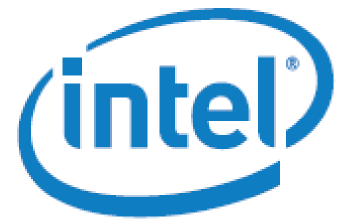

ホワイトペーパー

インテル® Xeon Phi™ コプロセッサー 開発者向けクイック・スタート・ガイド

バージョン 1.7

目次

はじめに 4
目的 4
本ガイドに含まれるトピック: 4
本ガイドに含まれないトピック: 4
用語 4
システム構成 5
インテル® Xeon Phi™ コプロセッサー向けソフトウェア 5
インテル® メニー・インテグレーテッド・コア (インテル® MIC) アーキテクチャーの概要 7
管理タスク 8
初めて使用する前のシステム準備 8
ドライバーのインストールとカードの起動手順 8
ソフトウェア開発ツールのインストール手順 9
既存のシステムのアップデート 10
インテル® Xeon Phi™ コプロセッサーが設定済みのシステムのアップデート 10
再起動後のインテル® Xeon Phi™ コプロセッサーへのアクセスの確立 11
インテル® Xeon Phi™ コプロセッサーがハングアップした場合の再起動 11
インテル® Xeon Phi™ コプロセッサーの監視 12
ホストシステムからのインテル® Xeon Phi™ コプロセッサー向けプログラムの実行 12
インテル® Xeon Phi™ コプロセッサー の uOS 環境での直接操作 12
便利な管理ツール 13
インテル® Xeon Phi™ コプロセッサー向けソフトウェアの開発 13
利用可能なソフトウェア開発ツール/環境 13
開発環境: コンパイラーとライブラリー 13
開発環境: ツール 14
開発に関する一般情報 14
開発環境のセットアップ 14
ドキュメントとサンプルコード 14
ビルドに関する情報 16
コンパイラー・オプションと makefile 16
実行中のデバッグ 16
サポート 16
オフロード・コンパイラーの使用 – 明示的なメモリー・コピー・モデル 16
リダクション 17
オフロードバージョンの作成 17
非同期オフロードとデータ転送 18
オフロード・コンパイラーの使用 – 暗黙的なメモリー・コピー・モデル 19
ネイティブコンパイル 20

インテル® Xeon Phi™ コプロセッサーで利用可能な並列プログラミング・モデル 21
インテル® Xeon Phi™ コプロセッサーで利用可能な並列プログラミング・モデル: OpenMP* 21
インテル® Xeon Phi™ コプロセッサーで利用可能な並列プログラミング・モデル: OpenMP* + インテル® Cilk™ Plus の配列表記 22
インテル® Xeon Phi™ コプロセッサーで利用可能な並列プログラミング・モデル: インテル® Cilk™ Plus 23
インテル® Xeon Phi™ コプロセッサーで利用可能な並列プログラミング・モデル: インテル® スレッディング・ビルディング・ブロック (インテル® TBB) 24
インテル® MKL の使用 25
SGEMM サンプル 26
インテル® MKL の自動オフロードモデル 27
インテル® Xeon Phi™ コプロセッサーでのデバッグ 27
インテル® Xeon Phi™ コプロセッサーでのパフォーマンス解析 27
著者紹介 28
著作権と商標について 29
パフォーマンスに関する注意事項 30
最適化に関する注意事項 30

はじめに

このガイドは、インテル® メニー・インテグレーテッド・コア (インテル® MIC) アーキテクチャー・ベースのインテル® Xeon Phi™ コプロセッサーが装着されたシステム (ホスト) 向けアプリケーションを作成し実行する際に役立つ情報を提供します。さまざまなツールを紹介し、簡単なサンプルを例に C/C++ および Fortran プログラムを作成し、実行する方法を示します。サンプルコードを実際に実行する場合は、このガイドからコードをコピーアンドペーストしてください。

このガイドは、http://www.isus.jp/article/idz/mic-developer/ の「概要」タブからも入手できます。

目的

本ガイドに含まれるトピック:

1. インテル® メニーコア・プラットフォーム・ソフトウェア・スタック (インテル® MPSS) のインストール手順
2. インテル® Xeon Phi™ コプロセッサー対応ソフトウェアのビルド環境
3. インテル® Xeon Phi™ コプロセッサー向けコードの記述例とインテル® Composer XE 2013 SP1 でのビルド方法
4. インテル® マス・カーネル・ライブラリー (インテル® MKL) などのインテルのライブラリーの使用例
5. インテル® Xeon Phi™ コプロセッサーで実行中のプログラムのデバッグ方法とプロファイル方法
6. インテルによって開発された最も一般的な手法 (BKM)

本ガイドに含まれないトピック:

1. 各種ツールの詳細情報 (各ツールのユーザーガイドを参照してください)
2. 詳細なトレーニング

用語

ホスト – PCIe* スロットにインテル® XeonPhi™ コプロセッサーが装着されたインテル® Xeon® プロセッサー・ベースのプラットフォーム。次のオペレーティング・システム (OS) がサポートされています:
Red Hat* Enterprise Linux* 6.0、Red Hat* Enterprise Linux* 6.1、Red Hat* Enterprise Linux* 6.2、Red Hat* Enterprise Linux* 6.3、Red Hat* Enterprise Linux* 6.4、Red Hat* Enterprise Linux* 6.5、SUSE* Linux* Enterprise Server SLES 11 SP2、SUSE* Linux* Enterprise Server SLES 11 SP3

ターゲット – インテル® Xeon Phi™ コプロセッサーおよびコプロセッサー内にインストールされている対応するランタイム環境。

uOS – マイクロオペレーティング・システムの略。Linux* ベースのオペレーティング・システムとインテル® Xeon Phi™ コプロセッサー上で動作するツール。

ISA – 命令セット・アーキテクチャーの略。ネイティブデータ型、命令、レジスター、アドレッシング・モード、メモリー・アーキテクチャー、割り込み/例外処理、外部 I/O など、コンピューター・アーキテクチャーのプログラミングに関連する部分。[1]

VPU – ベクトル・プロセシング・ユニットの略。SIMD (Single Instruction Multiple Data) 命令を実行する CPU 部分。

NAcc – ネイティブ・アクセラレーションの略。処理されるデータとそのデータを処理するインテル® MKL 関数がインテル® Xeon Phi™ コプロセッサー上にある、インテル® MKL のモードまたは形式。

オフロード・コンパイラー – インテル® C/C++ コンパイラーおよびインテル® Fortran コンパイラー。ホスト上でのみ実行されるバイナリー、インテル® Xeon Phi™ コプロセッサー上でのみ実行されるバイナリー、そしてホストとインテル® Xeon Phi™ コプロセッサーが互いに通信し両方で実行されるバイナリーのペアを生成することができます。

[1] Intel acronyms dictionary, 8/6/2009, http://library.intel.com/Dictionary/Details.aspx?id=5600

インテル® MPSS – インテル® メニーコア・プラットフォーム・ソフトウェア・スタックの略。プログラムがインテル® Xeon Phi™ コプロセッサーと通信し、コプロセッサー上で実行できるようになる、ユーザーレベルおよびシステムレベルのソフトウェア。

SCIF – 対称コミュニケーション・インターフェイスの略。単一プラットフォーム内のノード間通信構造。ノードは、インテル® Xeon Phi™ コプロセッサーまたはインテル® Xeon® プロセッサー・ベースのホストです。特に、SCIF はすべてのノードで対称な API を提供し、PCIe バスを介した通信の詳細 (およびインテル® Xeon Phi™ コプロセッサーのハードウェア関連の制御) を抽象化します。

システム構成

このガイドでは、2 つのインテル® Xeon®プロセッサー、PCIe* x16 バスを介して装着された 1 つまたは 2 つのインテル® Xeon Phi™ コプロセッサー、およびグラフィックス表示用の GPU で構成されるインテルのワークステーションを想定しています。

インテル® Xeon Phi™ コプロセッサー向けソフトウェア

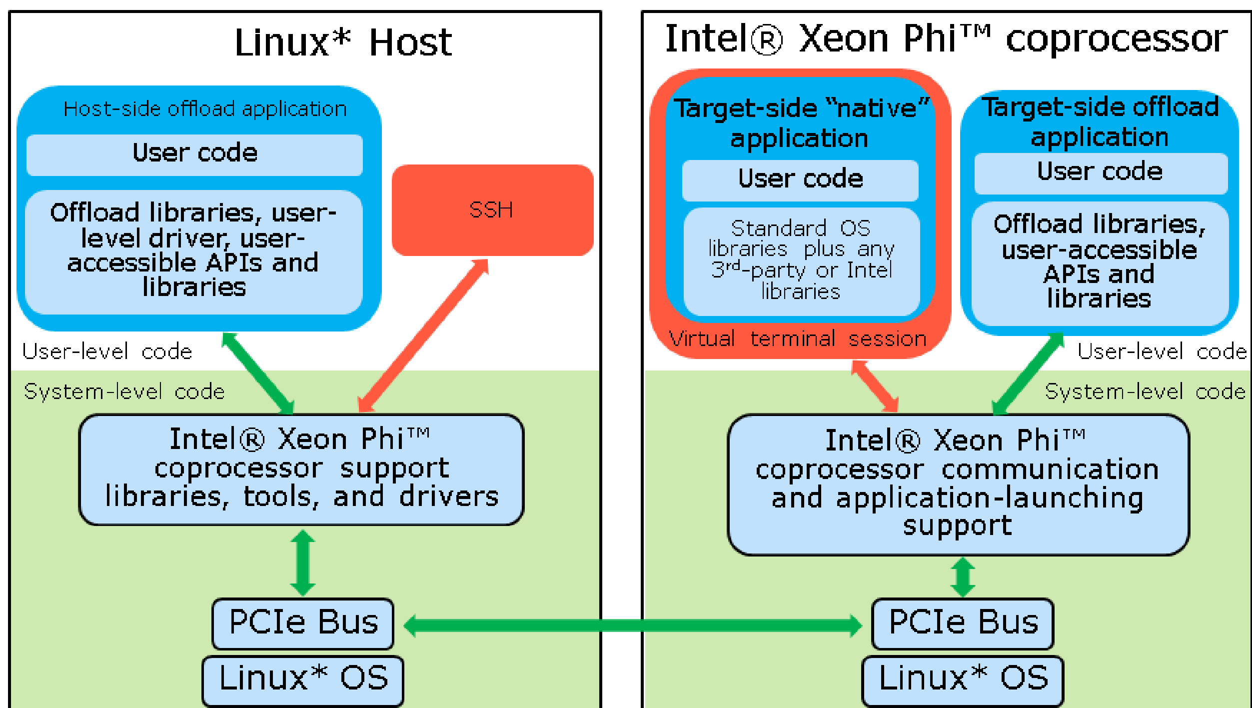

図 1: ソフトウェア・スタック

インテル® Xeon Phi™ コプロセッサーのソフトウェア・スタックは、図 1 と以下の説明に示すように、さまざまなソフトウェア・アーキテクチャーで構成されています。

ドライバースタック:
インテル® Xeon Phi™ コプロセッサー向け Linux* ソフトウェアは、次のコンポーネントで構成されています。

- **デバイスドライバー:** ソフトウェア・スタックの最下層のカーネル空間にあるインテル® Xeon Phi™ コプロセッサーのデバイスドライバー。デバイスの初期化およびホストとターゲットデバイス間の通信を管理します。
- **ライブラリー:** デバイスドライバーの上層のユーザーおよびシステム空間にあるライブラリー。システム内のカードの列挙、バッファー管理、ホストとカード間の通信などの基本的なカード管理機能に加え、インテル® Xeon Phi™ コプロセッサーへのユーザー実行ファイルのロード/アンロード、カード上の実行ファイルからの関数呼び出し、ホス

トとカード間の双方向通知構造の提供など高度な機能を提供します。バッファー管理と PCIe* バスを介した通信は、ライブラリーによって処理されます。
- **ツール:** ソフトウェア・スタックの保守に役立つ各種ツール。例えば、システム情報を照会する /usr/bin/**micinfo**、カードのフラッシュを更新する /usr/bin/**micflash**、カードの設定に役立つ /usr/sbin/**micctrl** などがあります。
- **カード OS (uOS)**: インテル® Xeon Phi™ コプロセッサーに搭載されている Linux* ベースのオペレーティング・システム。

注: 最新の uOS バージョンの Linux* ソース、デバイスドライバー、下位レベルの SCIF ライブラリー・インターフェイスについては、http://www.isus.jp/article/mic-article/software-stack-mpss/ をご覧ください。

インテル® メニー・インテグレーテッド・コア (インテル® MIC) アーキテクチャーの概要

インテル® Xeon Phi™ コプロセッサーは、最大 61 個のインオーダーのインテル® MIC アーキテクチャー・ベースのプロセッサー・コアを搭載しており、これらは 1GHz (最大 1.3GHz) で動作します。インテル® MIC アーキテクチャーは x86 ISA をベースに、64 ビットのアドレッシング、新しい 512 ビットの SIMD ベクトル命令とレジスターで拡張されています。各コアは 4 つのスレッドをサポートします。コアに加えて、複数のオンダイ・メモリー・コントローラーやその他のコンポーネントを搭載しています。

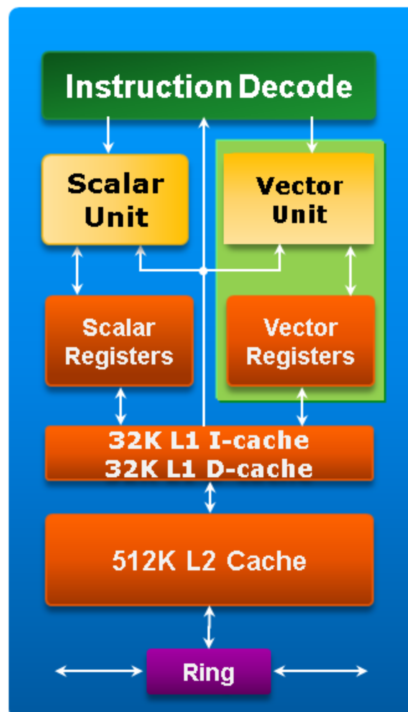

図 2: インテル® MIC アーキテクチャー・ベースのコアの概要

各コアには新しく設計されたベクトル・プロセシング・ユニット (VPU) があり、各 VPU には 32 個の 512 ビット・ベクトル・レジスターがあります。新しいベクトル・プロセシング・モデルをサポートするため、新たに 512 ビットの SIMD ISA が導入されました。

新しい VPU は、インテル® MIC アーキテクチャー・ベースのコアの主要機能です。インテル® Xeon Phi™ コプロセッサーで最良のパフォーマンスを得るには、VPU を最大限に活用することが重要です。インテル® MIC アーキテクチャー・ベースのコアは、ほかの SIMD ISA (例えば、インテル® MMX® 命令、インテル® SSE 命令、インテル® AVX 命令など) をサポートしないことに注意してください。

各コアには、32KB L1 命令キャッシュ、32KB L1 データキャッシュ、および 512KB L2 キャッシュが装備されています。すべてのコアの L2 キャッシュはインターコネクトにより互いに接続され、双方向リングバスを介してメモリー・コントローラーに接続されているため、最大 32MB の共有 LLC を効率良く利用できます。各コアには、短いインオーダーのパイプラインがあります。スカラー操作はレイテンシーなしで、ベクトル操作は低レイテンシーで実行できます。また、分岐予測ミスのオーバーヘッドも低くなっています。

マシン・アーキテクチャーについての詳細は、http://www.isus.jp/article/idz/mic-developer/ の「ツールとダウンロード」タブにある『インテル® Xeon Phi™ コプロセッサー・ソフトウェア開発者ガイド』(英語) を参照してください。

管理タスク

メーカーからインテル® Xeon Phi™ コプロセッサーを購入した場合は、インテル® デベロッパー・ゾーンの http://software.intel.com/mic-developer (英語) にアクセスし、「TOOLS & DOWNLOADS」タブにある「Software Drivers: Intel® Manycore Platform Software Stack (Intel® MPSS)」をクリックします。表示されるページから、最新のハードウェア・ドライバーとリリースノートをダウンロードできます。

初めて使用する前のシステム準備

ドライバーのインストールとカードの起動手順

1. インテル® デベロッパー・ゾーンの http://software.intel.com/mic-developer (英語) にアクセスし、「TOOLS & DOWNLOADS」タブにある「Software Drivers: Intel® Manycore Platform Software Stack (Intel® MPSS)」をクリックします。表示されるページの「ダウンロード」セクションから、Linux* 用の最新バージョンの Readme ファイル (*readme.txt*)、リリースノート (*releaseNotes-linux.txt*)、およびユーザーズガイドをダウンロードします。
2. システムに次のいずれかの OS をインストールします。
   Red Hat* Enterprise Linux* (64 ビット) 6.0 カーネル 2.6.32-71、Red Hat* Enterprise Linux* (64 ビット) 6.1 カーネル 2.6.32-131、Red Hat* Enterprise Linux* (64 ビット) 6.2 カーネル 2.6.32-220、Red Hat* Enterprise Linux* (64 ビット) 6.3 カーネル 2.6.32-279、Red Hat* Enterprise Linux* (64 ビット) 6.4 カーネル 2.6.32-358、Red Hat* Enterprise Linux* (64 ビット) 6.5 カーネル 2.6.32-431、SUSE* Linux* Enterprise Server SLES 11 SP2 カーネル 3.0.13-0.27-default または SUSE* Linux* Enterprise Server SLES 11 SP3 カーネル 3.0.76-0.11-default (*readme.txt* のセクション 2.1 を参照)。カードの uOS へのログインに使用するため、ssh も必ずインストールしてください。

   警告: Red Hat* のインストールでは、自動的に新しいバージョンの Linux* カーネルにアップデートされることがあります。その場合、事前ビルド済みのホストドライバーは利用できません。新しいカーネルバージョンでリビルドする必要があります。特定の Linux* カーネルでインテル® MPSS ホストドライバーをビルドする手順は、*readme.txt* のセクション 2.1 を参照してください。
3. root でログインします。
4. ステップ 1 でインストールしたオペレーティング・システム用のドライバーをダウンロードします (<*mpss-version*>-rhel-6.0.tgz、<*mpss-version*>-rhel-6.1.tgz、<*mpss-version*>-rhel-6.2.tgz、<*mpss-version*>-rhel-6.3.tgz 、<*mpss-version*>-rhel-6.4.tgz 、<*mpss-version*>-rhel-6.5.tgz 、<*mpss-version*>-suse-11.2.tgz、または <*mpss-version*>-suse-11.3.tgz)。このガイドの更新時点で、<*mpss-version*> は mpss-3.2 です。
5. *readme.txt* のセクション 2.2 の手順に従って、ホストドライバーの RPM をインストールします。コプロセッサー用の設定ファイルの作成も必ず行ってください。
6. readme.txt のセクション 2.4 の手順に従って、コプロセッサーのフラッシュをアップデートします。
7. システムを再起動します。
8. インテル® Xeon Phi™ コプロセッサーを起動します (ホストシステムの起動時にカードを起動するように設定することもできます。この設定はデフォルトでは行われません)。そして、micinfo コマンドを実行して適切に設定されていることを確認します。

   ```
   sudo service mpss start
   sudo micctrl -w
   sudo /usr/bin/micinfo
   ```

- 出力される Driver Version、MPSS Version、Flash Version が次の表の値と一致していることを確認します。

インテル® MPSS リリース	Driver Version	MPSS Version	Flash Version
mpss-3.2	3.2-xx	3.2	2.1.03.0386
mpss-3.1	3.1-xx	3.1	2.1.03.0386
mpss_gold_update_3-2.1.6720-13	6720-13	2.1.6720-13	2.1.02.0386
KNC_gold_update_2-2.1.5889-16	5889-16	2.1.5889-16	2.1.05.0385
KNC_gold_update_1-2.1.4982-15	4982-15	2.1.4982-15	2.1.05.0375
KNC_gold-2.1.4346-xx	4346-xx	2.1.4346-xx	2.1.01.0375

表 1: 各インテル® MPSS リリースの Driver Version、MPSS Version、Flash Version

ソフトウェア開発ツールのインストール手順

ソフトウェア開発ツールについては、http://www.xlsoft.com/jp/products/intel/products.html を参照してください。ニーズにあったツールを選択します (インテル® Cluster Studio XE 2013、インテル® C++ Composer XE Linux* 版、インテル® Fortran Composer XE Linux* 版など)。ツールを購入すると、シリアル番号を受け取ります。インテル® Xeon Phi™ コプロセッサーでサポートされているツールの最新リストは、http://software.intel.com/en-us/mic-developer/ の「Tools and Downloads」にある「Intel® Software Development Products"」で確認できます。

インテル® ツールのシリアル番号を受け取ったら、インテル® ソフトウェア開発製品レジストレーション・センター (http://registrationcenter.intel.com) で製品を登録し、ダウンロードします。必要項目を入力し [製品の登録] ボタンをクリックすると、購入したツールのダウンロード・ページが表示されます。ここでは、インテル® Cluster Studio XE Linux* 版を例に説明します。http://www.isus.jp/article/intel-software-dev-products/intel-cluster-studio-xe/ の「サポート」タブの「リソース」セクションにある「インテル® Cluster Studio XE 製品ドキュメント」から、インストール・ガイド、入門ガイド、リリースノートを入手できます。

1. インストール・ガイドの手順に従って、インテル® Cluster Studio XE Linux* 版をインストールします。インテル® C++ Composer XE Linux* 版またはインテル® Fortran Composer XE Linux* 版を購入した場合は、それぞれのインストール・ガイドを参照してインストールしてください。また、インテル® VTune™ Amplifier XE 2013 Linux* 版は別途インストールが必要です。
    - 初めてインストールする場合は、製品をアクティベーションするため、インストール時にシリアル番号を入力する必要があります。次回のインストールからは、[既存のライセンスを使用する] オプションを利用できます。
    - 製品のリリースノート (インテル® Cluster Studio XE Linux* 版の場合は *icsxe2013sp1-update1-release-notes.pdf*、インテル® C++ Composer XE Linux* 版の場合は *Release_Notes_C_2013SP1_L_EN_Update2.pdf*、インテル® Fortran Composer XE Linux* 版の場合は *Release-notes-f-2013sp1-l-en-u2.pdf*) をよくお読みください。
    - 製品のパッケージファイルを展開します。
        - `tar -xvzf l_ics_2013.<update>.<package_num>.tgz` (インテル® Cluster Studio XE Linux* 版の場合)
        - `tar -xvf l_ccompxe_intel64_2013.<update>.<package_num>.tgz` (インテル® C++ Composer XE Linux* 版の場合)
        - `tar -xvf l_fcompxe_intel64_2013.<update>.<package_num>.tgz` (インテル® Fortran Composer XE Linux* 版の場合)

2. 受け取ったシリアル番号を使って、ソフトウェア・ツールをインストールします。
3. ホストとインテル® Xeon Phi™ コプロセッサー間の通信内容を表示する `"setenv H_TRACE 2"` または `"export H_TRACE=2"` を指定し、`/opt/intel/composerxe/Samples/ja_JP/C++/mic_sample` (C/C++ コード) または `/opt/intel/composerxe/Samples/ja_JP/Fortran/mic_sample` (Fortran コード) にあるサンプルプログラムを実行して、カードが正常に動作することを確認します (プロセッサーからのメッセージには、プリフィクス "MIC:" が付けられます)。通信内容が表示されれば、カードは正しく動作しており、使用できる状態です。
4. インテル® VTune™ Amplifier XE 2013 を使用してパフォーマンス・データの収集を行う場合は、次の操作を行います。
   a) インテル® MPSS は起動後、自動でデータ収集ドライバーをロードします。しかし、何らかの理由によりインテル® MPSS がデータ収集ドライバーのロードに失敗した場合は、`/opt/intel/vtune_amplifier_xe/bin64/k1om/` に移動し、次のコマンドを実行して、手動でドライバーをロードできます。

      ```
      sudo sep_micboot_install.sh
      ```

   b) インテル® MPSS サービスを開始 (または再起動) します (前のステップでファイルのコピーが終わっている場合、サンプリング・ドライバーも開始されます)。

      ```
      sudo service mpss restart
      sudo micctrl -r
      sudo micctrl -w
      ```

      コプロセッサーが正常に再起動されると、`micctrl -w` コマンドは `micx: online` を出力します。
   c) 次回から、コプロセッサーが再起動されるたびにサンプリング・ドライバーも起動されます。
   d) サンプリング・ドライバーを再インストールする必要がある場合は、次のコマンドを実行します。

      ```
      sudo service mpss stop
      sudo sep_micboot_uninstall.sh
      sudo service mpss restart
      sudo micctrl -w
      ```

既存のシステムのアップデート

インテル® Xeon Phi™ コプロセッサーが設定済みのシステムのアップデート

1. インテル® デベロッパー・ゾーンの http://software.intel.com/mic-developer (英語) にアクセスし、「TOOLS & DOWNLOADS」タブにある「Software Drivers: Intel® Manycore Platform Software Stack(Intel® MPSS)」をクリックします。インストールするインテル® MPSS リリースの Readme ファイル (*readmetxt*) とリリースノート (*releaseNotes-linux.txt*) をダウンロードします。
2. インテル® MPSS の以前のバージョンをアンインストールし、*readme.txt* のセクション 2.3 の手順に従って、新しいバージョンをインストールします。
3. *readme.txt* のセクション 2.4 の手順に従って、コプロセッサーのフラッシュをアップデートします。
4. システムを再起動します。

5. インテル® Xeon Phi™ コプロセッサーを起動します (ホストシステムの起動時にカードを起動するように設定することもできます。この設定はデフォルトでは行われません)。そして、micinfo コマンドを実行して適切に設定されていることを確認します。

```
sudo service mpss start
sudo micctrl -w
/usr/bin/micinfo
```

- 出力される Driver Version、MPSS Version、Flash Version が前述の表 1 の値と一致していることを確認します。

再起動後のインテル® Xeon Phi™ コプロセッサーへのアクセスの確立

インテル® Xeon Phi™ コプロセッサーは、ホストシステムの再起動時に起動されません。そのため、手動でインテル® Xeon Phi™ コプロセッサーを起動し、micinfo コマンドを実行して正常に起動されたかどうかを確認する必要があります。root 以外のユーザーが sudo 権限でこの処理を行う場合は、そのユーザーのパスに /usr/sbin と /sbin を追加する必要があります。

```
sudo service mpss start
sudo micctrl -w
/usr/bin/micinfo
```

注: 起動時にコプロセッサーの uOS を自動的に起動し、必要なファイルをロードするように設定できます。詳細は、『インテル® MPSS ユーザーズガイド』のセクション 19.12 を参照してください。

インテル® Xeon Phi™ コプロセッサーがハングアップした場合の再起動

インテル® Xeon Phi™ コプロセッサーで、あるプロセスだけがハングアップし、ssh を介したその他の応答には問題がない場合、コプロセッサーにログインして、ほかの Linux* プロセスと同様の方法でそのプロセスを強制終了します。

コプロセッサーがハングアップしアクセスできない場合、あるいは ssh でも応答しない場合、コプロセッサーを再起動する方法は 2 つあります。最初に、次のコマンドでハングアップの原因を探ります。

```
sudo micctrl --status <micx>
```

インテル® MPSS サービスが正しく動作している場合は、次のコマンドを実行することで、装着されているほかのコプロセッサーに影響を与えることなく、問題のコプロセッサーの再起動を試みることができます。

```
sudo micctrl --reset <micx>
sudo micctrl --boot <micx>
sudo micctrl -w
/usr/bin/micinfo
```

インテル® MPSS サービスが正しく動作していない場合は、ドライバーと装着されているすべてのコプロセッサーを再起動する必要があります。

```
sudo service mpss stop
sudo service mpss unload
sudo service mpss start
sudo micctrl -w
/usr/bin/micinfo
```

インテル® Xeon Phi™ コプロセッサーの監視

コプロセッサーの負荷や温度などを監視するには、SMC (System Management and Configuration) ユーティリティーを実行します。詳細は、『インテル® MPSS ユーザーズガイド』のセクション 8.3 を参照してください。

次のコマンドでモニターを実行します。

```
/usr/bin/micsmc &
```

引数を指定しないと GUI モードになり、指定するとコマンドライン・モードになります。

ホストシステムからのインテル® Xeon Phi™ コプロセッサー向けプログラムの実行

micnativeloadex ユーティリティーを利用して、インテル® MIC アーキテクチャー用のネイティブバイナリーを指定されたインテル® Xeon Phi™ コプロセッサーにコピーし、実行することができます。このユーティリティーは、ライブラリー依存ファイルもコプロセッサーにコピーします。詳細は、『インテル® MPSS ユーザーズガイド』のセクション 8.5 を参照してください。

インテル® Xeon Phi™ コプロセッサー の uOS 環境での直接操作

コプロセッサーは Linux* が動作している独立したネットワーク・ノードなので、ssh を介して root または root 以外のユーザーとしてログインし、多くの一般的な Linux* コマンドを利用できます。コプロセッサーとのファイルの受け渡しには、scp やその他の手段を使用します。

デフォルトでは、ホストから見たコプロセッサーの IP アドレスは 172.31.<coprocessor>.1 で、コプロセッサーから見たホストの IP アドレスは 172.31.<coprocessor>.254 です。ホストからコプロセッサーを参照する場合は、エイリアス mic<coprocessor> を使用することもできます。例えば、システムに最初に装着したコプロセッサーは "mic0" となり、その IP アドレスは 172.31.1.1 になります。このコプロセッサーから見たホストの IP アドレスは 172.31.1.254 です。2 つ目のコプロセッサーは "mic1" で、172.31.2.1 になり、ホストは 172.31.2.254 になります。

root 以外のユーザー向けのカード設定、ネットワーク構成の調整、ホストによりインテル® Xeon Phi™ コプロセッサーへエクスポートされた NFS ファイルシステムのマウントなどに関する詳細は、『インテル® MPSS ユーザーズガイド』を参照してください。

便利な管理ツール

インテル® MPSS には便利な管理ツールが含まれています。これらは、`/usr/bin` ディレクトリーにあります。root およびこれらのツールを使用するユーザーは、このフォルダーをデフォルトのパスに追加しておくべきです。

- **micinfo** - ホストとコプロセッサーのシステム構成に関する情報を提供します。
- **micflash** - コプロセッサー上のフラッシュを更新します。フラッシュの各セクションのバージョンおよびその他の情報を保存/取得します。
- **micsmc** - インテル® Xeon Phi™ コプロセッサーの監視と管理を支援します。
- **miccheck** – さまざまな診断テストを実行してインテル® Xeon Phi™ コプロセッサーの設定を確認します。
- **micnativeloadex** – インテル® MIC アーキテクチャー用のネイティブバイナリーを、指定されたインテル® Xeon Phi™ コプロセッサーにコピーして実行します。
- **micctrl** – コプロセッサーの設定や再起動などを行うシステム管理ツールです。
- **micrasd** – ホストで動作します。ハードウェア・エラーを処理し記録します。
- **mpssflash** – *micflash* の POSIX* バージョンです。
- **mpssinfo** – *micinfo* の POSIX* バージョンです。

これらのツールの詳細と引数は、『インテル® MPSS ユーザーズガイド』のセクション 7、8、9 を参照してください。

インテル® Xeon Phi™ コプロセッサー向けソフトウェアの開発

インテル® MIC アーキテクチャー向けアプリケーションの開発は、マルチコアおよび SIMD プログラミングの既存の知識に基づいて行います。オフロード言語拡張により、インテル® Xeon Phi™ コプロセッサーで実行するため (C/C++ または FORTRAN で記述された) コードの一部を移植したり、あるいはインテル® MIC アーキテクチャー向けにアプリケーション全体を移植することができます。最良のパフォーマンスは、高度に最適化され、ほとんどの実行に (コンパイラーにより生成された、またはコンパイラーの組込み関数を使用して生成された) SIMD 操作を用いるアプリケーションでのみ達成できます。

利用可能なソフトウェア開発ツール/環境

既存の並列プログラミングの知識とホストの並列アプリケーション開発と同じ手法を利用して、インテル® Xeon Phi™ コプロセッサーの開発に取り掛かることができます。インテル® Xeon Phi™ コプロセッサー専用の新しい開発ツールはありませんが、インテル® MIC アーキテクチャーに対応するため、いくつかの標準言語と API の追加により、ホスト用の既存のインテル® ツールが拡張されています。開発ツールを最大限に利用し、インテル® Xeon Phi™ コプロセッサーから最良のパフォーマンスを引き出すには、インテル® MIC アーキテクチャーについて理解することが重要です。

開発環境: コンパイラーとライブラリー

- **コンパイラー**
  - インテル® C++ Composer XE 2013 SP1。インテル® 64 アーキテクチャーおよびインテル® MIC アーキテクチャーで動作するアプリケーションをビルドできます。
  - インテル® Fortran Composer XE 2013 SP1。インテル® 64 アーキテクチャーおよびインテル® MIC アーキテクチャーで動作するアプリケーションをビルドできます。
- **ライブラリー** (インテル® Composer XE に含まれる):
  - インテル® マス・カーネル・ライブラリー (インテル® MKL)。インテル® MIC アーキテクチャー向けに最適化されています。
  - インテル® スレッディング・ビルディング・ブロック (インテル® TBB)
  - インテル® インテグレーテッド・パフォーマンス・プリミティブ (インテル® IPP)
- **ライブラリー** (インテル® Composer XE に含まれる):

- インテル® メニー・インテグレーテッド・コア (インテル® MIC) アーキテクチャー対応のインテル® MPI ライブラリー Linux* 版
- インテル® Trace Collector & Analyzer
- インテル® SDK for OpenCL* Applications XE 2013 (http://www.isus.jp/article/idz/opencl-sdk/ から入手可能)

開発環境: ツール

上記のコンパイラーとライブラリーに加えて、次のツールを利用してインテル® Xeon Phi™ コプロセッサーで動作するソフトウェアのデバッグと最適化を行えます。

- **デバッガー**
  - インテル® デバッガー。インテル® 64 アーキテクチャーおよびインテル® MIC アーキテクチャーで動作するアプリケーション向け。
  - インテル® C++ コンパイラーの Eclipse* 拡張 (デバッグを含む)
- **プロファイル**
  - インテル® VTune™ Amplifier XE 2013 Linux* 版。Linux* ベースのホストシステムでインテル® Xeon Phi™ コプロセッサーのデータを収集し、収集したデータを確認できます。
  - インテル® Inspector XE 2013。シリアルおよび並列アプリケーションのメモリーエラーとスレッドエラーを検出します。
  - インテル® Advisor XE 2013。スレッドの設計を支援します。

開発に関する一般情報

開発環境のセットアップ

- 開発環境でインテル® ツールを利用するには、source コマンドを使用して次のセットアップ・スクリプトを実行します。
  - **インテル® C++/Visual Fortran Composer XE 2013 SP1:** 次のように、`intel64` を引数として `/opt/intel/composerxe/bin` 以下の `compilervars.csh` または `compilervars.sh` スクリプトを実行します。
    `source /opt/intel/composerxe/bin/compilervars.sh intel64`

  compilervars スクリプトを呼び出すと、次のスクリプトが実行されます。環境の初期化が適切に行われるようにするには、これらのスクリプトを個別に実行すべきではありません (実行順序によっては、予期しない動作を引き起こします)。

  - **インテル® デバッガー:** `intel64` を引数として `/opt/intel/composerxe/pkg_bin` 以下の `idbvars.csh` または `idbvars.sh` スクリプトを実行します。
  - **インテル® TBB:** `intel64` を引数として `/opt/intel/composerxe/tbb/bin` 以下の `tbbvars.csh` または `tbbvars.sh` を実行します。
  - **インテル® MKL:** `intel64` を引数として `/opt/intel/composerxe/mkl/bin` 以下の `mklvars.csh` または `mklvars.sh` を実行します。

ドキュメントとサンプルコード

- 次の役立つドキュメントが、`/opt/intel/composerxe/Documentation/ja_JP/` にインストールされます。
  - **`compiler_c/main_cls/index.htm`** および **`compiler_f/main_cls/index.htm`** - インテル® C++ コンパイラー XE 14.0 およびインテル® Fortran コンパイラー XE 14.0 のドキュメント。

- インテル® MIC アーキテクチャー向けのビルドに関するほとんどの情報は、「主な機能」>「インテル® MIC アーキテクチャー」>「インテル® MIC アーキテクチャー用にビルド」セクションにあります。
- インテル® MIC アーキテクチャー向け組込み関数に関する情報は、「コンパイラー・リファレンス」>「組込み関数」>「インテル® MIC アーキテクチャー向け組込み関数」セクションにあります。

  - **`Release_notes_*_2013SP1_l_en.pdf`** - インテル® MIC アーキテクチャーをサポートするすべてのツールに関する既知の問題とその回避方法、インストール手順をよくお読みください。インテル® MIC アーキテクチャーに関する情報は、主にセクション 4 にあります。
    - **注**: さまざまな理由から、このドキュメントには最新の情報が含まれていない可能性があります。最新のリリースノート (*Release_notes_*_2013SP1_l_en.pdf*) は、インテル® ソフトウェア開発製品レジストレーション・センターからダウンロードできます (「ソフトウェア開発ツールのインストール手順」セクションを参照)。
  - **`debugger/debugger_documentation.htm`** (英語) – インテル® デバッガーの使用方法が記載されています。インテル® MIC アーキテクチャー向けアプリケーションに関する情報は、「Debugging with the Intel® Debugger on Eclipse*」セクションと「Debugging on the Command Line」セクションにあります。
- 次のドキュメントにもインテル® Xeon Phi™ コプロセッサーに関するセクションがあります。
  - インテル® MKL ユーザーズガイド。`/opt/intel/composerxe/Documentation/ja_JP/mkl` 以下の **`mkl_documentation.htm`** から利用できます。「インテル® Xeon Phi™ コプロセッサーでのインテル® MKL の使用」セクションにインテル® MKL 関数の「自動オフロード」と「コンパイラーによるオフロード支援」の説明があります。
  - インテル® VTune™ Amplifier XE 2013 Windows* 版によるインテル® Xeon Phi™ コプロセッサーでのパフォーマンス・データの収集に関する情報は、`/opt/intel/vtune_amplifier_xe_2013/documentation/en/tutorials/find_lw_hotspots/C++/index.htm` (英語) にあります。
- 役立つ Web ドキュメント:
  - http://www.isus.jp/article/idz/mic-developer/ から、さまざまなドキュメントをダウンロードできます。特に「プログラミング」タブにある「インテル® Xeon Phi™ コプロセッサー・ソフトウェア開発者ガイド」(英語)、「System V Application Binary Interface K1OM Architecture Processor Supplement」(英語)、「パフォーマンス・モニタリング・ユニット」、および「概要」タブの「インテル® Xeon Phi™ コプロセッサー命令セットアーキテクチャー・リファレンス・マニュアル」(英語) が役立ちます。このページには、ツールやコードサンプルに関する情報もあります。
  - http://www.isus.jp/article/mic-article/xeon-phi/ には、コンパイラーに関するさまざまな情報があります。
- 明示的なメモリー・コピー・モデルを使用するサンプル・オフロード・コード:
  - **C++**: `/opt/intel/composerxe/Samples/ja_JP/C++/mic_samples/intro_sampleC/`
  - **Fortran**: `/opt/intel/composerxe/Samples/ja_JP/Fortran/mic_samples/`
  - **インテル® MKL**: `/opt/intel/composerxe/mkl/examples/mic*`
    - インテル® MKL の自動オフロード機能のサンプル: `/opt/intel/composerxe/mkl/examples/mic_ao` 以下の `blasc` および `blasf`
    - インテル® MKL のコンパイラーによるオフロード支援のサンプル: `/opt/intel/composerxe/mkl/examples/mic_offload`
- 暗黙的なメモリー・コピー・モデルを使用するサンプル・オフロード・コード:
  - **C**: `/opt/intel/composerxe/Samples/ja_JP/C++/mic_samples` 以下の `shrd_sampleC` および `LEO_tutorial`

  - **C++**: /opt/intel/composerxe/Samples/ja_JP/C++/mic_samples/shrd_sampleCPP

ビルドに関する情報

- オフロード・コンパイラーは、ホスト用のコードとインテル® Xeon Phi™ コプロセッサー用のコードを含む「ファット」バイナリーと .so ファイルを生成します。
- 利用可能なインテル® Xeon Phi™ コプロセッサーがあるかどうか、ランタイム実行環境を確認するコードも生成します。オフロード・コンパイラーは、オフロード用と表記されているすべてのコードに対し、ホスト用とインテル® MIC アーキテクチャー用のバージョンを作成します。
- *releaseNotes-linux.txt* に多くの回避方法とヒントが掲載されています。

コンパイラー・オプションと makefile

一部のコードをインテル® Xeon Phi™ コプロセッサーにオフロードするアプリケーションをビルドする場合、ホストコードとオフロードコードで異なるコンパイラー・オプションを指定できます。各コンパイラー・オプションの指定方法は、コンパイラー・ドキュメントの「コンパイラー・リファレンス」> 「コンパイラー・オプション」> 「コンパイラー・オプションのカテゴリーと説明」セクションに記載されています。ここで、`-offload-option` オプションと、`-offload-attribute-target` オプションの説明を確認してください。場合によっては、ソースファイルを変更する代わりに `–offload-attribute-target` オプションを利用できます (このオプションは、プラグマベースのオフロード手法に適用されます)。また、`-no-offload` を指定すると、コンパイラーは `_Cilk_offload` 構造と `#pragma_offload` 構造を無視します (その結果、デフォルトでヘテロジニアス・バイナリーが生成されます)。

実行中のデバッグ

オフロード・アクティビティーをデバッグするには、次の環境変数を利用します。

- プログラムのオフロード領域がホストで実行されているか、コプロセッサーで実行されているかを検出するには、次のように設定します。

  csh の場合: – `setenv H_TRACE 1`

  sh の場合: – `export H_TRACE=1`
- より詳細なデバッグ情報を取得するには、次のように設定します。

  csh の場合: – `setenv H_TRACE 2`

  sh の場合: – `export H_TRACE=2`
- コンパイラー内部のオフロードタイマーを出力する場合、1 に設定するとホストにより測定されたオフロード時間とコプロセッサーによる計算時間のみ出力され、2 に設定するとホストとコプロセッサー間のデータ転送量も出力されます。

  csh の場合: – `setenv OFFLOAD_REPORT <1 または 2>`

  sh の場合: – `export OFFLOAD_REPORT=<1 または 2>`

詳細は、コンパイラー・ドキュメントの「コンパイル」> 「サポートされている環境変数」セクションを参照してください。

サポート

質問がある場合は、インテル® Xeon Phi™ コプロセッサーのフォーラム (http://software.intel.com/en-us/forums/intel-many-integrated-core) (英語) に投稿することができます。

オフロード・コンパイラーの使用 – 明示的なメモリー・コピー・モデル

このセクションでは、リダクションを例に、オフロード・コンパイラーでインテル® Xeon Phi™ コプロセッサー向けアプリケーションを生成する方法を説明します。オフロード・コンパイラーは、ホスト CPU とターゲット・コンパイル環境に対応

するヘテロジニアス [2]・コンパイラーです。ホスト CPU 用のコードとインテル® Xeon Phi™ コプロセッサー用のコードはどちらもホスト環境でコンパイルされ、オフロードコードは自動的にターゲット環境で実行されます。オフロード動作は、コンパイラー・プラグマ (C/C++) とコンパイラー宣言子 (Fortran) により制御されます。

インテル® マス・カーネル・ライブラリー (インテル® MKL) のような一部の一般的なライブラリーには、CPU バージョンとターゲットバージョンがあります。アプリケーションで最初のオフロードを実行する際にターゲットが利用可能であれば、ランタイムはインテル® Xeon Phi™ コプロセッサーにターゲット用の実行ファイルをロードし、ターゲットコードとリンクされているライブラリーを初期化します。ホストプログラムが終了するまで、ロードしたターゲット用の実行ファイルはターゲットのメモリーに残ります。そのため、ライブラリーによって維持されるグローバル状態はすべてのオフロード・インスタンスで維持されます。

注: プログラマーがターゲットで実行するコード領域を指定した場合であっても、そのコード領域がインテル® Xeon Phi™ コプロセッサーで実行される保証はありません。ターゲット・ハードウェアが利用可能かどうか、あるいは実行がオフロード領域に到達したときにインテル® Xeon Phi™ コプロセッサーで利用可能なリソースに応じて、そのコードをインテル® Xeon Phi™ コプロセッサーで実行するかどうかが決定されます。

次の例は、オフロードプラグマを使って、リダクション・コードをインテル® Xeon Phi™ コプロセッサー向けに移植する方法を示します。

リダクション

次の式を計算します。

```
ans = a[0] + a[1] + … + a[n-1]
```

ホストバージョン:

次のサンプル C コードは、このリダクション操作を実装します。

```
float reduction(float *data, int size)
{
    float ret = 0.f;
    for (int i=0; i<size; ++i)
    {
        ret += data[i];
    }
    return ret;
}
```

サンプルコード 1: リダクション・コードの実装 (C/C++)

オフロードバージョンの作成

オフロードを使用するシリアル・リダクション

プログラマーは、(次のサンプルコードに示すように) **`#pragma offload target(mic)`** を用いてインテル® Xeon Phi™ コプロセッサーで実行する文 (オフロード構造) を指定できます。オフロード領域は、オフロード構造と関数呼び出しによりターゲットで実行される追加のコード領域で定義されます。ホスト上の文の実行は、ターゲット上の文の実行が完了すると再開され、ターゲットで実行された処理の結果はホストで利用できます (つまり、このプラグマには非同期実行が可能なバージョンがあるにもかかわらず、オフロードによりホストの実行がブロックされます)。**`in`**、**`out`**、**`inout`** 節は、ホストとターゲット間のデータ転送の方向を示します。

[2] http://dictionary.reference.com/browse/heterogeneous

オフロード構造の有効範囲外 (ファイルの有効範囲外も含む) で宣言され、オフロード構造内で使用される変数は、実行前にホストからターゲットにコピーされ (デフォルト)、実行後にターゲットからホストに戻されます。

例えば、次のコードで変数 **ret** は、実行前にホストからターゲットに自動的にコピーされ、実行後にターゲットからホストに戻されます。次のオフロードコードは、1 つのインテル® MIC アーキテクチャー・ベースのコアで 1 つのスレッドによって実行されます。

```
float reduction(float *data, int size)
{
    float ret = 0.f;
    #pragma offload target(mic) in(data:length(size))
    for (int i=0; i<size; ++i)
    {
        ret += data[i];
    }
    return ret;
}
```

サンプルコード 2: オフロードを使用するシリアル・リダクション

オフロードを使用するベクトル・リダクション

インテル® Xeon Phi™ コプロセッサーの各コアには VPU が装備されています。オフロード・コンパイラーでは、デフォルトで自動ベクトル化オプションが有効になります。さらに、次のコードのように、インテル® Cilk™ Plus の配列表記でベクトル化を最大限にし、インテル® MIC アーキテクチャー・ベースのコアにある 32 個の 512 ビット・レジスターを利用することができます。このオフロードコードは、1 つのコアで 1 つのスレッドによって実行されます。スレッドは、ビルトインのリダクション関数 *__sec_reduce_add()* により、コアの 32 個の 512 ビット・ベクトル・レジスターを使用し、一度に配列の 16 個の要素をレデュースします。

```
float reduction(float *data, int size)
{
    float ret = 0;
    #pragma offload target(mic) in(data:length(size))
    ret = __sec_reduce_add(data[0:size]); // インテル® Cilk™ Plus の配列表記
    return ret;
}
```

サンプルコード 3: オフロードを使用するベクトル・リダクション (C/C++)

非同期オフロードとデータ転送

ホストとインテル® Xeon Phi™ コプロセッサー間では、非同期のオフロードおよびデータ転送が可能です。詳細は、『インテル® C++ コンパイラー・ユーザー・リファレンス・ガイド』の「主な機能」>「インテル® MIC アーキテクチャー」>「インテル® MIC アーキテクチャー向けプログラミング」以下にある、「非同期計算について」と「非同期データ転送について」を参照してください。

非同期のオフロードおよび転送の使用例は、`/opt/intel/composerxe/Samples/ja_JP/C++/mic_samples/intro_sampleC/sampleC13.c` を参照してください。

C/C++ の明示的なメモリー・コピー・モデルでは、配列要素がスカラー型か、ビット単位でコピーできる構造体またはクラスの場合のみ、配列を利用できます。ポインターの配列はサポートされません。C/C++ の複雑なデータ構造では、暗黙的なメモリー・コピー・モデルを使用してください。詳細は、『インテル® C++ コンパイラー・ユーザー・リファレンス・

ガイド』の「主な機能」>「インテル® MIC アーキテクチャー」>「インテル® MIC アーキテクチャー向けプログラミング」>「プラグマを使用したオフロード」>「プラグマを使用したオフロードの制約事項」を参照してください。

オフロード・コンパイラーの使用 – 暗黙的なメモリー・コピー・モデル

インテル® Composer XE 2013 では、リンクリストやバイナリーツリーなどの複雑なポインター・ベースのデータ構造を処理するため、「共有メモリー」オフロード・プログラミング・モデルを提供する C/C++ 向けの 2 つのキーワード拡張 (_Cilk_shared および _Cilk_offload) が新たに追加されています (これらのキーワードは Fortran では利用できません)。「共有メモリー」オフロード・プログラミング・モデルは、ホストとコプロセッサーで共有する (_Cilk_shared キーワードで示された) 変数をそれぞれのマシンの同じ仮想アドレスに配置し、_Cilk_offload キーワードで示されたオフロード関数の開始時と終了時にその値を同期します。また、同期するデータは、それぞれのマシンで同じ仮想アドレスにメモリーが割り当てられることを保証する、特殊な割り当て/解放関数を用いて動的に割り当てられます。

共有メモリーの動的割り当て API:

```
void *_Offload_shared_malloc(size_t size);
_Offload_shared_free(void *p);
```

アライメントされた共有メモリーの動的割り当て API:

```
void *_Offload_shared_aligned_malloc(size_t size, size_t alignment);
_Offload_shared_aligned_free(void *p);
```

これは、実際には「共有メモリー」でないことに注意してください。インテル® Xeon Phi™ コプロセッサーの一部をホストシステムに割り当てることができるハードウェアはありません。コプロセッサーとホストのメモリー・サブシステムは完全に独立しており、このプログラミング・モデルは、適切に定義された同期ポイントで 2 つのメモリー・サブシステム間のデータをコピーする方法の 1 つにすぎません。コピーは暗黙的に行われます。同期ポイント (_Cilk_offload で示されたオフロード呼び出し) でコピーするデータを指定しません。代わりに、ホストとコプロセッサー間で変更のあったデータをランタイムが特定し、差分のみをオフロード呼び出しの開始時と終了時にコピーします。

次のコードは、_Cilk_shared および _Cilk_offload キーワードの使用法と「共有」メモリーを動的に割り当てる方法を示します。

```
float * _Cilk_shared data; // 「共有」メモリーへのポインター

_Cilk_shared float MIC_OMPReduction(int size)
{
    #ifdef __MIC__
    float Result;
    int nThreads = 32;
    omp_set_num_threads(nThreads);

    #pragma omp parallel for reduction(+:Result)
    for (int i=0; i<size; ++i)
    {
        Result += data[i];
    }
    return Result;


    #else
    printf("Intel(R) Xeon Phi(TM) Coprocessor not available\n");
    #endif
    return 0.0f;
}

int main()
```

```
{
    size_t size = 1*1e6;
    int n_bytes = size*sizeof(float);
    data = (_Cilk_shared float *)_Offload_shared_malloc (n_bytes);
    for (int i=0; i<size; ++i)
    {
        data[i] = i%10;
    }

    _Cilk_offload MIC_OMPReduction(size);

    _Offload_shared_free(data);
    return 0;
}
```

サンプルコード 4: _Cilk_shared および _Cilk_offload キーワードと動的割り当ての使用 (C/C++)

このほかにも、暗黙的なメモリー・コピー・モデルの使用例として次のサンプルがあります。

C: /opt/intel/composerxe/Samples/ja_JP/C++/mic_samples 以下の shrd_sampleC および LEO_tutorial

C++: /opt/intel/composerxe/Samples/ja_JP/C++/mic_samples/shrd_sampleCPP

また、『インテル® C++ コンパイラー・ユーザー・リファレンス・ガイド』、『インテル® Fortran コンパイラー・ユーザー・リファレンス・ガイド』も参考にしてください。

『インテル® C++ コンパイラー・ユーザー・リファレンス・ガイド』の「主な機能」 > 「インテル® MIC アーキテクチャー」 > 「インテル® MIC アーキテクチャー向けプログラミング」 > 「共有仮想メモリーを使用したオフロード」 > 「共有仮想メモリーを使用したオフロードコードの制約」セクションに、このプログラミング・モデルに関するいくつかの制約が記載されています。

ネイティブコンパイル

アプリケーションをインテル® Xeon Phi™ コプロセッサーでネイティブ実行することもできます。その場合、コプロセッサーはスタンドアロンのマルチコア・コンピューターと見なされます。ホストシステムでバイナリーをビルドしたら、バイナリーと関連バイナリーおよびデータをインテル® Xeon Phi™ コプロセッサーのファイルシステムにコピーします (または、コプロセッサーが NFS を介して必要なファイルにアクセスできるようにします)。

例:

1. openmp_sample.c を /opt/intel/composerxe/Samples/ja_JP/C++/openmp_samples/ からホーム・ディレクトリーにコピーします。
2. -mmic オプションを指定してアプリケーションをビルドします。

```
icc -mmic -vec-report3 -openmp openmp_sample.c
```

3. バイナリーをコプロセッサーにアップロードします。

```
scp a.out mic0:/tmp/a.out
```

4. アプリケーションで必要なすべての共有ライブラリーをコプロセッサーのディレクトリーにコピーします。ここでは OpenMP* ランタイム・ライブラリーをコプロセッサーの /tmp ディレクトリーにコピーします。

```
scp /opt/intel/composerxe/lib/mic/libiomp5.so mic0:/tmp/libiomp5.so
```

5. ssh を使ってコプロセッサーに接続し、アプリケーションが必要な共有ライブラリー (ここでは、OpenMP* ランタイム・ライブラリー) にアクセスできるように、ローカル・ディレクトリーをエクスポートします。

```
ssh mic0
export LD_LIBRARY_PATH=/tmp
```

6. 適切なスタックサイズが設定されていない場合、このアプリケーションはセグメンテーション違反になります。スタックサイズを変更するには、次のコマンドを実行します。

```
ulimit -s unlimited
```

7. /tmp ディレクトリーに移動し、a.out を実行します。

```
cd /tmp
./a.out
```

インテル® Xeon Phi™ コプロセッサーで利用可能な並列プログラミング・モデル

ホストシステムで利用可能な並列プログラミング・モデルのほとんどは、以下を含めインテル® Xeon Phi™ コプロセッサーでも利用できます。

1. インテル® スレッディング・ビルディング・ブロック (インテル® TBB)
2. OpenMP*
3. インテル® Cilk™ Plus
4. Pthreads*

後続のセクションでは、オフロード拡張により、コードでこれらの並列プログラミング・モデルを使用する方法を述べます。インテル® Xeon Phi™ コプロセッサーでネイティブ実行するコードでは、スレッド数が多いことを除き、特に問題なくホストと同様にこれらの並列プログラミング・モデルを使用できます。

インテル® Xeon Phi™ コプロセッサーで利用可能な並列プログラミング・モデル: OpenMP*

ホスト CPU の OpenMP* スレッドとインテル® Xeon Phi™ コプロセッサーの OpenMP* スレッド間で通信は発生しません。オフロード/プラグマ内の OpenMP* 並列領域は 1 つの単位としてオフロードされ、オフロード・コンパイラーは、インテル® Xeon Phi™ コプロセッサーで利用可能なリソースに応じてスレッドチームを作成します。OpenMP* 構文全体がインテル® Xeon Phi™ コプロセッサーで実行されるため、構文内では、共有データおよびプライベート・データに対し通常の OpenMP* セマンティクスが適用されます。

いつでも、複数のホスト CPU スレッドがインテル® Xeon Phi™ コプロセッサーにオフロードできます。CPU スレッドがインテル® Xeon Phi™ コプロセッサーへのオフロードを試み、コプロセッサーに利用可能なリソースがない場合、オフロードコードはホストで実行されます。コプロセッサー上のスレッドは omp parallel 宣言子に到達すると、コプロセッサーで利用可能なリソースに応じてスレッドチームを作成します。作成可能なハードウェア・スレッドの理論的な最大数は、インテル® Xeon Phi™ コプロセッサーのコア数の 4 倍です。実際には、1 つ目のコアが uOS とそのサービス用に予約されるため、これよりも 4 少なくなります (オフロードコードの場合)。

次のサンプルコードは、オフロード構造で OpenMP* を使用し、1 つのホスト CPU スレッドでリダクション・コードをインテル® Xeon Phi™ コプロセッサーにオフロードします。

```
float OMP_reduction(float *data, int size)
{
    float ret = 0;
    #pragma offload target(mic) in(size) in(data:length(size))
    {
        #pragma omp parallel for reduction(+:ret)
        for (int i=0; i<size; ++i)
        {
            ret += data[i];
        }
    }
    return ret;
}
```

サンプルコード 5: オフロード・リダクション・コードでの OpenMP* の使用 (C/C++)

```
real function FTNReductionOMP(data, size)
    implicit none
    integer :: size
    real, dimension(size) :: data
    real :: ret = 0.0

!dir$ omp offload target(mic) in(size) in(data:length(size))
!$omp parallel do reduction(+:ret)
    do i=1,size
        ret = ret + data(i)
    enddo
!$omp end parallel do

    FTNReductionOMP = ret
    return
end function FTNReductionOMP
```

サンプルコード 6: オフロード・リダクション・コードでの OpenMP* の使用 (Fortran)

インテル® Xeon Phi™ コプロセッサーで利用可能な並列プログラミング・モデル: OpenMP* + インテル® Cilk™ Plus の配列表記

次のサンプルは、OpenMP* とインテル® Cilk™ Plus の配列表記を併用する方法を示します。各スレッドは、インテル® Cilk™ Plus 配列表記のビルトイン・リダクション関数 `__sec_reduce_add()` により、インテル® MIC アーキテクチャーの 32 個の 512 ビット・ベクトル・レジスターをすべて使って、配列要素をレデュースします。

```
float OMPnthreads_CilkPlusEAN_reduction(float *data, int size)
{
    float ret=0;
    #pragma offload target(mic) in(data:length(size))
    {
        int nthreads = omp_get_max_threads();
        int ElementsPerThread = size/nthreads;
        #pragma omp parallel for reduction(+:ret)
        for(int i=0;i<nthreads;i++)
        {
            ret =_sec_reduce_add(
                   data[i*ElementsPerThread:ElementsPerThread]);
        }
        // 配列の残りの要素
        for(int i=nthreads*ElementsPerThread; i<size; i++)
        {
            ret+=data[i];
        }
    }
    return ret;
}
```

サンプルコード 7: Open MP* とインテル® Cilk™ Plus を併用する配列のリダクション (C/C++)

インテル® Xeon Phi™ コプロセッサーで利用可能な並列プログラミング・モデル: インテル® Cilk™ Plus

デフォルトでは、インテル® Cilk™ Plus のヘッダーファイルはターゲット環境で利用できません。インテル® Cilk™ Plus を使うインテル® MIC アーキテクチャー向けアプリケーションでこれらのヘッダーファイルを利用するには、次のように *#pragma offload_attribute(push,target(mic))* と *#pragma offload_attribute(pop)* でヘッダーファイルをラップします。

```
#pragma offload_attribute(push,target(mic))
#include <cilk/cilk.h>
#include <cilk/reducer_opadd.h>
#pragma offload_attribute(pop)
```

サンプルコード 8: ヘッダーファイルをラップ (C/C++)

次のサンプルでコンパイラーは、効率良い分割統治法により *cilk_for* ループを再起呼び出し関数に変換します。

```
float ReduceCilk(float*data, int size)
{
    float ret = 0;
    #pragma offload target(mic) in(data:length(size))
    {
        cilk::reducer_opadd<int> total;
        cilk_for (int i=0; i<size; ++i)
        {
            total += data[i];
        }
        ret = total.get_value();
    }
    return ret;
}
```

サンプルコード 9: *cilk_for* ループを再起呼び出し関数に変換

インテル® Xeon Phi™ コプロセッサーで利用可能な並列プログラミング・モデル: インテル® スレッディング・ビルディング・ブロック (インテル® TBB)

インテル® Cilk™ Plus と同様に、デフォルトでは、インテル® TBB のヘッダーファイルはターゲット環境で利用できません。インテル® Cilk™ Plus と同様の方法で、これらのヘッダーファイルをインテル® MIC アーキテクチャー・ベースのターゲット環境で利用できるようにします。

```
#pragma offload_attribute (push,target(mic))
#include "tbb/task_scheduler_init.h"
#include "tbb/blocked_range.h"
#include "tbb/parallel_reduce.h"
#include "tbb/task.h"
#pragma offload_attribute (pop)

using namespace tbb;
```

サンプルコード 10: インテル® TBB ヘッダーファイルをラップ (C/C++)

オフロード構造内で呼び出される関数とインテル® Xeon Phi™ コプロセッサーで必要なグローバルデータには、関数属性 `__attribute__((target(mic)))` を追加します。

例えば、*parallel_reduce* は、分割コンストラクターを使用して再帰的に配列をサブ範囲に分割し、各スレッドに 1 つ以上のコピー (作業) を割り当てます。そして、各分割ごとに、`join` メソッドを呼び出して結果を集計します。

1. コプロセッサー向けのバージョンを生成する場合は、プリフィクスとして各クラスに `__MIC__` マクロを、クラス名に `__attribute__((target(mic)))` を追加します。

```
#ifdef __MIC__
class __attribute__((target(mic))) ReduceTBB
{
private:
    float *my_data;
public:
    float sum;

    void operator()( const blocked_range<size_t>& r )
    {
        float *data = my_data;
        for( size_t i=r.begin(); i!=r.end(); ++i)
        {
            sum += data[i];
        }
    }

      ReduceTBB( ReduceTBB& x, split) : my_data(x.my_data), sum(0) {}

      void join( const ReduceTBB& y) { sum += y.sum; }

      ReduceTBB( float data[] ) : my_data(data), sum(0) {}
};
#endif
```

サンプルコード 11: インテル® MIC アーキテクチャー向けコードを生成するため
インテル® TBB クラスにプリフィクスを追加 (C/C++)

2. インテル® Xeon Phi™ コプロセッサーへオフロードする関数に、プリフィクス `__attribute__((target(mic)))` を追加します。

```
__attribute__((target(mic)))
float MICReductionTBB(float *data, int size)
{
    ReduceTBB redc(data);
    // ライブラリーの初期化
    task_scheduler_init init;
    parallel_reduce(blocked_range<size_t>(0, size), redc);
    return redc.sum;
}
```

サンプルコード 12: インテル® MIC アーキテクチャー向けコードを生成するため インテル® TBB 関数にプリフィクスを追加 (C/C++)

3. `#pragma offload target(mic)` を指定して、インテル® TBB の並列コードをコプロセッサーにオフロードします。

```
float MICReductionTBB(float *data, int size)
{
    float ret(0.f);
    #pragma offload target(mic) in(size) in(data:length(size)) out(ret)
    ret = _MICReductionTBB(data, size);
    return ret;
}
```

サンプルコード 13: インテル® TBB コードをコプロセッサーへオフロード (C/C++)

注: オフロードで使用するインテル® TBB コードは、`-ltbb` の代わりに `-tbb` を指定してビルドします。

インテル® MKL の使用

オフロードする場合、インテル® MKL はよくネイティブ・アクセラレーション (NAcc) モードが使用されます。NAcc では、すべてのデータとバイナリーがインテル® Xeon Phi™ コプロセッサー上に配置されます。データは、オフロード・コンパイラー・プラグマとオフロード領域内またはオフロード関数内のインテル® MKL 呼び出しで使用されるセマンティクスを用いて、プログラマーによって転送されます。NAcc では、BLAS、LAPACK、FFT、VML、VSL、(スパース行列ベクトル) と、必要なインテル® MKL サービス関数を利用できます。最適化されている関数、サポートされていない関数などの詳細は、インテル® MKL のリリースノートを参照してください。

NAcc モードは、インテル® MIC アーキテクチャー向けのネイティブコードでも使用できます。その場合、実行前にインテル® MKL 共有ライブラリーをインテル® Xeon Phi™ コプロセッサーにコピーする必要があります。

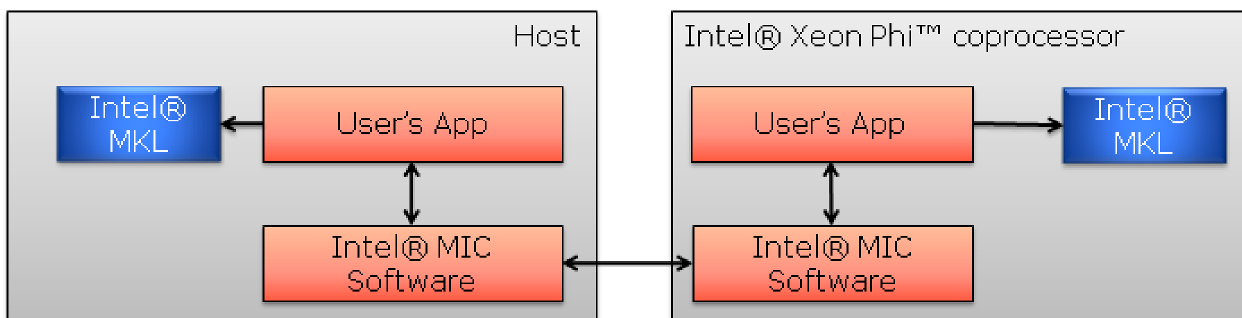

図 3.1: オフロードを使用するインテル® MKL のネイティブ・アクセラレーション

SGEMM サンプル

BLAS ライブラリーから SGEMM ルーチンを使用します。

サンプルコード – sgemm

ステップ 1: 行列を初期化します。このサンプルでは、データが保持されるように、行列をグローバル変数にする必要があります。

ステップ 2: `#pragma offload` を指定して、インテル® Xeon Phi™ コプロセッサーにデータを転送します。このサンプルでは、`free_if(0)` 修飾子を使って、インテル® Xeon Phi™ コプロセッサーでデータが保持されるようにします。

```
#define PHI_DEV 0
#pragma offload target(mic:PHI_DEV) \
    in(A:length(matrix_elements) free_if(0)) \
    in(B:length(matrix_elements) free_if(0)) \
    in(C:length(matrix_elements) free_if(0))
    {
    }
```

サンプルコード 14: インテル® Xeon Phi™ コプロセッサーへのデータ転送

ステップ 3: オフロード領域内で sgemm を呼び出し、インテル® Xeon Phi™ コプロセッサーでインテル® MKL の NAcc バージョンを使用します。`nocopy()` 修飾子により、ステップ 2 でコピーしたデータを再利用します。

```
#pragma offload target(mic:PHI_DEV) \
    in(transa, transb, N, alpha, beta) \
    nocopy(A: alloc_if(0) free_if(0)) nocopy(B: alloc_if(0) free_if(0)) \
    out(C:length(matrix_elements) alloc_if(0) free_if(0)) // output data
    {
        sgemm(&transa, &transb, &N, &N, &N, &alpha, A, &N, B, &N,
            &beta, C, &N);
    }
```

サンプルコード 15: オフロード領域内での sgemm の呼び出し

ステップ 4: ステップ 2 でカードにコピーしたメモリーを解放します。オフロード領域の開始時に `alloc_if(0)` 修飾子でカードにあるデータを再利用し、終了時に `free_if(1)` 修飾子でカード上のデータを解放します。

```
#pragma offload target(mic:PHI_DEV) \
        in(A:length(matrix_elements) alloc_if(0) free_if(1)) \
        in(B:length(matrix_elements) alloc_if(0) free_if(1)) \
        in(C:length(matrix_elements) alloc_if(0) free_if(1))
        {
        }
```

サンプルコード 16: コピーしたメモリーの解放

ほかのプラットフォームでインテル® MKL を使用する場合と同様に、オフロードコード内でインテル® MKL 関数を実行する前に、許容する OpenMP* スレッドの数を設定することで使用するスレッドの数を制限できます。

```
#pragma offload target(mic:PHIDEV) \
    in(transa, transb, N, alpha, beta) \
    nocopy(A: alloc_if(0) free_if(0)) nocopy(B: alloc_if(0) free_if(0))
    out(C:length(matrix_elements) alloc_if(0) free_if(0)) // output data
    {
        omp_set_num_threads(64); // set num threads in openmp
        sgemm(&transa, &transb, &N, &N, &N, &alpha, A, &N, B, &N,
            &beta, C, &N);
    }
```

サンプルコード 17: omp_set_num_threads() を使用してインテル® Xeon Phi™ コプロセッサーのスレッド数を制御

インテル® MKL の自動オフロードモデル

ホスト用のインテル® MKL 関数のいくつかは自動オフロードに対応しており、ホストで通常通り呼び出すことができます。しかし、ライブラリー呼び出しの前に `mkl_mic_enable()` 呼び出しがある場合、インテル® MKL は実行時に自動で問題サイズ、プロセッサーとコプロセッサーの負荷、その他のメトリックを考慮して、呼び出しを完了するのに必要な作業の一部またはすべてを、ホストとインテル® Xeon Phi™ コプロセッサー間で分配すべきかどうかを決定します。この機能は、`mkl_mic_disable()` で無効にできます。

自動オフロードは、`_Cilk_offload` または `#pragma offload` によりインテル® Xeon Phi™ コプロセッサーで実行されるコードとは別に、選ばれたインテル® MKL ライブラリー呼び出しにのみ適用されます。そのため、自動オフロード呼び出しと `_Cilk_offload` または `#pragma offload` によりコプロセッサーで実行されるコードの両方で、同じデータの転送を最小限に抑える必要があります。現在、自動オフロードと、プログラマーによって (`_Cilk_offload` または `#pragma offload` を介して) 制御される明示的なオフロード間の共通データをコプロセッサー上に保持する方法はありません。

自動オフロードの制御方法を示すサンプルは、`<install-dir>/opt/intel/composerxe/mkl/examples/mic_ao/blasc` (C コード) と `/opt/intel/composerxe/mkl/examples/mic_ao/blasf` (Fortran コード) があります。

インテル® Xeon Phi™ コプロセッサーでのデバッグ

インテル® MIC アーキテクチャー向けアプリケーションに関する情報は、`/opt/intel/composerxe/Documentation/en_US/debugger/debugger_documentation.htm` (英語) の「Debugging with the Intel® Debugger on Eclipse*」セクションと「Debugging on the Command Line」セクションにあります。

インテル® Xeon Phi™ コプロセッサーでのパフォーマンス解析

インテル® VTune™ Amplifier XE 2013 Linux* 版を使って、インテル® Xeon Phi™ コプロセッサーのパフォーマンス・データを収集する方法は、`/opt/intel/vtune_amplifier_xe_2013/documentation/help/index.htm` (英語) の「Getting Started」> 「Intel Xeon Phi Coprocessor Analysis Workflow」セクションを参照してください。

著者紹介

Sudha Udanapalli Thiagarajan
2008 年にインドのアンナ大学チェンナイでコンピューター・サイエンスの学士号を、2010 年にクレムゾン大学でコンピューター工学の修士号を取得しています。2010 年からインテルでイネーブリング・アプリケーション・エンジニアとして、ISV アプリケーションの最適化とインテル® MIC アーキテクチャーの販促資料の制作に取り組んでいます。

Charles Congdon
インテル コーポレーションのソフトウェア & サービスグループのシニア・ソフトウェア・エンジニア。アプリケーションのパフォーマンスとスケーラビリティーの向上に取り組んでおり、社内外のプロジェクトでソフトウェア開発とドキュメント制作を行っています。前職はオラクルのコンサルティング・ソフトウェア・エンジニアで、DEC Alpha プロセッサー上の Oracle* RDBMS Windows* NT バージョンと OpenVMS* バージョンの並列化および 64 ビット対応に携わっていました。

Sumedh Naik

2009 年にインドのムンバイ大学で電子工学の学士号を、2012 年にクレムゾン大学でコンピューター工学の修士号を取得しています。2012 年からインテル コーポレーションでソフトウェア・エンジニアとして、インテル® Xeon Phi™ コプロセッサーの販促資料の制作に取り組んでいます。

Loc Q Nguyen
ダラス大学で MBA を、マギル大学で電気工学の修士号を、モントリオール理工科大学で電気工学の学士号を取得しています。現在は、インテル コーポレーションのソフトウェア & サービスグループのソフトウェア・エンジニアで、コンピューター・ネットワーク、コンピューター・グラフィックス、並列処理を研究しています。

著作権と商標について

本資料に掲載されている情報は、インテル製品の概要説明を目的としたものです。本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスも許諾するものではありません。製品に付属の売買契約書『Intel's Terms and Conditions of Sale』に規定されている場合を除き、インテルはいかなる責任を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証 (特定目的への適合性、商品適格性、あらゆる特許権、著作権、その他知的財産権の非侵害性への保証を含む) に関してもいかなる責任も負いません。

「ミッション・クリティカルなアプリケーション」とは、インテル製品がその欠陥や故障によって、直接的または間接的に人身傷害や死亡事故が発生するようなアプリケーションを指します。そのようなミッション・クリティカルなアプリケーションのためにインテル製品を購入または使用する場合は、直接的か間接的かにかかわらず、あるいはインテル製品やそのいかなる部分の設計、製造、警告にインテルまたは委託業者の過失があったかどうかにかかわらず、製造物責任、人身傷害や死亡の請求を起因とするすべての賠償請求費用、損害、費用、合理的な弁護士費用をすべて補償し、インテルおよびその子会社、委託業者および関連会社、およびそれらの役員、経営幹部、従業員に何らの損害も与えないことに同意するものとします。

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ないでください。

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

最新の仕様をご希望の場合や製品をご注文の場合は、お近くのインテルの営業所または販売代理店にお問い合わせください。
本資料で紹介されている資料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 (http://www.intel.com/design/literature.htm) を参照してください。

Intel、インテル、Intel ロゴ、Xeon、Xeon Phi、Cilk、VTune は、アメリカ合衆国および / または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。

© 2013 Intel Corporation. 無断での引用、転載を禁じます。

パフォーマンスに関する注意事項

* パフォーマンスおよびベンチマークの結果に関する詳細は、www.intel.com/benchmarks (英語) を参照してください。

最適化に関する注意事項

最適化に関する注意事項

インテル® コンパイラーは、互換マイクロプロセッサー向けには、インテル製マイクロプロセッサー向けと同等レベルの最適化が行われない可能性があります。これには、インテル® ストリーミング SIMD 拡張命令 2 (インテル® SSE2)、インテル® ストリーミング SIMD 拡張命令 3 (インテル® SSE3)、ストリーミング SIMD 拡張命令 3 補足命令 (SSE3) 命令セットに関連する最適化およびその他の最適化が含まれます。インテルでは、インテル製ではないマイクロプロセッサーに対して、最適化の提供、機能、効果を保証していません。本製品のマイクロプロセッサー固有の最適化は、インテル製マイクロプロセッサーでの使用を目的としています。インテル® マイクロアーキテクチャーに非固有の特定の最適化は、インテル製マイクロプロセッサー向けに予約されています。この注意事項の適用対象である特定の命令セットに関する詳細は、該当する製品のユーザー・リファレンス・ガイドを参照してください。

改訂 #20110804