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™ コプロセッサーのプログラミング例 (約 17 分)

2015 年 10 月

内容

- Windows* 環境におけるプログラミングの注意点
- Windows* アプリケーションのオフロードの例
- デバイスの有無とオフロード
- シンメトリック同時処理

©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。

Windows* 環境におけるプログラミングの注意点

インテル® Xeon Phi™ コプロセッサー上のマイクロ OS は Linux* です;

1. ネイティブもしくはオフロード実行するコード領域では Win32* API は呼び出すことができません
2. オフロードを行う場合、ソースは Windows* 向けのコンパイラーと Linux* 向けのクロスコンパイラーでコンパイルされます
3. インテル® Xeon Phi™ コプロセッサー向けのマクロをうまく使いましょう
4. Linux* 環境で作成されたソースを Windows* 上でコンパイルする場合、言語方言の違いに注意が必要です

©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。

インテル® Xeon Phi™ コプロセッサー上では Win32* API を呼び出すことはできません

```
void FindPrimes(int start, int end){
    // 開始値は常に奇数
     int range = end - start + 1;

#pragma offload target(mic) in(start, end) out(gPrimesFound) inout(gProgress)
#pragma omp parallel for reduction(+:gPrimesFound)
     for( int i = start; i <= end; i+=2 ){
          if( TestForPrime(i) )
               gPrimesFound++;
          InterlockedIncrement((LPLONG) &gProgress);
     }
}
```

```
#pragma omp atomic
     gProgress++;
```

オフロード領域はインテル® Xeon Phi™ コプロセッサー上の Linux* マイクロカーネル環境で実行されるため、Windows* 固有の API を呼び出すことはできません

`InterlockedIncrement((LPLONG) &gProgress);`

©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。

ソースは ネイティブ・コンパイラーとクロスコンパイラーでコンパイルされます

クロスコンパイラーが識別できないインクルードファイルはエラーとなります

```
#include <stdio.h>
#ifndef __MIC__
#include <windows.h>
#endif
main(){

    #pragma offload target(mic)
        printf("I'm MIC¥n");
}
```

インテル® MIC 向けコードのコンパイルに必要がない部分は、マクロを使用して制御します

```
sample1.c
sample1.c(3): 致命的なエラー: *MIC* ソース ファイル "windows.h" を開けません。
  #include <windows.h>
                      ^
```

©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™ コプロセッサー関連マクロ

マクロ名	
__MIC__	コプロセッサーで実行するコードをビルドする際にコンパイラーによって定義されます。このマクロを使用して、コプロセッサー向けにコンパイルされ、コプロセッサー上でのみ実行されるようにコードを保護できます (インテル® MIC アーキテクチャー上でのみ実行可能な組込み関数や zmmintrins.h のような対応するヘッダーファイルなど)。 デフォルトで定義されます。CPU とコプロセッサーで実行するコードをビルドできますが、このマクロはコプロセッサーのコンパイルにのみ定義されます。[Q]offload オプションの否定形を使用した場合の動作は未定義です。 コプロセッサー上で実行するコードをビルドするには、Qmic コンパイラー・オプション (Windows*) または mmic コンパイラー・オプション (Linux*) を使用します。 このマクロは、Qmic (Windows*) または mmic (Linux*) を使用する場合に定義されます。
__INTEL_OFFLOAD	CPU とコプロセッサーで実行するコードをビルドする際にコンパイラーによって定義されます。このマクロと [Q]offload コンパイラー・オプションの否定形、あるいは Qmic (Windows*) または mmic (Linux*) を一緒に使用して、offload.h の omp_set_num_threads_target API の呼び出しのようにオフロードビルドでのみ実行されるホスト上のコードを保護します。 デフォルトで定義されます。CPU とコプロセッサーで実行するコードをビルドします。[Q]offload コンパイラー・オプションの否定形、あるいは Qmic (Windows*) または mmic (Linux*) を使用した場合の動作は未定義です。

©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。

Linux* 環境で作成されたソースを Windows* 上でコンパイルする場合

Linux 向けに作成されたソースコードを Windows* 環境でコンパイルする際の注意点 ;

- ネイティブ・コンパイルする場合は、ソース中の gcc 固有表現や API はそのまま利用できる
- コードの一部分をオフロードする場合、オフロードされる領域以外を Windows* 環境に適合させる必要があります
  - 例 : `float fa[FLOPS_ARRAY_SIZE] __attribute__((align(64)));`

↓

```
__declspec (align(64)) float fa[FLOPS_ARRAY_SIZE];
```

©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。

内容

- Windows* 環境におけるプログラミングの注意点
- Windows* アプリケーションのオフロードの例
- デバイスの有無とオフロード
- シンメトリック同時処理

©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。

Windows* 向けのソースのオフロード例

```
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <windows.h>
#include <process.h>

#define PRIME_END 50000000
static int gProgress = 0, gPrimesFound = 0;
```

```
unsigned WINAPI ShowProgressThread(LPVOID val) {
        int *ip = (int *) val;
        int percentDone = 0, range = *ip;
        while( percentDone < 100 ) {
                percentDone =
(int)((float)gProgress/(float)range *200.0f + 0.5f);
                if( percentDone % 10 == 0 ){
                printf("¥b¥b¥b¥b%3d%%", percentDone);
                Sleep(1);
                }
        }
        return 0;
}
```

```
bool TestForPrime(int val){
    int limit, factor = 3;

    limit = (long)(sqrtf((float)val)+0.5f);
    while( (factor <= limit) && (val % factor))
        factor ++;

    return (factor > limit);
}
```

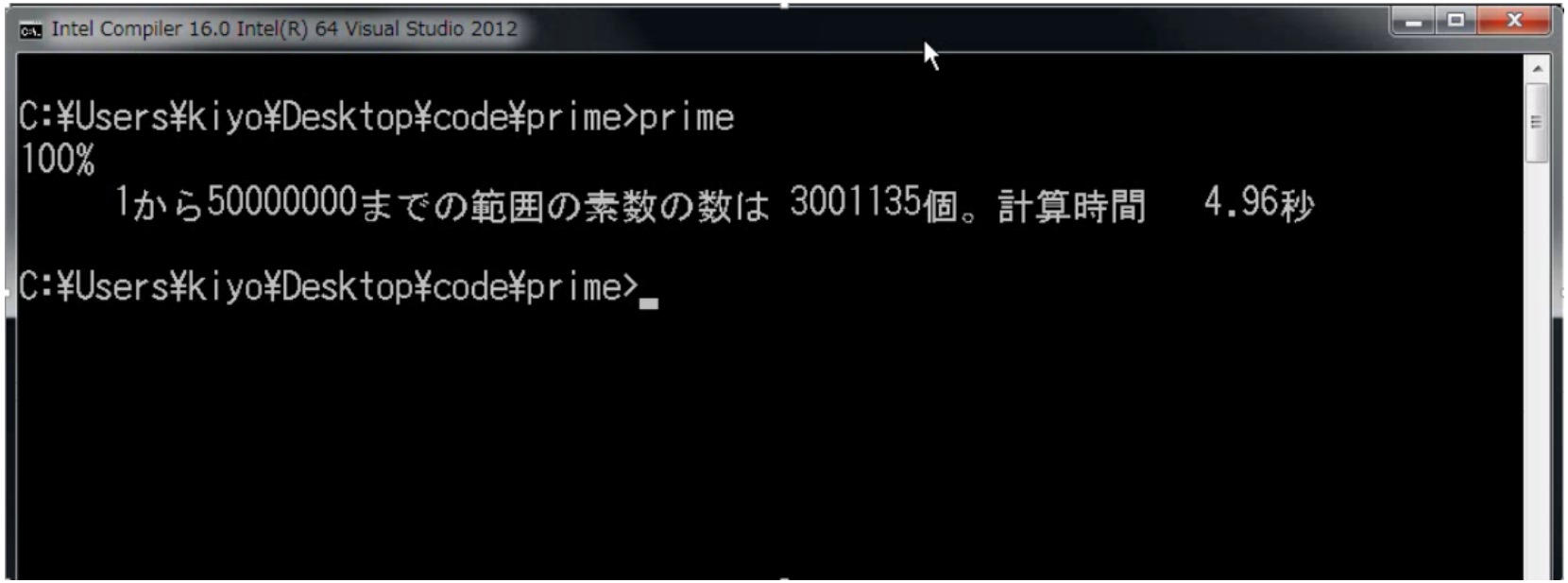
```
void FindPrimes(int start, int end){
    // 開始値は常に奇数
        int range = end - start + 1;
#pragma omp parallel for reduction(+:gPrimesFound)
    for( int i = start; i <= end; i+=2 ){
        if( TestForPrime(i) )
                gPrimesFound++;
         InterlockedIncrement((LPLONG) &gProgress);
    }
}
```

```
int main(){
Int start = 1, end = PRIME_END, range;
HANDLE hThread;
clock_t before, after;

        gPrimesFound = 1;  // 2 は特殊ケースなので1つカウント

        range = end - start + 1;
        hThread= (HANDLE) _beginthreadex(NULL, 0, ShowProgressThread,
&range, 0, NULL);

        before = clock();
        FindPrimes(start, end);
        after = clock();

        WaitForSingleObject(hThread, INFINITE);

        printf("¥n%6dから%6dまでの範囲の素数の数は%8d個。計算時間%7.2f秒¥n",
            start, end, gPrimesFound,(float)(after - before)/
CLOCKS_PER_SEC);
}
```

このページは自動送りされません。
画面をクリックして進んでください

©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。

アプリケーションの実行



このアプリケーションは、1から 50000000 (5 千万) までの範囲にある素数を数え表示します。
その際、計算中の進行状況を % で示します。インテル® Core™ i7-5820K プロセッサー 3.3GHz (6 コア、12 スレッド) 環境で実行すると、およそ 4.96 秒かかりました

©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。

Windows* 向けのソースのオフロード例

```
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <windows.h>
#include <process.h>

#define PRIME_END 50000000
static int gProgress = 0, gPrimesFound = 0;

unsigned WINAPI ShowProgressThread(LPVOID val) {
        int *ip = (int *) val;
        int percentDone = 0, range = *ip;
        while( percentDone < 100 ) {
                percentDone =
(int)((float)gProgress/(float)range *200.0f + 0.5f);
                if( percentDone % 10 == 0 ){
                printf("¥b¥b¥b¥b%3d%%", percentDone);
                Sleep(1);
                }
        }
        return 0;
}
bool TestForPrime(int val){
    int limit, factor = 3;

    limit = (long)(sqrtf((float)val)+0.5f);
```

これらの Windows* 固有のソースは、マクロを使用してクロスコンパイル時に未定義にする必要があります

```
void FindPrimes(int start, int end){
    // 開始値は常に奇数
        int range = end - start + 1;
#pragma omp parallel for reduction(+:gPrimesFound)
    for( int i = start; i <= end; i+=2 ){
        if( TestForPrime(i) )
                gPrimesFound++;
         InterlockedIncrement((LPLONG) &gProgress);
    }
}

int main(){
Int start = 1, end = PRIME_END, range;
HANDLE hThread;
clock_t before, after;

        gPrimesFound = 1;  // 2 は特殊ケースなので1つカウント

        range = end - start + 1;
        hThread= (HANDLE) _beginthreadex(NULL, 0, ShowProgressThread,
&range, 0, NULL);

        before = clock();
        FindPrimes(start, end);
        after = clock();

        WaitForSingleObject(hThread, INFINITE);

        printf("¥n%6dから%6dまでの範囲の素数の数は%8d個。計算時間%7.2f秒¥n",
            start, end, gPrimesFound,(float)(after - before)/
CLOCKS_PER_SEC);
}
```

このページは自動送りされません。
画面をクリックして進んでください

©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。

オフロード領域の宣言

OpenMP* 並列領域をオフロードし、Win32* API を等価の処理に置き換えます

```
#pragma offload target(mic) in(start, end) inout(gPrimesFound, gProgress)
#pragma omp parallel for reduction(+:gPrimesFound)
    for( int i = start; i <= end; i+=2 ){
        if( TestForPrime(i) )
            gPrimesFound++;
#pragma omp atomic
       gProgress++; // InterlockedIncrement((LPLONG) &gProgress);
    }
```

Intel Compiler 16.0 Intel(R) 64 Visual Studio 2012

```
C:¥Users¥kiyo¥Desktop¥code¥prime>icl prime_offload.cpp /Qopenmp /GR-
インテル(R) 64 対応インテル(R) C++ コンパイラー (インテル(R) 64 対応アプリケーシ
ョン用)、バージョン 16.0.0.079 ビルド 20150623
(C) 1985-2015 Intel Corporation.  無断での引用、転載を禁じます。

prime_offload.cpp
prime_offload.cpp(59): エラー: オフロード領域で呼び出される 関数 "TestForPrime"
には互換性のある "target" 属性がなければなりません。
          if( TestForPrime(i) )
              ^

コンパイルは prime_offload.cpp で異常終了しました (コード 2)。
```

関数呼び出しがある!!

©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。

オフロード領域の宣言 (2)

オフロード領域内で呼び出すほかの関数は、属性宣言が必要

```
#pragma offload_attribute(push, target(mic))
bool TestForPrime(int val){
    int limit, factor = 3;

    limit = (long)(sqrtf((float)val)+0.5f);
    while( (factor <= limit) && (val % factor))
        factor ++;

    return (factor > limit);
}
#pragma offload_attribute(pop)
…
#pragma offload target(mic) in(start, end) inout(gPrimesFound, gProgress)
#pragma omp parallel for reduction(+:gPrimesFound) // reduction(+:gProgress)
    for( int i = start; i <= end; i+=2 ){
        if( TestForPrime(i) )
…
```

©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。

実行可能なオフロードコード

```
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <offload.h>

#ifndef __MIC__
#include <windows.h>
#include <process.h>
#endif

#define PRIME_END 10000000

int gProgress    = 0;
int gPrimesFound = 0;

#ifndef __MIC__
unsigned WINAPI ShowProgressThread(LPVOID val)
{
   int *ip = (int *) val;
   int percentDone = 0, range = *ip;

   while( percentDone < 100 {
   percentDone = (int)((float)gProgress/(float)range *200.0f + 0.5f);
          if( percentDone % 10 == 0 ){
                    printf("¥b¥b¥b%3d%%", percentDone);
                    Sleep(1);
          }
   }
   return 0;
}
#endif
#pragma offload_attribute(push, target(mic))
bool TestForPrime(int val){
    int limit, factor = 3;

    limit = (long)(sqrtf((float)val)+0.5f);
    while( (factor <= limit) && (val % factor))
        factor ++;

    return (factor > limit);
}
#pragma offload_attribute(pop)
```

```
void FindPrimes(int start, int end){
          int range = end - start + 1;

#pragma offload target(mic) in(start, end) inout(gPrimesFound)
#pragma omp parallel for reduction(+:gPrimesFound)
    for( int i = start; i <= end; i+=2 ){
        if( TestForPrime(i) )
                    gPrimesFound++;
#pragma omp atomic
          gProgress++;
    }
}

int main(){
    int     start = 1, end = PRIME_END, range;
#ifndef __MIC__
    HANDLE hThread;
#endif
    clock_t before, after;

    gPrimesFound = 1;  // 2 は特殊ケースなので1つカウント
    range = end - start + 1;
#ifndef __MIC__
          hThread= (HANDLE) _beginthreadex(NULL, 0, ShowProgressThread, &range, 0, NULL);
#endif
    before = clock();
    FindPrimes(start, end);
    after = clock();

#ifndef __MIC__
          WaitForSingleObject(hThread, INFINITE);
#endif

          printf("¥n%6dから%6dまでの範囲の素数の数は%8d個。計算時間%7.2f秒¥n",
           start, end, gPrimesFound,(float)(after - before)/ CLOCKS_PER_SEC);
}
```

このページは自動送りされません。
画面をクリックして進んでください

©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。

変更後のソースをコンパイルして実行

> icl prime_offload3.cpp /Qopenmp

処理時間は、2.56 秒に短縮されましたが、進行状況がリアルタイムに更新されていません

©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。

問題は？

ホスト

```
int gPrimesFound, gProgress;
main(){
// 進行状況を表示する Win32* スレッドを生成
  FindPrimes();  // 素数をカウント
}
```

```
ShowProgressThread(){
 gProgress を参照し進行状況を表示
}
```

オフロード

インテル® Xeon Phi™ コプロセッサー

```
TestForPrime() { …}
FindPrimes(){
  TestForPrime();
     gPrimesFound++;  // 素数の個数
     gProgress++;        // 進行カウント
}
```

gProgress 変数はオフロードされた処理が終了した時にホストで更新される

©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。

解決策

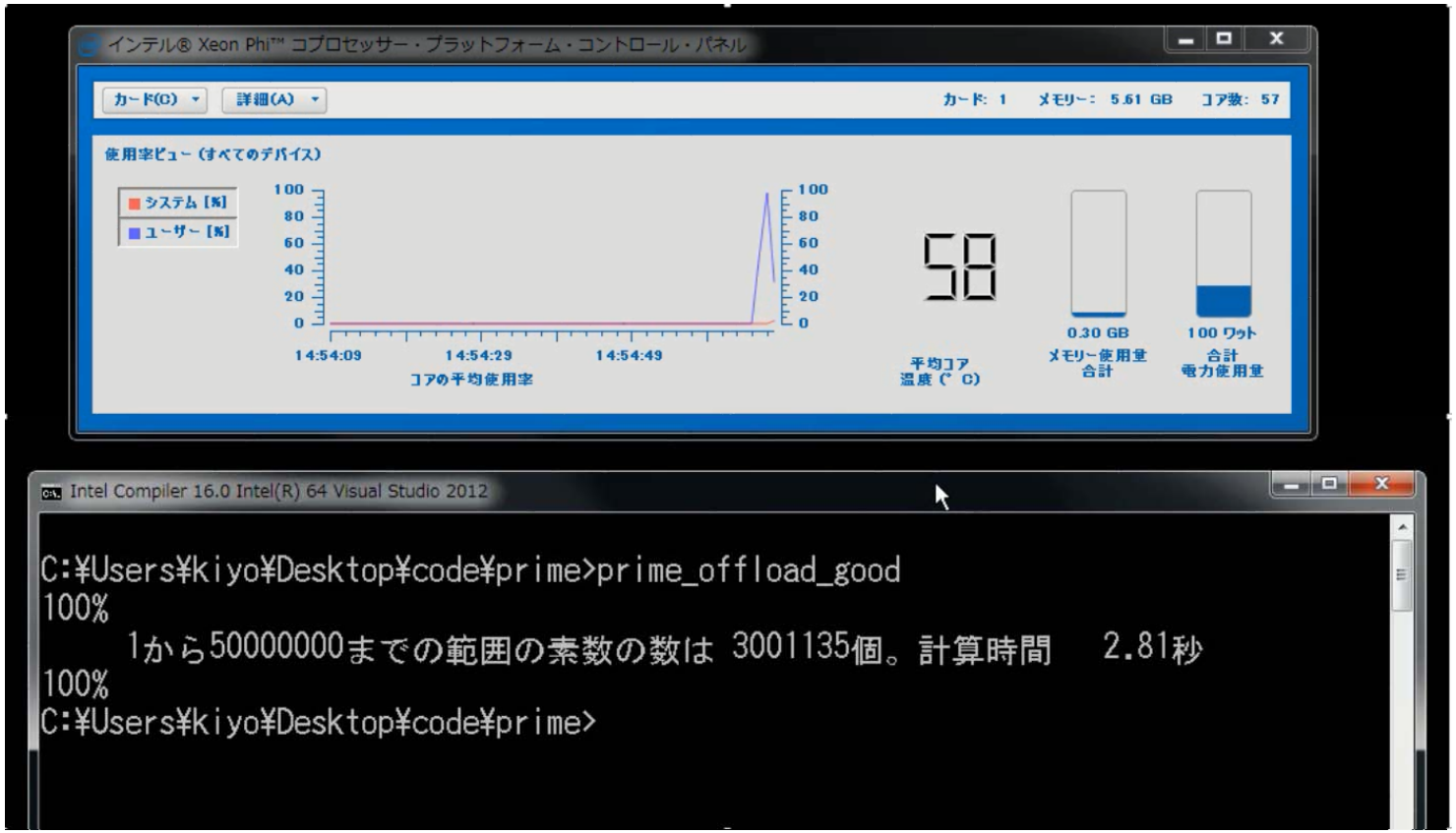
```
#pragma offload target(mic) optional in (start, end) inout(gPrimesFound, gProgress)
#pragma omp parallel for reduction(+:gPrimesFound) schedule(static,4)
    for( int i = start; i <= end; i+=2 ){
        if( TestForPrime(i) )
           gPrimesFound++;

        ShowProgress(i, range);
    }
```

```
#pragma offload_attribute(push, target(mic))
void ShowProgress( int val, int range ){
    int percentDone = 0;
#pragma omp atomic
    gProgress++;
#pragma omp master
  {
    percentDone = (int)((float)gProgress/(float)range *200.0f + 0.5f);
    if( percentDone % 10 == 0 ){
        printf("¥b¥b¥b¥b%3d%%", percentDone);
    }
  }
}
#pragma offload_attribute(pop)
```

進行状況表示処理も
コプロセッサーへオフロードする

©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。

解決策



4.96 秒

↓

2.81 秒
に高速化!!

©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。

内容

- Windows* 環境におけるプログラミングの注意点
- Windows* アプリケーションのオフロードの例
- デバイスの有無とオフロード
- シンメトリック同時処理

©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。

コプロセッサーの有無による振る舞い

オフロード向けインテル® コンパイラー拡張と OpenMP* 4.0 の実装では、プログラムがデバイスを利用できない場合の振る舞いが若干異なります；

```
#pragma offload target(mic) in(num_steps, step) inout(sum)
          もしくは
#pragma omp target map(alloc: x) map(to: num_steps, step) map(tofrom: sum) {
#pragma omp parallel for simd reduction(+:sum) private(x)
     for (i=0;i< num_steps; i++){
     x = (i+0.5)*step;
     sum = sum + 4.0/(1.0+x*x);
     }
}
```

デバイスを利用できない場合

Intel Compiler 16.0 Intel(R) 64 Visual Studio 2012

```
C:¥Users¥kiyo¥Desktop¥code¥pi>pi_intel_offload
offload error: cannot offload to MIC - device is not available
```

©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 – 解決策

```
#pragma omp target map(alloc: x) map(to: num_steps, step) map(tofrom: sum)
if(omp_get_num_devices())
{
#pragma omp parallel for simd reduction(+:sum) private(x)
     for (i=0;i< num_steps; i++){
     x = (i+0.5)*step;
     sum = sum + 4.0/(1.0+x*x);
     }
}
```

OpenMP* API omp_get_num_devices() は利用可能なデバイス数を返します。つまり戻り値が 0 (ゼロ) であれば利用可能なデバイスがないため、omp target 構文の if 節はオフロードを行わないことを指示します

Intel Compiler 16.0 Intel(R) 64 Visual Studio 2012

```
C:¥Users¥kiyo¥Desktop¥code¥pi>pi_test
Pi = 3.141593 Time = 0.469000 Available threads = 8
```

©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。

オフロード向けインテル® コンパイラー拡張 – 解決策

```
#pragma offload target(mic) optional in(num_steps, step) inout(sum)
#pragma omp parallel for reduction(+:sum)
     for (i=0;i< num_steps; i++){
        double x = (i+0.5)*step;
        sum += 4.0/(1.0+x*x);
     }
```

ターゲットが利用できない場合、optional、mandatory、および status 節により文の実行方法が決定されます

指定する節	動作
optional	文はホストで実行されます
optional status (status変数)	文はホストで実行され、status 変数にターゲットを利用できない理由が格納されます
mandatory	文は無視され、プログラムは終了します (デフォルト)
mandatory status (status変数)	文は無視され、プログラムは実行を継続します。status 変数にターゲットを利用できない理由が格納されます

©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。

オフロード向けインテル® コンパイラー拡張 – エラー処理

```
#include <offload.h>
_Offload_status var;

main(){
OFFLOAD_STATUS_INIT(var); // var.result のステータス値
…
1. #pragma offload target(mic) mandatory in(num_steps, step) inout(sum)
2. #pragma offload target(mic) mandatory status(var) in(num_steps, step) inout(sum)
3. #pragma offload target(mic) optional in(num_steps, step) inout(sum)
…
}
```

var.result の値	説明
OFFLOAD_SUCCESS = 0	文はターゲットで実行されました
OFFLOAD_DISABLED	文はターゲットで実行されませんでした。*if-clause* が指定され値が false の場合文はホストで実行されました
OFFLOAD_UNAVAILABLE	ターゲットが利用できないため、文はターゲットで実行されませんでした
OFFLOAD_OUT_OF_MEMORY	オフロード用のメモリーが不足していたため、文はターゲットで実行されませんでした
OFFLOAD_PROCESS_DIED	ターゲットでランタイムエラーが発生し、ターゲットのプロセスが終了したため、文はターゲットで実行されませんでした
OFFLOAD_ERROR	エラーが発生したため、文はターゲットで実行されませんでした

©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。

内容

- Windows* 環境におけるプログラミングの注意点
- Windows* アプリケーションのオフロードの例
- デバイスの有無とオフロード
- シンメトリック同時処理

©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。

デフォルトのオフロードではホストは待機する

オフロード中に待機しているホスト・プロセッサーも活用

```
void FindPrimes(int start, int end){
char sig = 0;

#pragma offload target(mic:0) in (start, end) inout(gPrimesFound) signal(&sig)
#pragma omp parallel for reduction(+:gPrimesFound)
    for( int i = ?0000001; i <= end; i+=2 ){
        if( TestForPrime(i) )
            gPrimesFound++;
    }

#ifndef __MIC__
#pragma omp parallel for reduction(+:gPrimesFound2)
    for( int i = start; i <= ?0000000; i+=2 ){
        if( TestForPrime(i) )
            gPrimesFound2++;
    }
#pragma offload_wait target(mic:0) wait(&sig)
   gPrimesFound += gPrimesFound2;
#endif
}
```

コプロセッサーはオフロードを完了したらシグナル sig を発生

ホストはオフロード後待機せず、数値範囲の一部分を処理。処理後、offload_wait 句でコプロセッサーのシグナル sig を待機

©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。

ホストとコプロセッサーで同時実行

©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・ゲインは？

プログラミング・モデル	時間
シリアル	17.13 秒
ホスト上で OpenMP*	3.45 秒
コプロセッサー上で OpenMP*	1.58 秒
ホストとコプロセッサーで OpenMP*	1.26 秒 (1-20000000 vs 20000001 – 50000000)
ホストとコプロセッサーで OpenMP*	2.14 秒 (1-30000000 vs 30000001 – 50000000)
ホストとコプロセッサーで OpenMP*	1.34 秒 (1-10000000 vs 10000001 – 50000000)

実行環境：　インテル® Core™ i7-5820K プロセッサー 3.3GHz (6 コア、12 スレッド)、メモリー 32GB、
Windows 7 SP1 64ビット 、インテル® Xeon Phi™ コプロセッサー 3120A、インテル® MPSS 3.5.2

©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。

まとめ

- Windows* 環境におけるプログラミングの注意点
- Windows* アプリケーションのオフロードの例
- デバイスの有無とオフロード
- シンメトリック同時処理

©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®

experience
what's inside™

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