この記事は、インテル® デベロッパー・ゾーンに掲載されている「Programming for Multicore and Many-core Products including Intel® Xeon® processors and Intel® Xeon Phi™ coprocessors」の日本語参考訳です。

インテル® Xeon® プロセッサーおよびインテル® Xeon Phi™ コプロセッサーを含むマルチコア / メニーコア製品向けのプログラミング

インテル® Xeon® プロセッサーおよびインテル® Xeon Phi™ コプロセッサーを含むマルチコア / メニーコア製品向けのプログラミング (インテル® Xeon Phi™ コプロセッサーへのオフロード用言語拡張を含む)

概要

マルチコア・プロセッサーで利用されているプログラミング・モデルは、メニーコア・コプロセッサーでも利用可能です。このため、インテル® Xeon® プロセッサーおよびインテル® Xeon Phi™ コプロセッサーのプログラミング方法を説明するには、並列プログラミングの方式について説明することがベストでしょう。この記事では、抽象的、直感的、効率的なユニファイド・プログラミング・アプローチを利用するプログラムがマルチコア・プロセッサーとメニーコア・コプロセッサーに適している理由を理解するための基本的な情報を提供します。このアプローチは、最新のアプリケーションに簡単に適用でき、優れた結果が得られる、シンプルかつ自然なアプローチです。インテルのメニーコア・プロセッサーおよびマルチコア・コプロセッサーに備わるインテル® アーキテクチャー命令と組み合わせると、直感的ではないアプローチよりも、高度な並列コンピューティングで高いパフォーマンスを簡単に達成できます。

マルチコア・プロセッサーとメニーコア・コプロセッサーを利用するプログラムには、常に変化するニーズを満たすさまざまな方式が用意されています。これらの方式は、C、C++、Fortran、OpenMP*、MPI、および 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) のような、広く採用されている既存のソリューションを利用しており、OpenCL* のような新しい標準規格やインテル® Cilk™ Plus のような新しいオープンソース・ソフトウェアによる開発も急速に進んでいます。

はじめに

世界中のプロセッサーにおけるシングルコア・プロセッサーの割合は減少を続けており、並列コンピューティングが可能なマルチコア・プロセッサーに置き換えられています。将来のコンピューティングは並列コンピューティングであり、将来のプログラミングは並列プログラミングです。

マルチコア・プロセッサーを利用する手法はこの数年で大いに進化し、かつてないほど多くの優れた選択肢が開発者に提供されています。インテル® TBB の人気は高まり、業界を挙げての OpenCL* の採用とサポートも進んでいます。

同時に、マルチコア・プロセッサーやプログラミング手法も一般的になりつつあり、インテルでは、インテル® アーキテクチャーの長所を犠牲にすることなく、この進化に対応したメニーコア・プロセッサーをリリースしています。メニーコア・プロセッサーで新しく追加された機能は自然かつ直感的な手法で、インテルのマルチコア・プロセッサーと同じツール、プログラミング言語、プログラミング・モデル、実行モデル、メモリーモデルなどを利用することができます。

この記事は、広く採用されたソリューションや新しい標準規格に焦点を当てながら、マルチコア・プロセッサーとメニーコア・プロセッサーで利用可能なプログラミング手法について説明します。

現在の並列プログラミング

マルチコア・プロセッサーとメニーコア・コプロセッサーの両方でインテル® アーキテクチャーを利用する目標は、直感的で一般的なプログラミング手法であるため、まず現在のマルチコア向け並列プログラミングを確認し、将来の方向性を理解しておくことが重要です。インテル® アーキテクチャーの基本は共通であるため、メニーコア・プロセッサー向け並列プログラミングの基本も正確に定義できます。

ライブラリー

ライブラリーは、プログラミングを始める前に考慮すべき重要な抽象的並列プログラミング手法を提供します。BLAS、ビデオやオーディオのエンコーダー/デコーダー、高速フーリエ変換 (FFT)、ソルバー/ソーターを含むライブラリーによるアルゴリズムの実装を検討することは重要です。インテル® マス・カーネル・ライブラリー (インテル® MKL) のようなライブラリーは、インテル® ストリーミング SIMD 拡張命令 (インテル® SSE)、インテル® アドバンスト・ベクトル・エクステンション (インテル® AVX)、マルチコア・プロセッサー、およびメニーコア・コプロセッサーを活用するようにチューニングされた、多くのアルゴリズムの高度な実装を提供しています。インテル® MKL は、線形代数パッケージ (LAPACK) の Fortran と C の業界標準インターフェイスをサポートしており、インテル® MKL のルーチンへの呼び出しをプログラムに追加することにより、これらの実装を利用できます。多くの開発者は、標準規格にインテルのハイパフォーマンスな実装が組み合わされたこのようなライブラリーを、並列プログラミングで最初に利用する対象として選択しています。

ライブラリーで特定のプログラミング・ニーズが解決されない場合、開発者はプログラミング言語によるコードを記述します。

従来の主なプログラミング言語はどれも、並列プログラミング用に設計されていませんでした。その結果、新しいプログラミング言語と既存言語の拡張について多くの提案が行われました。最終的に、多くの提案の中から、C、C++ および Fortran による並列プログラミングにには複数のソリューションが採用されました。

並列プログラミングで最も広く使用されている抽象化は、OpenMP* (主に C と Fortran)、インテル® TBB (主に C++)、MPI (C、C++、Fortran) です。これらは、さまざまなプロセッサーと

オペレーティング・システムをサポートしており、汎用的で信頼性の高いプログラミングの選択肢として利用されています。

また、開発者は、オペレーティング・システムのネイティブ・スレッド手法を利用することもできます。POSIX* スレッド (Pthreads) および Windows* スレッドなどのこれらのインターフェイスは、高水準プログラミングの抽象化を含まない、フル・コントロールを可能にする低水準インターフェイスを提供します。これらのインターフェイスは、並列プログラミングにおけるアセンブリー言語プログラミングと言えるでしょう。開発者はこれらのインターフェイスを利用して、あらゆる操作が可能ですが、現在はより高い抽象化への移行が進み、アセンブリー言語のようなプログラミングが行われることはほとんどありません。同様に、マルチコア・プロセッサーの普及により、スレッドを直接制御することを回避する傾向が加速されました。“スレッド” ではなく “タスク” によるプログラムへの移行は、抽象化プログラミング・モデルによってサポートされた、プログラミング習慣の根本的かつ重大な変化と言えます。

現在採用されているほとんどの並列プログラミングは、並列処理で最もポピュラーな 3 つの抽象化 (OpenMP*、MPI、インテル® TBB) の 1 つ、あるいはオペレーティング・システムの低水準スレッド・インターフェイスのいずれかにより行われています。

これらの標準規格は進化を続けており、新しいさまざまな手法が提案されています。現在、これらの進化の技術的な推進力は、高度なデータ並列ハードウェアと最新のコンパイラー・テクノロジーに向けられています。これらの推進力はどちらも、より高い水準の抽象化でプログラミングして開発者の生産性を向上させ、開発と保守コストを削減するという強い要望が基になっています。

最も構成可能な並列プログラミング・モデル

詳細: http://Intel.com/go/parallel

上記で説明した理由から、最も構成可能な並列プログラミング手法はインテル® TBB およびインテル® Cilk™ Plus です。これらの手法は、パフォーマンスを向上させてプログラミングへの投資を保護する、効率良い抽象化プログラミングの長所を備えています。また、効率良い並列プログラム向けに選択および組み立て可能な再結合コンポーネントを提供します。これらの手法は個別に習得することもできますが、簡単に利用できる機能のコレクションとして考えるのが最適です。これはライブラリーを含むモジュールの組み合わせに構成の容易性 (コンポーザビリティー) を提供するため、非常に重要です。どちらの手法も、スレッド化と OpenMP* にはない自己構成容易性を備えています。OpenMP* および OpenCL* は構成の容易性による制限がありますが、低水準スレッドよりも適しています。インテル® アーキテクチャーのマルチコア・プロセッサーとメニーコア・コプロセッサーの長所の 1 つは、これらの手法をすべて強力にサポートし、現在および将来のプログラミング・ニーズに最も適したソリューションを提供できることです。

OpenMP*

詳細: http://openmp.org/wp/

1996 年に、コンパイラーによって並列ハードウェアの利用を支援する方法として、OpenMP* 標準規格が提案されました。それから 15 年以上が経過し、今ではすべての主要な C、C++、およ

び Fortran コンパイラーが OpenMP* をサポートしています。OpenMP* は、特に Fortran と C で記述された科学プログラムに適しています。インテルは OpenMP* 作業部会のメンバーであり、OpenMP* の実装および支援ツールを提供する主要ベンダーです。OpenMP* は、マルチコアおよびメニーコア・プログラミングに利用できます。

OpenMP* ドット積の例 ― Fortran:

```
!$omp do
   do j = 1, n
      adotb = adotb + a(j) * b(j)
    end do
!$omp end do
```

OpenMP* 和 (リダクション) の例 ― C:

```
#pragma omp parallel for reduction(+: s)

for (int i = 0; i < n; i++)
   s += x[i];
```

今後、OpenMP* 仕様は “オフローディング” や “アクセラレーティング” と呼ばれるコンピューティングの新しいコントロールを標準化するべく拡張されるでしょう。現在、インテルは、“オフロード言語拡張” (LEO) と呼ばれる OpenMP* 非標準の拡張機能を提供しています。OpenMP* 委員会は、インテル® Xeon Phi™ コプロセッサーと GPU の両方で利用できる機能をサポートするため、LEO と GPU 向けの非 OpenMP* オフロード指示句のセットである OpenACC を調査しています。

インテル® TBB

詳細: http://threadingbuildingblocks.org/

インテルは 2006 年にインテル® TBB をリリースし、2007 年にインテル® TBB のオープンソース・プロジェクトを開始しました。2009 年には、インテル® TBB を利用する開発者の数は OpenMP* の開発者の数を超えるまでになりました (Evans Data Corp のリサーチ: http://www.evansdata.com/research/market_alerts.php および研究レポートに基づく)。インテル® TBB は、C++ 開発者のニーズに合わせて設計されたものです。OpenMP* は、C および Fortran 開発者のニーズに合わせて設計されているため、インテル® TBB と OpenMP* の間に実質的な競合はありません。同じプログラムで OpenMP* とインテル® TBB を併用できることも、C++ 開発者にとって注目すべき点です。

並列関数呼び出しの例 ― C++ (インテル® TBB を使用):

```
parallel_for (0, n,

   [=](int i) {
   Foo(a[i]);
});
```

コンパイラーによる支援を直接必要としないインテル® TBB の登場は、タスク・プログラミングの価値を高め、タスクスチール・システムが広く受け入れられるきっかけとなりました。コンパイラーは並列プログラミングへの取り組みを支援するために発展を続け、インテル® Cilk™ Plus プロジェクトが開始されました。コンパイラーによる並列プログラミングの利用が増加したことで、並列化の真価が発揮されるようになりました。インテルは、現在でもインテル® TBB オープンソース・プロジェクトの主要な貢献者であり、インテル® TBB のサポートおよび支援ツールを提供する主要なサプライヤーです。インテル® TBB は、マルチコアおよびメニーコア・プログラミングに利用できます。

MPI

詳細: http://Intel.com/go/mpi

クラスター (プロセッサーはメッセージを渡すために接続されるが常にメモリーを共有するとは限らない) を使用する開発者にとって、メッセージ・パッシング・インターフェイス (MPI) は最も一般的なプログラミング手法です。クラスター間の通信には、ノードにメニーコア・プロセッサーが含まれるかどうかにかかわらず、MPI が引き続き利用できます。

インテルのコプロセッサーは、他のコプロセッサーやマルチコア (インテル® Xeon® プロセッサーなど) のランクと通信可能なランクをサポートしているため、現在の MPI で記述されているプログラムはインテル® Xeon Phi™ コプロセッサー・ベースのシステムに容易に移行できます。インテル® Xeon Phi™ コプロセッサーは、マルチコア・プロセッサーと同様に、開発者が必要とする数のランクを作成します。これらのランクは、マルチコア・プロセッサーやメニーコア・プロセッサー上にあるかどうかにかかわらず、ほかのランクと通信できます。

インテル® Xeon Phi™ コプロセッサーは汎用プロセッサーであるため、MPI ジョブはコプロセッサー上で実行されます。既存の MPI プログラムと同じ結果を得るためにアルゴリズムの変更やリファクタリングが不要なため、非常に強力です。コプロセッサーの一般的な能力をインテル® Xeon Phi™ コプロセッサーの強力な MPI サポートと組み合わせることで、開発者にとって直感的な方法で結果を得ることができます。

広く利用されているインテル® MPI ライブラリーは、事実上すべての相互接続でハイパフォーマンスなサポートを提供します。インテル® MPI ライブラリーは、今日の MPI プログラミングに合致する方法により、マルチコア・プロセッサーとメニーコア・コプロセッサーにランクを作成するシステム環境でマルチコアとメニーコアの両方をサポートします。

インテル® Xeon Phi™ コプロセッサー上で MPI は、マルチコア・プロセッサー・ベースのシステムで一般的となったスレッドモデル (OpenMP*、インテル® TBB、インテル® Cilk™ Plus など) との組み合わせも可能です。

インテルは MPI 実装とツールを提供する主要ベンダーです。MPI は、マルチコアおよびメニーコア・プログラミングに利用できます。

並列プログラミングの新しい標準規格

データ並列ハードウェアでは、C、C++ に特定の拡張をサポートすることで、開発者の生産性に焦点を当てた重要な選択肢を提供します。

インテル® Cilk™ Plus

詳細: http://cilk.com

インテルは 2010 年後半にインテル® Cilk™ Plus を発表しました。Cilk は、米マサチューセッツ工科大学 (M.I.T.) で研究開発された後、Cilk Arts 社により Cilk++ としてリリースされました。インテルは、インテル® Cilk™ Plus として Linux* 、Windows* および OS* X 版のコンパイラーにタスクスチールを実装しました。インテルでは、インテル® Cilk™ Plus の完全な仕様を公開して、ほかの実装や、API コンプライアンスによるインテルのランタイム機能の利用や変換可能なランタイムの構築が可能になるように支援しています。より多くのコンパイラーでサポートされるよう、インテルは他のコンパイラー・ベンダーと協力して取り組んでいます。インテルは、インテル® Cilk™ Plus およびツールを提供する主要なサポーターであることを誇りに思っています。

インテル® Cilk™ Plus は、3 つの新しいキーワード、特殊なリダクション操作のサポート、データ並列拡張を提供します。次の例で、関数 fib が後続のコードと同時に実行できることを示すには、cilk_spawn キーワードを、x = cilk_spawn fib(n-1) のように関数呼び出しの前に記述します。cilk_sync キーワードは、関数からのスポーンがすべて完了するまで後続の実行を待機することを示します。関数をスポーンの 1 つの単位として使うことでコードが分かりやすくなり、ベースラインの C/C++ 言語に依存して変数のスコープ規則を定義し、インテル® Cilk™ Plus プログラムを構成可能にできます。

再帰フィボナッチ計算の並列スポーン ― C (インテル® Cilk™ Plus を使用):

```
int fib (int n) {

   if (n < 2) return 1;
   else {
      int x, y;
      x = cilk_spawn fib(n-1);
      y = fib(n-2);
      cilk_sync;
      return x + y;
   }
}
```

Cilk は、C および C++ 開発者に直感的で効率良いコンパイラーによるサポートを提供します。Cilk は習得が簡単で、広く採用が進むことが期待されています。内部ループの依存関係がない通常の “for” ループは、キーワード “for” を “cilk_for” に変更するだけで並列ループに変換できます。これは、ループの反復間に順序がないことをコンパイラーに伝えます。

並列関数呼び出し ― C (インテル® Cilk™ Plus を使用):

```
cilk_for (int i=0; i<n; ++i){
```

```
        Foo(a[i]);
}
```

Cilk を利用する開発者は、スレッドに対応するメモリー・アロケーターやソートルーチンのように、新しいコンパイラー・キーワードや最適化が不要な特定のアルゴリズムや機能にはインテル® TBB を併用できます。インテル® Cilk™ Plus は、マルチコアおよびメニーコア・プログラミングに利用できます。

C / C++ データ並列拡張

詳細: http://cilk.com

現在、データ並列拡張機能を提供するため、C (および C++) を拡張する方法について議論が行われています。実装、経験、そして採用は標準化への重要なステップです。インテルは、インテル® Cilk™ Plus (Linux* 版、Windows* 版、OS* X 版) の一部として、基本的なデータ並列向けの拡張を実装しました。より多くのコンパイラーでサポートされるよう、インテルは他のコンパイラー・ベンダーと協力して取り組んでいます。Fortran 90 の配列操作に似た直感的な構文拡張は、インテル® Cilk™ Plus の重要な要素として提供され、配列の操作を簡略化します。C/C++ 言語では、配列の演算を表現する方法は提供されていません。開発者はループを記述し、明示的なシリアル順序を作成して、配列の要素をアクセスして演算を表現する必要があります。構文拡張を利用して要素単位の加算を示すには、a[:] = b[:] + c[:]; と記述します。不要なシリアル順序を指定する必要はありません。この簡略化されたセマンティクスにより、コンパイラーは常に最適なベクトルコードを生成します。この配列操作の構文拡張は、配列表記と呼ばれます。

意図しないシリアル化を回避する手法により、開発者は C/C++ の標準表記でスカラー関数を記述し、“要素関数” として宣言することができます。要素関数を宣言すると、コンパイラーは関数を 1 セットの引数で操作する代わりに、ベクトルレジスターとベクトル命令を用いたショートベクトルの引数で操作が行えるように、その関数のショートベクトル・バージョンを生成します。関数内の制御フローがデータ値に依存しないケースでは、ショートベクトル関数を実行すると結果のベクトルを取得できます。同時に、通常の関数が実行されると単一の結果を取得できます。

要素関数 ― C (インテル® Cilk™ Plus を使用):

```
__declspec (vector) void saxpy(float a, float x, float &y)
{
   y += a * x;
}
```

インテルは、製品およびツールでこれらの C および C++ 向け構文拡張をサポートしています。また、より広いサポートのため、他のコンパイラー・ベンダーと協力して取り組んでいます。C、C++ およびデータ並列拡張は、マルチコアおよびメニーコア・プログラミングで利用できます。

OpenCL*

詳細: http://Intel.com/go/opencl

OpenCL* は、最初にアップル社により提案された後、業界標準化団体 (クロノスグループ) に移管されました。インテルはこの団体に参加しています。OpenCL* は、“ハードウェアに近い” インターフェイスを備え、幅広い業界の関心と取り組みによって生み出された重要な抽象化と十分な制御を提供します。OpenCL* は、この記事で説明するソリューションの中で最もリファクタリングが必要になります。特に、ハードウェアの深い知識に基づいたリファクタリングが必要です。リファクタリング作業の結果は、マルチコアおよびメニーコアのパフォーマンスで顕著に現れます。ただし、パフォーマンスはリファクタリングを行わない場合でも得られることがあります。OpenCL* の目標は、リファクタリングへの取り組みを生産性の高い方法で行うことです。OpenCL* 以外のソリューションでは、(ハードウェアの深い知識に基づいたときに最適に行われる) リファクタリングを回避する代わりの手法を提供します。

OpenCL* による簡単な要素単位の乗算:

```
kernel void
   dotprod(  global const float *a,
             global const float *b,
             global float *c)
   {
      int myid = get_global_id(0);
      c[myid] = a[myid] * b[myid];
   }
```

インテルは OpenCL* 標準化団体の主要メンバーで、現在利用可能な実装を備えたソリューションと関連ツールを提供するベンダーでもあります。OpenCL* プログラム内のコードは通常、各ターゲット用に個別に複製されますが、OpenCL* は、マルチコア、メニーコアおよび GPU プログラミングに利用できます。インテルは現在、インテル・マルチコア・プロセッサー (インテル® SSE 命令およびインテル® AVX 命令を使用) およびインテル® HD グラフィックス (多くの第 3 世代インテル® Core™ プロセッサーの一部として利用可能な統合グラフィックス) の両方で OpenCL* をサポートしています。

複数のモデルを利用する構成の容易性

構成の容易性は重要な概念です。異なるニーズに合わせて複数の並列プログラム手法を使う場合、これらの手法が相互に排他的ではないことが不可欠です。上記で説明した抽象化プログラミング手法は、アプリケーションで組み合わせて利用することができます。構成可能なプログラミングをサポートする、より新しいプログラミング・モデルを提供することにより、プログラマーはアプリケーション開発において、複数のプログラミング手法を組み合わせる煩わしく、非生産的な制限から解放されます。

最も構成可能な手法は、インテル® TBB およびインテル® Cilk™ Plus (C/C++ データ並列拡張を含む) です。OpenMP* と OpenCL* は構成の容易性による制限がありますが、低水準スレッドよりは適しています。

インテル® TBB およびインテル® Cilk™ Plus は、効率良い並列プログラム開発において、選択および組み合わせ可能な再結合コンポーネントを提供します。これはライブラリーを含むモジュー

ルの組み合わせに構成の容易性を提供するため、非常に重要です。どちらの手法も、スレッド化と OpenMP* や OpenCL* にはない自己構成容易性を備えています。

メニーコアの利用

マルチコアとメニーコアを組み合わせて利用することにより、さまざまな可能性が広がります。

インテル® Xeon Phi™ コプロセッサーは、高度なプログラミングを可能にしたまま、高度な並列ワークを高い電力効率で処理できるように設計されています。マルチコア・プロセッサーとインテル® Xeon Phi™ コプロセッサーの両方を含むプラットフォームは、*ヘテロジニアス*・プラットフォームと呼ばれます。ヘテロジニアス・プラットフォームは、汎用のシリアルおよび並列ワークロードを制御できるマルチコア・プロセッサーの柔軟性と、高度な並列ワークロード用に特化したメニーコア・プロセッサーの処理能力の両方を備えています。ヘテロジニアス・プラットフォームでは、データのコピーを管理し、コントロールを転送するプログラミング・モデルを利用します。

アプリケーションは単一ソースコードからビルドできます。インテル® アーキテクチャーのマルチコア・プロセッサーとメニーコア・コプロセッサーの汎用性により、直感的で効率良いプログラミングが可能です。

メニーコアの明示的な利用と暗黙的な利用

メニーコア・プロセッサーは、メニーコア・プロセッサーを検出しそれを利用するコードにより、インテル® MKL のようなライブラリーを通じて暗黙的に利用できます。開発者はインテルのライブラリーに備わった明示的なコントロールを利用できますが、マルチコア・プロセッサーの利用を自動判定するライブラリーに任せたシンプルなアプローチも効果的です。

ソースコードで開発者が明示的に指示する場合、追加のプログラミングが可能です。メニーコアをアプリケーションが明示的に利用するには、ヘテロジニアス・プログラムを記述します。具体的には、並列アプリケーションを記述して、マルチコア・プロセッサーとメニーコア・コプロセッサー間のワークを分割します。

明示的なコントロールを含む場合でも柔軟に対応できるように (メニーコア・プロセッサーが存在しない場合でも動作し、また将来の利用に備えるように) インテルは拡張機能を実装しています。これらの 2 つの点は重要です。まず、単一のソースプログラムで、インテル® Xeon Phi™ コプロセッサーにオフロードするように指示できます。ただし、実行時にコプロセッサーがシステムに存在しない場合でも、マルチコア・プロセッサーとメニーコア・コプロセッサーの両方でインテル® アーキテクチャーを使用できるよう、コプロセッサーにオフロードするコードをどちらの種類のプロセッサーでもシームレスに実行できることを意味します。

オフロード

最新のハードウェアは、マルチコア・プロセッサーに 1 つ (クラスターの場合は複数)、各メニーコア・プロセッサーに 1 つの個別のメモリー空間を含む、ヘテロジニアス・プラットフォームです。マルチコア・プロセッサーとメニーコア・コプロセッサー間の接続は、考慮すべきボトルネックです。

これらのヘテロジニアス・プラットフォームを利用するには 2 つのアプローチがあります。1 つ目のアプローチは、メモリー空間を分離して扱い、オフロード指示句を使って、マルチコア・プロセッサーとの間でコントロールとデータを移動します。2 つ目のアプローチは、単一システムまたはクラスターの単一ノードに存在するマルチコア・プロセッサーとメニーコア・プロセッサー間の共有を可能にする、MYO と呼ばれる共有メモリー・ソフトウェア管理機能によってデータ処理を簡略化します。MYO は “Mine Yours Ours” の頭文字で、現在のアクセス権と所有権を決定するためにシステム内のメモリーを共有するソフトウェア抽象化を指します。

1 つ目のアプローチでは、マルチコア・プロセッサーとメニーコア・プロセッサーはメモリーを共有しません。この実行モデル用の指示句のコンパイラー・サポートは、システムの低レベルな制御からプログラマーを解放しますが、ターゲットがヘテロジニアスであることを意識しなければならず、より複雑な問題を解決するのに時間を費やさなければいけません。

簡単なオフロード ― Fortran:

```
!dir$ offload target(MIC1)
!$omp parallel do
      do i=1,10
         A(i) = B(i) * C(i)
      enddo
!$omp end parallel
```

プログラマーは、オフロード用プラグマ (#pragma offload) を追加して、後続する言語構造をインテル® Xeon Phi™ コプロセッサーで実行するように指示できます。オフロードされたコードを実行する前にプロセッサーとコプロセッサー間でデータをコピーする必要がある場合、プラグマに節を追加します。結果をマルチコア・プロセッサーのメモリーにコピーし戻すデータを指定することも可能です。オフロード用プラグマは、C、C++、および Fortran で利用できます。

簡単なオフロード ― C (データ転送を使用):

```
float *a, *b; float *c;

#pragma offload target(MIC1)
   in(a, b : length(s))
   out(c : length(s) alloc_if(0))
for (i=0; i<s; i++) {
   c[i] = a[i] + b[i];
}
```

データのコピーに代わるアプローチは、MYO と呼ばれるランタイム・ユーザー・モード・ライブラリーです。MYO は、マルチコア・プロセッサーとインテル® Xeon Phi™ コプロセッサー間のデータの同期と、同じ仮想アドレスへのデータの割り当てを可能にするコンパイラー・サポートを有効にします。つまり、マルチコアとメニーコアのメモリー空間でデータを共有できるということです。ツリー、リンクリスト、その他のポインターベースのデータ構造のコピーは全面的にサポートされます。データのマーシャリングは必要ありません。MYO の機能を利用するには、_Cilk_shared キーワードを追加して両側から見えるようにデータ項目をマークし、オフロードによりインテル® Xeon Phi™ コプロセッサーでワークを実行します。ステートメント x = _Offload func(y); は、関数 func()

をインテル® Xeon Phi™ コプロセッサー上で実行し、戻り値を変数 x に割り当てます。関数はその実行の一部として共有データを読み取りまたは変更し、変更された値はすべてのプロセッサーで実行しているコードから参照することができます。

オフロードアプローチは明示的で、一部のプログラムには適しています。MYO アプローチはより暗黙的で、いくつかの長所があります。MYO では、C++ クラスをマーシャリングしてコピーすることなくクラスをコピーできます。この機能はオフロードプラグマではサポートされません。重要なのは、MYO は同期の際にすべての共有変数をコピーしないことです。代わりに、2 つの同期ポイント間で変更された値のみコピーします。

これらのオフロード・プログラミング手法は、コントロールがメニーコア・プロセッサーにワークを指示できるように設計されていますが、マルチコアとメニーコア・プログラミングに利用できます。コードの移植性は高く、システムが変更された場合でも長く使用できます。メニーコア・プロセッサーを搭載しているシステムと搭載していないシステムでソースコードを変更する必要はありません。

その他のオフロード機能

キーワードとプラグマによるメカニズムはどちらも、インテル® Xeon Phi™ コプロセッサー上のワークを呼び出すことによりデータのコピーを実行します。データのコピー中にほかのワークをスケジューリングできるように、指示句オプションで計算を呼び出す前に非同期にデータのコピーを開始できる機能が実装されています。

多くのシステムは 1 ノードあたり複数のマルチコア・プロセッサーと 1 つ以上のインテル® Xeon Phi™ コプロセッサーで構成されるため、追加の言語サポートやプラグマにより、プログラマーがオフロードの対象を選択できる機能も実装されています。このオプションを利用することで、環境に応じて、よりダイナミックかつ最適な決定が下せるでしょう。

標準規格

マルチコアおよびメニーコアの両方でインテル® アーキテクチャー命令を利用することにより、プログラミング・ツールとモデルを両方で活用できます。さまざまな洞察と正しいモデルにより、直感的で効率良い手法でマルチコア・プロセッサー、ヘテロジニアス・システムおよび将来のシステムでの利用に適した単一ソースベースを構築できます。単一ソースベースは既存のプログラミング言語で構築可能です。

C、C++、Fortran、OpenMP*、MPI、インテル® TBB を含む最適なソリューションは、インテル® アーキテクチャーのマルチコアシステムおよびメニーコアシステムに適用できます。

インテル® Cilk™ Plus、オフロード拡張および OpenCL* などの新しい機能はインテルにより強力にサポートされ、広く採用が進むことが期待されており、将来もサポートされる予定です。

標準化への道は、強力な製品と仕様の公開、ユーザー (顧客) への浸透、そして他のベンダーによるサポートから始まり、実現可能な標準規格へとつながります。OpenCL*、インテル® Cilk™ Plus、オフロード拡張は、製品サポートと仕様が存在し、顧客の利用が順調に進んでいます。より

広いサポートとユーザーの経験に基づいて洗練された標準規格が続くと期待することは理に適っているでしょう。

まとめ

インテル® アーキテクチャーと業界標準のプログラミング・ツールは、マルチコアおよびメニーコア・プロセッサーの両方に適用できる並列プログラミング手法を提供します。これらの手法は、使い慣れたツール、プログラミング言語、既存のソースコードを使った、単一のソースコードを利用できます。最新のソリューションにより、マルチコア・プロセッサーとメニーコア・コプロセッサーの両方で最適な単一コードセクションでアプリケーションを構築することができます。マルチコアおよびメニーコア並列化に利用できる手法は、将来への投資を保護し直感的なプログラム手法を提供しつつ、優れたパフォーマンスを実現します。

標準規格はプログラミング手法で重要な役割を果たします。インテルは、標準プログラミング・モデルと手法のサポートおよび実装に、これまで多額の投資を行っています。さらに、インテルは新しい課題を克服するべく標準規格の発展に対し重要な役割を果たしています。

インテル® Xeon Phi™ コプロセッサー向けのプログラミングを行うことで、同じコードでマルチコアシステムをフルサポートする、移植可能で将来のアーキテクチャーでもスケーラブルな、インテル® Xeon Phi™ コプロセッサーの能力を引き出すアプリケーションを開発することができます。

複数のノードで構成されたマルチコア・プロセッサーとメニーコア・コプロセッサーのクラスターは、直感的かつ効率良い手法で、インテル® アーキテクチャーで利用可能なツールとプログラミング・モデルの豊富な機能を活用できます。

既存の開発ツール、標準規格、パフォーマンスを活用する能力とその柔軟性は、インテルのマルチコア/メニーコア・ソリューションを比類なきソリューションにしています。

著者紹介

James Reinders。インテル コーポレーションのソフトウェア・エバンジェリスト兼ディレクター。

1989 年にインテル コーポレーションに入社した後、シストリック・アレイ・システム WARP および iWarp、世界初のテラ FLOPS スーパーコンピューター (ASCI Red)、世界初のテラ FLOPS シングルチップ・コンピューティング・デバイス (最初のインテル® Xeon Phi™ コプロセッサー、当時の名称は Knights Corner) を含むさまざまなプロジェクトに貢献してきました。また、インテル® プロセッサーと並列システム向けのコンパイラーおよびアーキテクチャー関連作業にも取り組んできました。さらに、ソフトウェア開発製品の主要プロバイダーとしてインテルの地位を確立するためこれまで尽力してきました。現在は、ソフトウェア開発製品部門のチーフ・エバンジェリストを務めています。O'Reilly Media から出版された『Intel® Threading Building Blocks: Outfitting C++ for Multicore Processor Parallelism』(日本語:『インテル スレッディング・ビルディング・ブロック ―マルチコア時代の C++ 並列プログラミング』) の著者です。この書籍は日本語のほかに、中国語、韓国語の翻訳版があります。2012 年に Morgan Kaufmann から出版された『Structured

Parallel Programming』(日本語:『構造化並列プログラミング』) の共著者でもあります。このほかにも、さまざまな記事を執筆しており、多くの書籍に寄与しています。ミシガン大学で電子情報工学の学士号およびコンピューター工学の修士号を取得しています。

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を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。

インテルによる書面での合意がない限り、インテル製品は、その欠陥や故障によって人身事故が発生するようなアプリケーションでの使用を想定した設計は行われていません。

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 サイトを参照してください。

Intel、インテル、Intel ロゴ、Cilk、Xeon、Xeon Phi 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。

© 2012 Intel Corporation. 無断での引用、転載を禁じます。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。