語 インターフェイスのサポート

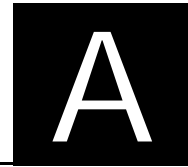


表 A-1 は各関数ドメイン用にインテル® マス・カーネル・ライブラリー (インテル® MKL) が提供する言語インターフェイスを、表 A-2 はそれぞれのヘッダーファイルを示しています。ただし、インテル® MKL ルーチンは混在言語プログラミングを使用してほかの言語から呼び出すこともできます。例えば、Fortran ルーチンを C/C++ から呼び出す方法は、「[混在言語プログラミングとインテル® MKL](#)」を参照してください。

表 A-1 言語インターフェイスのサポート

| 関数ドメイン                                                  | FORTRAN 77<br>インターフェイス | Fortran 90/95<br>インターフェイス | C/C++<br>インターフェイス |
|---------------------------------------------------------|------------------------|---------------------------|-------------------|
| Basic Linear Algebra Subprograms (BLAS)                 | ○                      | ○                         | CBLAS 利用          |
| BLAS 形式の拡張転置ルーチン                                        | ○                      |                           | ○                 |
| スパース BLAS レベル 1                                         | ○                      | ○                         | CBLAS 利用          |
| スパース BLAS レベル 2 およびレベル 3                                | ○                      | ○                         | ○                 |
| 連立線形方程式を解くための LAPACK ルーチン                               | ○                      | ○                         | †                 |
| 最小二乗問題、固有値ならびに特異値問題、<br>およびシルベスター式を解くための LAPACK<br>ルーチン | ○                      | ○                         | †                 |
| 補助 LAPACK ルーチン                                          | ○                      |                           | †                 |
| Parallel Basic Linear Algebra Subprograms (PBLAS)       | ○                      |                           |                   |
| ScaLAPACK ルーチン                                          | ○                      |                           | †                 |
| DSS/PARDISO* ソルバー                                       | ○                      | ○                         | ○                 |
| その他の直接法および反復法スパース・<br>ソルバー・ルーチン                         | ○                      | ○                         | ○                 |
| ベクトル・マス・ライブラリー (VML) 関数                                 | ○                      | ○                         | ○                 |
| ベクトル・スタティスティカル・ライブラリー<br>(VSL) 関数                       | ○                      | ○                         | ○                 |
| フーリエ変換関数 (FFT)                                          |                        | ○                         | ○                 |
| クラスター FFT 関数                                            |                        | ○                         | ○                 |
| 三角変換ルーチン                                                |                        | ○                         | ○                 |
| 高速ポアソン、ラプラス、およびヘルムホルツ・<br>ソルバー (ポアソン・ライブラリー) ルーチン       |                        | ○                         | ○                 |

表 A-1 言語インターフェイスのサポート ( 続き )

| 関数ドメイン                      | FORTRAN 77<br>インターフェイス | Fortran 90/95<br>インターフェイス | C/C++<br>インターフェイス |
|-----------------------------|------------------------|---------------------------|-------------------|
| 最適化 (Trust-Region) ソルバールーチン | ○                      | ○                         | ○                 |
| GMP* 数学関数                   |                        |                           | ○                 |
| サービسلーチン ( メモリー割り当てを含む )    |                        |                           | ○                 |

† 混在言語プログラミング呼び出しをサポート。それぞれのヘッダーファイルは、表 A-2 を参照してください。

表 A-2 は、すべてのインテル® MKL 関数ドメインで利用可能なヘッダーファイルを示しています。

表 A-2 インクルード・ファイル

| 関数ドメイン                   | インクルード・ファイル                        |                       |
|--------------------------|------------------------------------|-----------------------|
|                          | Fortran                            | C/C++                 |
| すべての関数ドメイン               | mkl.fi                             | mkl.h                 |
| BLAS ルーチン                | blas.f90<br>mkl_blas.fi            | mkl_blas.h            |
| BLAS 形式の拡張転置ルーチン         | mkl_trans.fi                       | mkl_trans.h           |
| BLAS に対する CBLAS インターフェイス |                                    | mkl_cblas.h           |
| スパース BLAS ルーチン           | mkl_spgblas.fi                     | mkl_spgblas.h         |
| LAPACK ルーチン              | lapack.f90<br>mkl_lapack.fi        | mkl_lapack.h          |
| ScaLAPACK ルーチン           |                                    | mkl_scalapack.h       |
| すべてのスパース・ソルバー・ルーチン       | mkl_solver.f90                     | mkl_solver.h          |
| ▪ PARDISO                | mkl_pardiso.f77<br>mkl_pardiso.f90 | mkl_pardiso.h         |
| ▪ DSS インターフェイス           | mkl_dss.f77<br>mkl_dss.f90         | mkl_dss.h             |
| ▪ RCI 反復ソルバー             | mkl_rci.fi                         | mkl_rci.h             |
| ▪ ILU 因数分解               |                                    |                       |
| 最適化ソルバールーチン              | mkl_rci.fi                         | mkl_rci.h             |
| ベクトル数学関数                 | mkl_vml.f77<br>mkl_vml.fi          | mkl_vml.h             |
| ベクトル・スタティスティカル関数         | mkl_vml.f77<br>mkl_vsl.fi          | mkl_vsl.h             |
| フーリエ変換関数                 | mkl_dfti.f90                       | mkl_dfti.h            |
| クラスターフーリエ変換関数            | mkl_cdft.f90                       | mkl_cdft.h            |
| 偏微分方程式サポートルーチン           |                                    |                       |
| ▪ 三角変換                   | mkl_trig_transforms.f90            | mkl_trig_transforms.h |
| ▪ ポアソンソルバー               | mkl_poisson.f90                    | mkl_poisson.h         |
| GMP インターフェイス             |                                    | mkl_gmp.h             |



表 A-2 インクルード・ファイル ( 続き )

| 関数ドメイン            | インクルード・ファイル |               |
|-------------------|-------------|---------------|
|                   | Fortran     | C/C++         |
| サービスルーチン          |             | mkl_service.h |
| メモリー割り当てルーチン      |             | i_malloc.h    |
| MKL サンプル・インターフェイス |             | mkl_example.h |

# サードパーティー・ インターフェイスのサポート



本付録では、インテル® マス・カーネル・ライブラリー (インテル® MKL) がサポートする特定のインターフェイスについて簡単に説明します。

## GMP\* 関数

インテル MKL に実装されている GMP 数学関数には、任意精度の整数演算が含まれています。これらの関数のインターフェイスは、GMP (GNU Multiple Precision) 演算ライブラリーと互換性があります。これらの関数の仕様は、<http://www.intel.com/software/products/mkl/docs/gnump/WebHelp/> (英語) を参照してください。

GMP ライブラリーを現在使用している場合、`mkl_gmp.h` をインクルードするようにプログラムの `INCLUDE` ステートメントを修正する必要があります。

## FFTW インターフェイスのサポート

インテル® MKL には、インテル® MKL フーリエ変換関数の呼び出しに使用される、FFTW インターフェイス ([www.fftw.org](http://www.fftw.org)) の上部構造である 2 つのラッパー・コレクションが用意されています。これらのコレクションはそれぞれ、FFTW バージョン 2.x と 3.x に対応しており、インテル® MKL バージョン 7.0 以降で利用できます。

これらのラッパーを使用することで、現在 FFTW を使用しているプログラムの開発者は、ソースコードを変更することなく、インテル® MKL フーリエ変換を使用してパフォーマンスを向上できます。ラッパーの使用についての詳細は、『インテル® MKL リファレンス・マニュアル』の「*FFTW to Intel® MKL Wrappers*」を参照してください。

# 索引

---

## A

Advanced Vector Extensions、命令ディスパッチ 6-10

## B

BLAS

    C から呼び出し 7-5

    Fortran-95 インターフェイス 7-2

## C

CBLAS 7-5

CBLAS、コードの例 7-8

C、LAPACK、BLAS、CBLAS の呼び出し 7-4

## E

Eclipse\* CDT

    インテル Web サイトの検索 10-2

    構成 4-2

Eclipse\* CDT、インテル (R) MKL ヘルプ 10-1

    状況依存 10-3

## F

FFT インターフェイス

    最適化基数 6-13

FFT 関数、データのアライメント 6-11

FFTW インターフェイスのサポート B-1

Fortran-95、LAPACK と BLAS のインターフェイス 7-2

## G

GMP (GNU\* Multiple Precision) 演算ライブラリー B-1

## H

HT テクノロジー、

    →ハイパースレッディング・テクノロジー

## I

ILP64 プログラミング、サポート 3-4

## J

Java\* の例 7-10

## L

LAPACK

    C から呼び出し 7-4

Fortran-95 インターフェイス 7-2

    圧縮ルーチンのパフォーマンス 6-11

LINPACK ベンチマーク 11-1

## M

MP LINPACK ベンチマーク 11-3

    ハイブリッド・バージョン 11-3

## O

OpenMP\*

    互換ランタイム・ライブラリー 3-3

    レガシー・ランタイム・ライブラリー 3-3

OpenMP\*、ランタイム・ライブラリー 5-3

## P

PARDISO\* OOC、構成ファイル 4-3

## R

RTL 7-3

## S

ScaLAPACK、リンク 9-1

## U

uBLAS、行列 - 行列乗算、インテル® MKL 関数に置換 7-8

## あ

アフィニティー・マスク 6-12

安定性、数値計算 8-1

## い

インストール、確認 2-3

インターフェイス・レイヤー 3-3

## か

開発環境の構成 4-1

    Eclipse\* CDT 4-2

カスタム共有オブジェクト 5-10

    関数のリストの指定 5-11

    構築 5-10

    メイクファイル・パラメーターの指定 5-10

環境変数、設定 4-1

## く

クラスター FFT、リンク 9-1  
クラスター・ソフトウェア 9-1  
    リンク構文 9-1  
    例のリンク 9-3

## け

計算レイヤー 3-3  
言語インターフェイスのサポート A-1  
    言語固有インターフェイス 7-1

## こ

構成ファイル、OOC DSS/PARDISO\* 4-3  
構成、開発環境 4-1  
構文  
    リンク、クラスター・ソフトウェア 9-1  
コーディング  
    混在言語の呼び出し 7-6  
    データのアライメント 8-1  
    パフォーマンスを向上する手法 6-11  
互換 OpenMP\* ランタイム・ライブラリー 3-3  
混在言語プログラミング 7-4  
コンパイラ依存の関数 7-3  
コンパイラ・サポート・ランタイム・ライブラリー 3-3

## さ

サポートするコンパイラ 2-4  
サポート、テクニカル 1-1

## し

状況依存ヘルプ、インテル (R) MKL、Eclipse\* CDT 10-3  
使用法 1-1

## す

数値計算の安定性 8-1  
スレッド化  
    インテル (R) MKL コントロール 6-7  
    環境変数と関数 6-7  
    競合の回避 6-3  
    →スレッド数  
スレッド数  
    設定する手法 6-3  
スレッド化レイヤー 3-3  
スレッド数  
    OpenMP\* 環境変数を使用した設定 6-4  
    インテル (R) MKL の選択、特定の場 6-8  
    クラスター用の設定 9-2  
    ランタイムの変更 6-4  
スレッド化  
    インテル® MKL の動作、特定の場 6-8  
スレッドの安全性、インテル (R) MKL 6-2

## て

ディスパッチ、AVX 命令 6-10  
ディレクトリー構造

高レベル 3-1

詳細 3-6

ドキュメント 3-15

データのアライメント 8-2

テクニカルサポート 1-1

## と

ドキュメント 3-15  
    インテル (R) MKL、Eclipse\* IDE で表示 10-1

## は

ハイパースレッディング・テクノロジー、構成のヒント 6-11  
ハイブリッド、バージョン、MP LINPACK 11-3  
パフォーマンス 6-1  
    LAPACK 圧縮ルーチンの～ 6-11  
    向上するためのハードウェアのヒント 6-11  
    向上するためのヒント 6-11  
    非正規化数 6-13  
    マルチコア 6-12

## ひ

非正規化数、パフォーマンス 6-13  
表記規則 1-3

## ふ

不安定性、数値計算、回避 8-1  
不安定な出力、数値計算、回避 8-1

## へ

並列処理 6-1  
並列パフォーマンス 6-3  
ヘルプ、インテル (R) MKL、Eclipse\* CDT 10-1  
ベンチマーク 11-1

## ま

マルチコア・パフォーマンス 6-12

## め

メモリー関数名の変更 6-14  
メモリー関数、再定義 6-14  
メモリー管理 6-13

## も

モジュール、Fortran-95 7-4

## よ

呼び出し  
    C から Fortran 形式のルーチン 7-4  
    C から複素 BLAS レベル 1 関数 7-7  
    C で BLAS 関数 7-6  
    C++ から複素 BLAS レベル 1 関数 7-7

---

## ら

### ライブラリー

ランタイム、互換 OpenMP\* 3-3

ランタイム、レガシー OpenMP\* 3-3

### ライブラリー構造 3-1

### ランタイム・ライブラリー 7-3

互換 OpenMP\* 3-3

レガシー OpenMP\* 3-3

## り

### リンク 5-1

ScaLAPACK 9-1

クラスター FFT 9-1

### リンクコマンド

例 5-7

### リンク・ライブラリー

インテル (R) 64 アーキテクチャー 5-5

計算 5-5

スレッド化 5-4

## れ

### 例

ScaLAPACK、Cluster FFT、リンク 9-3

コード 2-4

リンク、一般 5-7

### レイヤー

インターフェイス 3-3

計算 3-3

コンパイラー・サポート RTL 3-3

スレッド化 3-3

### レイヤーモデル 3-2