このセッションには音声が入っています。
スピーカーの音量に注意してください。

スライドの画面をクリックするか、
プレイヤーの再生ボタンを押すとセッションが始まります。

音量に注意してください。調整が終わったら、スタートしてください。

このスライドに音声はありません

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

インテル® Xeon Phi™ コプロセッサー Code Modernization セミナー

インテル® Xeon Phi™ コプロセッサー向けソフトウェア開発 (約 30 分)

Part 3 プログラミング・モデル

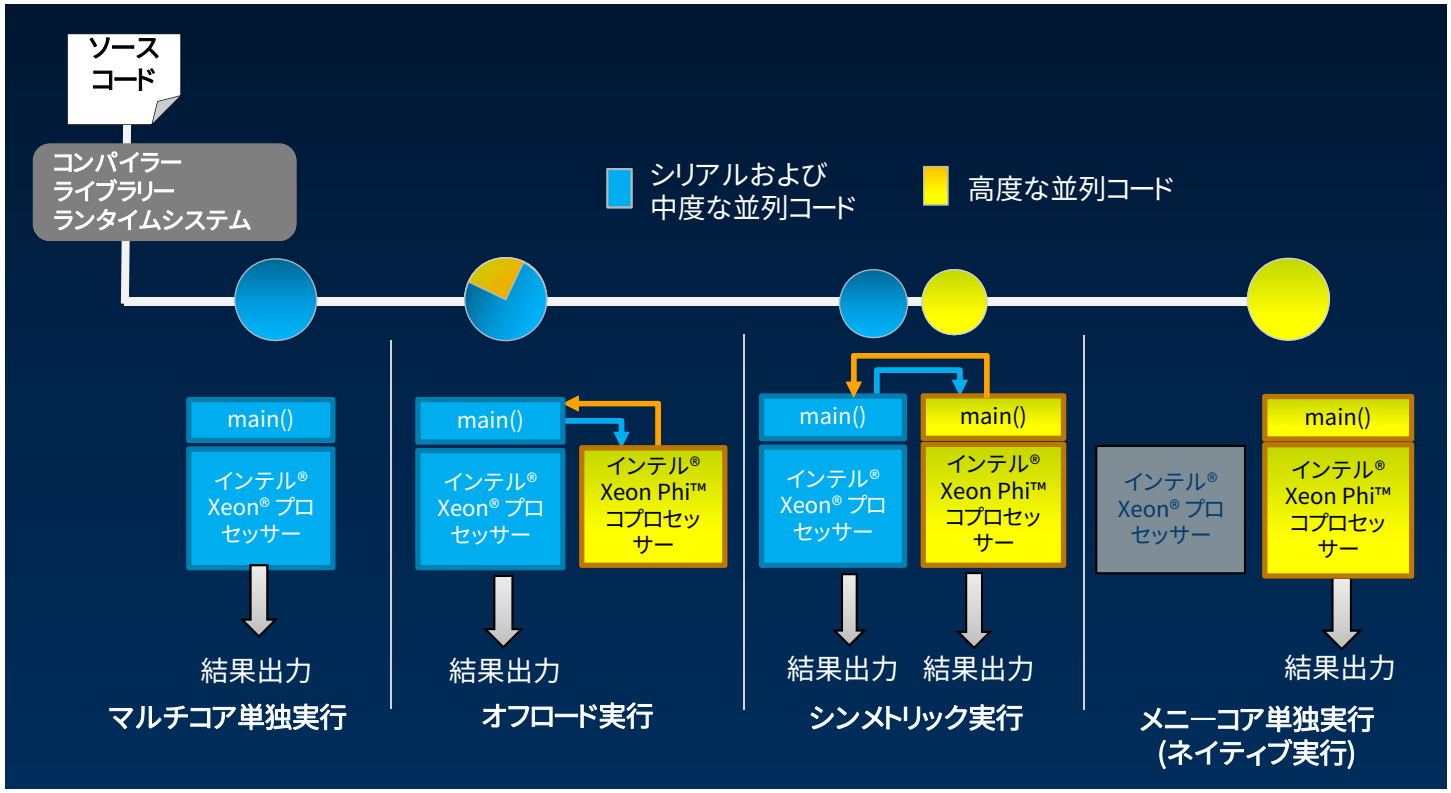
2015 年 10 月

内容

- インテル® Xeon Phi™コプロセッサーに適したアプリケーション領域とプログラミング環境 (Part 1)
- インテル® Xeon Phi™ コプロセッサーのソフトウェア開発をサポートする開発ツール (Part 2)
- インテル® Xeon Phi™ コプロセッサーのプログラミング (Part 3)

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

実行モデルの概要



©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

ヘテロジニアス・プログラミングについて

- CPU とコプロセッサーの命令セットは似ているが同じではない
- CPU とコプロセッサーはメインメモリーを共有しない

「オフロード実行」と「ネイティブ実行」

- **オフロード実行**: プログラムの特定の領域をインテル® Xeon Phi™ コプロセッサーにオフロードして実行するモデル
- **ネイティブ実行**: プログラム全体をインテル® Xeon Phi™ コプロセッサーで実行するモデル

インテル® コンパイラーは、

- CPU とコプロセッサー両方のバイナリーを生成できる
- オフロード実行を記述する言語拡張が提供される
- オフロード機能を定義する OpenMP* 4.0 & 4.1 仕様をサポート

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

ヘテロジニアス・プログラミング
オフロード実行

オフロード実行モデルの概要

オフロードするデータの転送方法には、以下の 2 つのタイプがある

非共有メモリー型

- プログラマーが転送するデータを “offload” プラグマや指示句を使用して明示的に指定する
- ビット単位でコピー可能なスカラー、配列、構造体のような、フラットなデータ構造を扱う場合に適している
  - C/C++ の例: #pragma offload target(mic) in(data:length(size))
  - Fortran の例: !dir$ offload target(mic) in(a1:length(size))

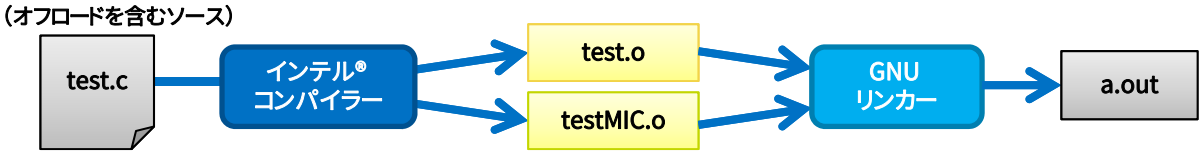
仮想共有メモリー型

- ホストとコプロセッサー間で仮想共有するデータを宣言することでランタイムが暗黙的にデータの同期を行う
- リンクリストやツリーのような、複雑なポインターベースのデータ構造を扱う場合に適している
- C/C++ のみサポート (Fortran は非サポート)
  - C/C++の例: _Cilk_shared double foo; _Cilk_offload func(y);

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

オフロード実行モデルの概要 (続き)

- インテル・コンパイラーはオフロードセクションに対し、ホスト CPU 用と
  インテル® Xeon Phi™ コプロセッサー用の 2 つのオブジェクト・ファイルを生成する



- インテル® Xeon Phi™ コプロセッサーが利用できない場合、オフロードセクションはホスト CPU で実行される。ただし以下のようにターゲットデバイスを明示的に指定した場合はランタイムエラーとなる;

  “target(mic:#)” または “_Cilk_offload_to(coproc_num)”

- オフロードの開始、終了、データ転送、同期処理はコンパイラーおよびランタイム・ライブラリーによって自動管理される
- C/C++ 言語、Fortran 言語、Pthreads*、インテル® MKL、OpenMP*、インテル® TBB、インテル® Cilk™ Plus、MPI などでプログラミング可能

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

非共有型オフロードコードの例 (PI の計算)

```
#include <stdio.h>
#include <stdlib.h>
#define INTERVALS 1000000000

int main(void){
    int i;
    double x, pi=0.0;
    double Step = 1.0 / INTERVALS;

 #pragma offload target(mic)
#pragma omp parallel for private(x) reduction(+:pi)
    for (i=0; i<INTERVALS; ++i)
    {
        x = Step * ((double)i-0.5);
        pi += 4.0 / (1.0 + x*x);
    }

    pi = Step * pi;
    printf("Pi = %lf¥n", pi);
}
```

オフロード宣言子の追加

オフロード
セクション

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

非共有型オフロード宣言子 (インテル・コンパイラー拡張)

	C/C++ シンタックス	意味
オフロードプラグマ	#pragma offload <指示句> <ステートメント>	次のステートメントを インテル® MIC またはCPUで実行
変数 / 関数用 キーワード	__attribute__((target(mic))) もしくは、 __declspec(target(mic))	CPU およびインテル® MIC 用に変数のアロケート、関数のコンパイルを行う
コードブロック用 プラグマ	#pragma offload_attribute(push, target(mic)) #pragma offload_attribute(pop)	CPU およびインテル® MIC 用にファイル全体またはコードブロックを指定

	Fortran シンタックス	意味
オフロード宣言子	!dir$ offload <指示句> <ステートメント>	次のステートメントを インテル® MICまたはCPUで実行
コードブロック用 宣言子	!dir$ attributes offload:<mic> :: <ret-name> もしくは <var1,var2,...>	CPU およびインテル® MIC 用に関数／変数コンパイル

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

非共有型オフロード宣言子 (インテル・コンパイラー拡張)

節 / 修飾子	シンタックス	意味
ターゲット指定	target(name[:#])	実行するインテル® MIC カードの指定
条件付きオフロード	if (condition)	Boolean 表現
入力	in (var-list modifiersopt)	ホストからインテル® MIC へコピー
出力	out (var-list modifiersopt)	インテル® MIC からホストへコピー
入力および出力	inout (var-list modifiersopt)	ホストからインテル® MIC へコピーしオフロード後ホストへコピー
コピーしないデータ	nocopy (var-list modifiersopt)	データはインテル® MIC に存在
修飾子	**シンタックス**	**意味**
ポインター長を指定	length (element-count-expr)	複数のポインター要素をコピー
メモリー割当て制御	alloc_if (condition)	条件が真ならメモリー割り当て
メモリー解放制御	Free_if (condition)	条件が真ならメモリー解放
アライメント制御	align (expression)	アライメントの指定

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

非共有型オフロード修飾子の例

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

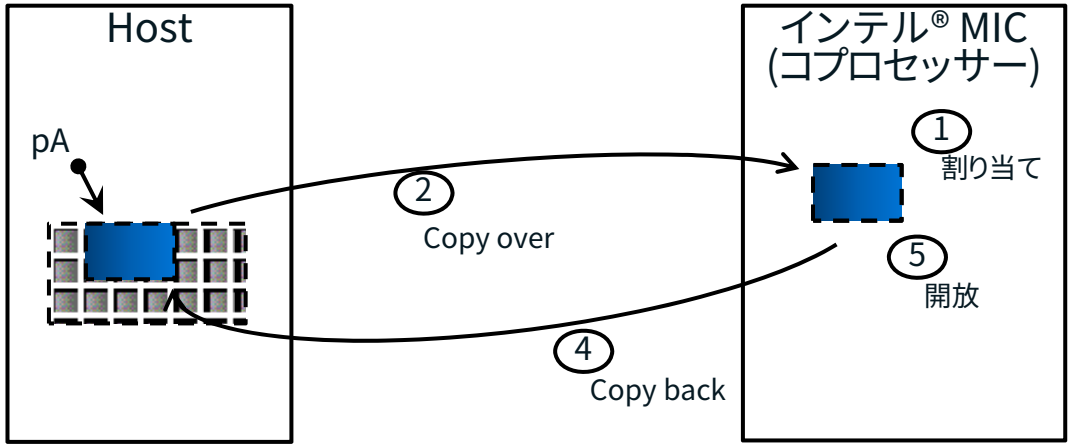
ノート: “length”修飾子に指定する値は要素数でありバイト数ではない。コンパイラーはデータ型をすでに知っている

Intel Compiler 16.0 Intel(R) 64 Visual Studio 2012

```
len.cpp
len.cpp(4): エラー: このオフロード領域のポインター変数 "data" は in/out/inout/no
copy 句で指定する必要があります。
    #pragma offload target(mic) // in(data:length(numberOf))
    ^
```

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

非共有型オフロードによるデータ転送イメージ



#pragma offload 文で指定される in/out 変数のデフォルト動作

- オフロードの開始時:
  ① インテル® MIC (コプロセッサー) に領域が確保される
  ② in / inout 変数は Host からインテル® MIC に転送される
- オフロードの終了時:
  ④ out / inout 変数はインテル® MIC から Host に転送される
  ⑤ インテル® MIC 上の in および out 変数の領域が解放される (inout 変数についても同様)

3

```
#pragma offload inout(pA:length(n))
{...}
```

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

OpenMP* 4.0 による非共有型オフロードのサポート

インテル® C/C++ および Fortran コンパイラーがサポートするオフロード拡張のサブセット

節 / 修飾子	シンタックス	意味
ターゲット指定	omp target [data][節」	実行と [データ] をデバイスへ転送
デバイス指定	device(スカラー式)	ターゲットデバイスを指定
結果の取得	omp target update [節]	ターゲットとホスト間のデータを再転送

節	シンタックス	意味
条件付きオフロード	if (condition)	Boolean 表現
入力	map(to:変数)	ホストからターゲットへコピー
出力	map (from: 変数)	ターゲットからホストへコピー
入力および出力	map (tofrom：変数)	ホストからターゲット へコピーしオフロード後ホストへコピー
メモリー割り当て	map (alloc：変数)	ターゲット上にメモリーを割り当て（初期値は未定義）

http://www.isus.jp/article/openmp-special/openmp4-0_summarycard_for_compiler15-0/

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

OpenMP* 4.0 のオフロード例

```
#pragma omp target data device(0) map(alloc:tmp[:N]) map(to:input[:N]) map(from:res)
  {
#pragma omp target device(0)
#pragma omp parallel for
    for (i=0; i<N; i++)
      tmp[i] = some_computation(input[i], i);

    update_input_array_on_the_host(input);

#pragma omp target update device(0) to(input[:N])

#pragma omp target device(0)
#pragma omp parallel for reduction(+:res)
    for (i=0; i<N; i++)
      res += final_computation(input[i], tmp[i], i)
  }
```

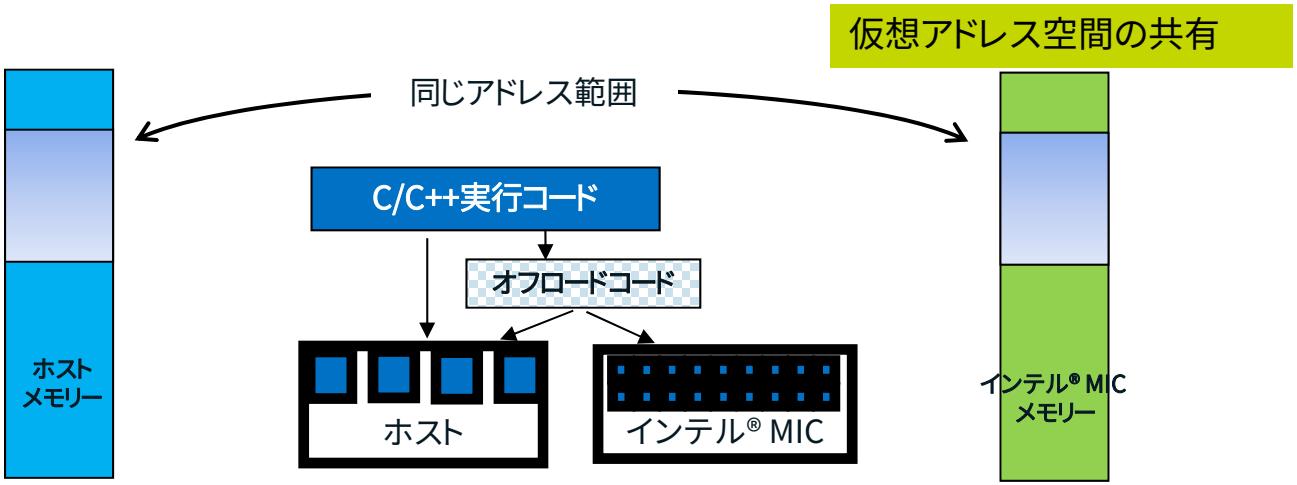
ホスト
ターゲット
ホスト
ターゲット
ホスト

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

仮想共有型オフロードの概要

ホストとインテル® MIC 間で同じ仮想アドレスにメモリーが確保される

- 複雑なポインターを含むデータ構造でも共有可能
- データ管理の煩わしさがない
- C/C++コードに対してシンプルな言語拡張キーワードを利用可能



©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

仮想共有型オフロードの概要 (続き)

◆ 共有メモリーの同期が行われるタイミング
- オフロードの近辺(オフロードの入り口と出口)で自動的に行われる
- 変更されたデータのみが CPU とインテル® MIC 間で転送される

◆ 動的メモリーを共有データとする場合は、以下のような特殊関数を使用する必要がある:
_Offload_shared_malloc、_Offload_shared_aligned_malloc、_Offload_shared_free、_Offload_shared_aligned_free

◆ C++ オブジェクトの転送も可能

◆ ポインターが共有データを指すことができる

◆ オフロードコード内で、共有データに対するアクセス制御やデータ競合の保護には、一般的な手法が適用できる
- 例: ロック、クリティカル・セクション、その他

◆ このモデルはインテル® Cilk™ Plus 並列言語拡張に統合されている

Fortran は非サポート、C/C++ コードのみ利用可能

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

仮想共有型オフロードのキーワード

対象	シンタックス	意味
関数	int _Cilk_shared f(int x) { return x+1; }	CPU とコプロセッサーの両方のバージョンが生成され、どちらか一方からコールされる
グローバル	_Cilk_shared int x = 0;	両方から参照可能
ソースファイル / 静的関数	static _Cilk_shared int x;	ソースファイルもしくは関数内でのみ、両方で参照できる
クラス	class _Cilk_shared x {…};	クラスメソッド、メンバー、オペレーターはすべて両方で利用可能
共有データへのポインター	int _Cilk_shared *p;	P はローカルに割り当てられたポインターで、共有データを指すことができる
共有ポインター	int *_Cilk_shared p;	P は共有ポインター（共有データのみ指すことができる）
コード内の特定範囲	#pragma offload_attribute(push, _Cilk_shared) #pragma offload_attribute(pop)	ソースファイル全体またはコード内の特定範囲をこのプラグマで囲み、_Cilk_shared とする

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

仮想共有型オフロードのキーワード (続き)

機能	シンタックス	意味
関数呼び出しをオフロードする	x = _Offload func(y);	func は、インテル® MIC で実行される
	x = _Offload_to (card_number) func(y);	func は指定されたコプロセッサーで実行されなければならない
非同期にオフロードする	x = _Cilk_spawn _Offload func(y);	非ブロッキングのオフロード

並列 for ループをオフロードする	_Offload _Cilk_for(i=0; i<N; i++)
	{
	a[i] = b[i] + c[i];
	}
		ループはインテル® MICで並列に実行される。
		ループは、関数呼び出しとして暗黙に定義される

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

仮想共有型オフロードコードの例 (PI の計算)

```
void findpi()
{
  int count = 10000;

  // Initialize shared global
  // variables
  pi = 0.0f;

  //  Compute pi on target
  _Offload compute_pi(count);

  pi /= count;
}
```

```
// Shared variable declaration for pi
_Cilk_shared float pi;

// Shared function declaration for
// compute
_Cilk_shared void compute_pi(int count)
{
    int i;

    #pragma omp parallel for ¥
          reduction(+:pi)
    for (i=0; i<count; i++)
    {
        float t = (float)((i+0.5f)/count);
        pi += 4.0f/(1.0f+t*t);
    }
}
```

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

ヘテロジニアス・プログラミング
ネイティブ実行

ネイティブ実行モデルの概要

◆ インテル® Xeon Phi™ コプロセッサー上で直接実行されるモデル

- ホスト上で、コンパイラー・オプション “-mmic (/Qmic)” を指定してコンパイルする (既存コードもそのまま再コンパイル)
- コンパイルした実行バイナリー、関連データファイル、必要ランタイム・ライブラリーは scp や ftp でインテル® Xeon Phi™ コプロセッサーに事前にコピーする必要がある
- ベクトル化、インテル® MKL、OpenMP*、インテル® TBB、インテル® Cilk™ Plus、インテル® MPI などが利用可能
- プログラムは高度に並列化されていることが望まれる。コプロセッサー上 でのシリアル実行はホスト CPU より遅くなる
- アプリケーション実行の際、ホストとメモリー量が異なることを考慮する必要がある

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

ネイティブ実行モデルのコンパイル/実行例

コンパイルと実行準備の例 (Linux*):

```
$ icc -mmic -openmp omp_app.cpp
$ su
# scp a.out root@mic0:/tmp/
a.out 100% 111KB 111.5KB/s 00:00
# scp /opt/intel/composerxe/lib/mic/libiomp5.so root@mic0:/tmp/
libiomp5.so 100% 939KB 939.2KB/s 00:01
#
```

コプロセッサーへのログインとプログラムの実行例 (Linux*):

```
# ssh root@mic0
[~]# cd /tmp
[tmp]# ls
a.out libiomp5.so
[tmp]# export LD_LIBRARY_PATH=/tmp
[tmp]# ./a.out Testarg
A big OpenMP hello to Testarg from 90 threads!
```

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

ネイティブ実行モデルのコンパイルと実行例 (micnativeloadex ツールを利用)

```
$ icc -mmic -openmp omp_app.cpp
$
$ export SINK_LD_LIBRARY_PATH=/opt/intel/composerxe/lib/mic
$
$ /opt/intel/mic/bin/micnativeloadex a.out
```

- SINK_LD_LIBRARY_PATH 環境変数にインテル® MIC 用のランタイム・ライブラリーのディレクトリーを設定することで、micnativeloadex が自動で必要な依存ライブラリーをコプロセッサーにコピーする
- “micnativeloadex a.out -l” で依存ライブラリーをチェック可能

“micnativeloadex”ツールでネイティブ実行作業を簡略化できる

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

インテル® Parallel Studio XE Composer Edition
(インテル® MIC アーキテクチャー用コンパイルオプションとマクロ)

オプション	意味
-mmic (/Qmic)	インテル® MIC アーキテクチャー用ネイティブコードを作成する
-no-offload (/Qoffload-)	コード内のオフロード宣言子を無視する
-opt-report-phase=offload (/Qopt-report-phase:offload)	ホストとターゲット間でのオフロードデータ入出力情報を表示する
-offload-attribute-target=mic (/Qoffload-attribute-target:mic)	ソースファイルを有効範囲とする変数や関数をオフロード可能属性とする
-offload-option (/Qoffload-option)	ターゲットツールに渡す特定オプションを指定する (compiler, ld, link, as など) 例) -offload-option,mic,compiler, "-O3 –vec-report2"
–watch=mic-cmd (/watch:mic-cmd)	コプロセッサー用コンパイルに渡されるオプションリストを表示する

マクロ	意味
__INTEL_OFFLOAD	CPU およびコプロセッサー用コンパイル時に定義されるマクロ。 -no-offload (/Qoffload-) や –mmic (/Qmic)オプションで未定義となる
__MIC__	コプロセッサー用コンパイル時に定義されるマクロ -no-offload (/Qoffload-) オプションで未定義となる

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

インテル® Parallel Studio XE Composer Edition
(インテル® MIC アーキテクチャー用環境変数とAPI)

環境変数	意味
OFFLOAD_REPORT	オフロード実行時間や転送データ情報等を出力する 設定値は、1、2、3 のいづれか 例) OFFLOAD_REPORT=2
OFFLOAD_DEVICES	利用可能なコプロセッサーを指定する。コプロセッサー指定番号は、0 から開始 例) OFFLOAD_DEVICES = 0,2
KMP_AFFINITY=balanced	インテル® MIC アーキテクチャー用 OpenMP* アフィニティ設定
KMP_PLACE_THREADS	OpenMP* で、使用するコア数、コアあたりのスレッド数等を設定する 例1) 最初の30コアとコアあたり2スレッド使用の場合: KMP_PLACE_THREADS=30C,2T 例2) オフセット30から30コア使用の場合: KMP_PLACE_THREADS=30C,30O

API	意味
_offload_number_of_devices()	コプロセッサーの数を取得する
_offload_get_device_number()	そのプログラムを実行しているコプロセッサーの番号を取得する

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

パフォーマンスガイドライン

インテル® Xeon Phi™ コプロセッサー 超並列プログラミング

ノード内のパフォーマンスを改善するならインテル® Xeon Phi™ コプロセッサーを利用する ;

1. アプリケーションのスケーリングは、インテル® Xeon Phi™ コプロセサーの高度な並列機能を利用する準備は出来ているか ?
2. ベクトルユニットを最大限に活用しているか ?
3. インテル® Xeon® プロセッサーよりも多くのローカルメモリー帯域を利用できるか ?

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

3 つの基本パフォーマンス概念

◆ スレッド/タスクの並列化

- マルチスレッドによる並列化
- 粒度、ロードバランス、バリア、フォルス・シェアリングの考慮

◆ SIMD 演算による最適化

● インテル・コンパイラーがベクトル化しやすいコーディングに努める

- データ依存の除去
- 条件分岐、関数コールの除去
- ループ回数の明確化
- タイプ変換の除去
- ベクトル化レポートオプション (-opt-report -opt-report-phase=vec) の活用
- インテル® Cilk™ Plus (配列表記、SIMD 対応関数、SIMD プラグマ) の活用

◆ メモリーアクセスに関する最適化 (キャッシュの有効利用)

- AOS (構造体の配列) から SOA (配列の構造体) への変換
- キャッシュ・ブロッキング・アルゴリズムの利用

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

アフィニティーとロードバランスの考慮

- コア単位の 4 つのハードウェア・スレッドをすべて使用しない方が良いケース
  - 複数スレッド間で L1 および L2 キャッシュのリソースを競合し、結果キャッシュ・スラッシング、TLB スラッシングが発生する場合
  - コアの唯一のベクトル演算ユニットへの競合が発生する場合
  - スレッド数を減らしてメインメモリーへのアクセスが減る場合
- コア単位の 4 つのハードウェア・スレッドをすべて使用した方が良いケース
  - データサイズがキャッシュに収まり、スレッド間でデータの局所性による利点が得られる場合
  - コア単位のワークロードが十分小さい場合
- KMP_AFFINITY 環境変数の利用
  - OMP_NUM_THREADS が利用可能スレッド数より小さい場合は特に重要
  - KMP_AFFINITY=compact を適用する時
    - データの局所性が重要である時、ただしロード・インバランスの発生に注意
  - KMP_AFFINITY=scatter を適用する時
    - システムリソースを最大限活用したい時
  - granularity= を使用してさらに細かくスレッドを制御することも可能
  - 不適切なアフィニティー設定は逆にパフォーマンスを低下させる可能性あり
- KMP_PLACE_THREADS 環境変数の利用
  - KMP_PLACE_THREADS=2T などでコア単位のスレッド数を調整する

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

メモリー・アライメントの適用

メモリー・アライメントは、ベクトル演算の高速化には必須

◆ **静的メモリーに対するアライメント**

コンパイラー / リンカーによってアロケートされる

- __attribute__((aligned(n))) や __declspec(align(n)) を変数に付けて宣言する
- グローバル変数、自動スタック変数にも適用する

◆ **動的メモリーに対するアライメント**

ランタイム関数によってアロケートされる

- __mm_aligned_malloc(size, alignment_bytes) でメモリー確保する
- __mm_aligned_free() でメモリー解放する

◆ **インテル® TBB の利用**

- scalable_aligned_malloc() / scalable_aligned_free() を利用する
- <tbb/scalable_allocator.h> を include し、-ltbbmalloc でリンクする

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

日本語情報サイト

http://www.isus.jp/article/idz/mic-developer/

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

関連書籍

Structured Parallel Programming: Patterns for Efficient Computation

著者 Michael McCool, James Reinders ,Arch Robison 出版日: 2012 年 7 月 9 日 | ISBN: 978-0-124159938

『構造化並列プログラミング: 効率良い計算を行うためのパターン』

著者　マイケル・マックール／アーク・D・ロビソン／ジェームス・レインダース（共著）

訳者　菅原 清文／エクセルソフト株式会社（共訳）| ISBN　978-4-87783-305-3

Intel® Xeon Phi™ Coprocessor High Performance Programming

著者 Jim Jeffers , James Reinders 出版日: 2013 年 3 月 | ISBN: 978-0-124104143

『インテル® Xeon Phi™ コプロセッサー ハイパフォーマンス・プログラミング』

著者　ジェームス・レインダース/アーク・D・ロビソン（共著）

訳者　菅原 清文／エクセルソフト株式会社（共訳）| ISBN　978-4-87783-332-9

High Performance Parallelism Pearls

著者 Jim Jeffers , James Reinders　出版日: 2014 年 11 月

簡単にインテル® Xeon Phi™ コプロセッサー・ファミリーの優れた並列性を利用してコードを実行できるため、最適化に集中し、ハイパフォーマンスを実現することが可能です。並列処理を細かくチューニングすることで、正しいアプリケーションを正しく効率良いアプリケーションにすることができます。インテル コーポレーションの並列プログラミング・エバンジェリストである James Reinders とインテル コーポレーションのエンジニアである Jim Jeffers により執筆された最新の書籍は、69 人の専門家の実際の経験を基に、インテルのマルチコアおよびメニーコア・プロセッサーを最大限に利用するための創意工夫を紹介しています。

©2015 Intel Corporation. 無断での引用、転載を禁じます。* 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。

intel®

法務上の注意書きと最適化に関する注意事項

本資料の情報は、現状のまま提供され、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、商品性に関する保証、第三者の特許権、著作権、その他、知的財産権の侵害への保証を含む) をするものではありません。
性能に関するテストに使用されるソフトウェアとワークロードは、性能がインテル® マイクロプロセッサー用に最適化されていることがあります。SYSmark* や MobileMark* などの性能テストは、特定のコンピューター・システム、コンポーネント、ソフトウェア、操作、機能に基づいて行ったものです。結果はこれらの要因によって異なります。製品の購入を検討される場合は、他の製品と組み合わせた場合の本製品の性能など、ほかの情報や性能テストも参考にして、パフォーマンスを総合的に評価することをお勧めします。

© 2015 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴ、Cilk、Intel Xeon Phi、VTune、Xeon 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。

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 (SSSE3) 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の詳細は、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