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

パフォーマンスを検証する (約 23 分)

2015 年 10 月

内容

- このセッションの目標
- ハードウェア仕様の理解
- 1 コアの性能を引き出す
- 全てのコアにスケールする
- オフロードモデル
- メモリー帯域幅

©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。

このセッションンの目標

- インテル® Xeon Phi™ コプロセッサーのハードウェア仕様とアプリケーションのパフォーマンスの関係を理解する
  - ✓ 1 コアの性能を引き出す
  - ✓ ベクトル化の効果
  - ✓ 全てのコアにスケールする
  - ✓ メモリー帯域幅とパフォーマンスの関係

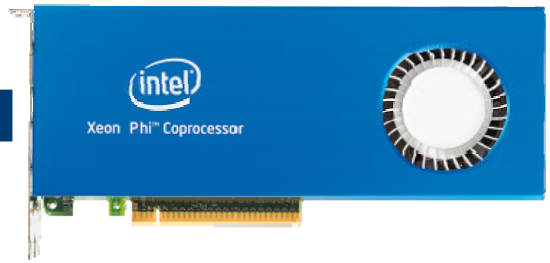
このセッションで使用するサンプルコードは、http://lotsofcores.com/downloads の最後にある「Example Code From Book Chapters 2 – 4」からダウンロードできます

©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。

テストに利用したインテル® Xeon Phi™ コプロセッサーの仕様 (3120A)

クロック周波数	1.100 GHz
コア数	57
メモリーサイズ/タイプ	6 GB / GDDR5
メモリースピード	5.0 GT/秒
ピーク DP/SP FLOPs	1.003/2.006 テラFLOPS
ピークメモリー帯域	240 GB/秒

注: 利用可能な複数の製品モデルの仕様は異なります



インテル® Xeon Phi™ コプロセッサーの単精度浮動小数点演算のピーク性能は、次の式で求める:

クロック周波数 x コア数 x 16 レーン x 2 (FMA) FLOPS/サイクル

ここで利用するコプロセッサー・モデルの FLOPS 値は以下になる:

1.100 GHz x 57 コア x 16 レーン x 2 = 2006.4 ギガ FLOPS

©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。

サンプルコード (helloflops1.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>

double dtime() {

    double tseconds = 0.0;
    struct timeval mytime;
    gettimeofday(&mytime,(struct timezone*)0);
    tseconds = (double)(mytime.tv_sec + mytime.tv_usec*1.0e-6);
    return( tseconds );
}

#define FLOPS_ARRAY_SIZE (1024*1024)
#define MAXFLOPS_ITERS 100000000
#define LOOP_COUNT 128

// Floating point ops per inner loop iteration

#define FLOPSPERCALC 2

// define some arrays - 64 byte aligned for fastest cache access

float fa[FLOPS_ARRAY_SIZE] __attribute__((align(64)));
float fb[FLOPS_ARRAY_SIZE] __attribute__((align(64)));
```

```c
int main(int argc, char *argv[] )  {
        int i,j,k;
        double tstart, tstop, ttime;
        double gflops = 0.0;
        float a=1.1;

        printf("Initializing¥r¥n");
        for(i=0; i<FLOPS_ARRAY_SIZE; i++)
        {
            fa[i] = (float)i + 0.1;
            fb[i] = (float)i + 0.2;
        }

        printf("Starting Compute¥r¥n");

        tstart = dtime();
        for(j=0; j<MAXFLOPS_ITERS; j++)  {
        for(k=0; k<LOOP_COUNT; k++)  {
                fa[k] = a * fa[k] + fb[k];
            }
         }
         tstop = dtime();

         // # of gigaflops we just calculated
         gflops = (double)(1.0e-9*LOOP_COUNT*
                      MAXFLOPS_ITERS*FLOPSPERCALC);

         // elasped time
         ttime = tstop - tstart;

         if ((ttime) > 0.0) {
             printf("GFlops = %10.3lf, Secs = %10.3lf,
               GFlops per sec = %10.3lf¥r¥n",
                      gflops, ttime, gflops/ttime);
         }
         return( 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。

最初のコード

サンプルコード helloflops.c を見てみましょう。初期化とセットアップの後、一般に SAXPY と呼ばれる演算を行う単一行がループの内側にあり、以下のように構成されます;

```
fa[k] = a *fa[k] + fb[k];
```

このコードは、インテル® Xeon Phi™ コプロセッサーの主要な演算能力である、単一命令で乗算と加算を融合する (FMA) を利用できる

このコードから得られる最大 FLOPS 値は？
1.100 GHz x 57 コア x 16 レーン x 2 =
2006.4 / 57 = 35.2 ギガ FLOPS

©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。

最初のコードをコンパイルして実行 (ネイティブモデル)

$ icc -mmic -qopt-report=3 –qopt-report-phase=vec -O3 helloflops1.c -o helloflops1 (Linux*)

> icl /Qmic helloflops1.c -qopt-report=3 –qopt-report-phase=vec –O3 –o helloflops1 (Windows*)

-mmic (/Qmic) オプションは、C コンパイラーにインテル® メニー・インテグレーテッド・コア・アーキテクチャー (インテル® MIC) のコプロセッサー向けのコードを生成することを指示。-qopt-report=3 –qopt-report-phase=vec オプションはベクトル化レポートを生成し、-O3 は最適化レベルを指定

```
Intel Parallel Studio XE 2015 Composer Edition Intel(R) 64 Visual Studio 2012

C:¥Users¥kiyo¥Desktop¥Book¥Ch2>micrun helloflops1
Initializing
Starting Compute
GFlops =     25.600, Secs =      1.461, GFlops per sec =     17.520
```

micrun コマンドは、micnativeloadex コマンドへのエリアス名です

”かなり良い” 結果ですが、1 コアのピーク性能のおよそ半分の 49.7% … 問題は？

©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。

最初のコードの問題を考える

コプロセッサーは、高度な並列性とスケーラブルなアプリケーションを対象としているため、命令のスケジュール・メカニズムは複数のスレッドが重い問題に適用されることを想定する

特殊なケースとして、1 つのコアで 1 つのスレッドのみが起動された時、スケジューラーは利用可能なスレッドを見つけられるまで、機能的には何もしない特殊な “NULL スレッド” に切り替えてクロックをパスする。基本的に、1 コアで 1 スレッドがアクティブである場合、他のすべてのサイクルはスキップされるため、期待される性能は半分になります

最初のコードは、シングルスレッドの理論上のピーク性能に非常に近いと考えることがでる

©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。

次に進む前に重要なことを確認

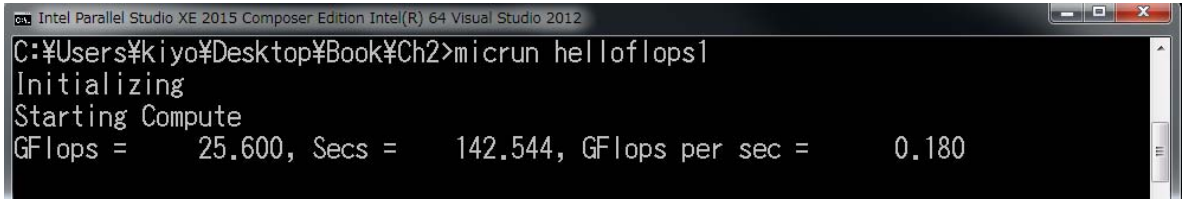
> icl /Qmic helloflops1.c -qopt-report=3 –qopt-report-phase=vec –O3 –o helloflops1

helloflops1.c (54): (col. 9) remark: ループがベクトル化されました。

helloflops1.c (70): (col. 13) remark: ループがベクトル化されました。

helloflops1.c (64): (col. 9) remark: ループはベクトル化されませんでした: 内部ループがすでにベクトル化されています。

> icl /Qmic helloflops1.c –no-vec -qopt-report=3 –qopt-report-phase=vec –O3 –o helloflops1



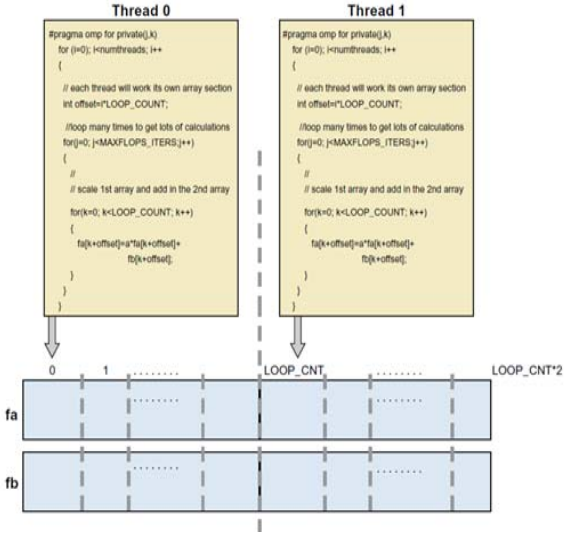
micrun コマンドは、micnativeloadex コマンドへのエリアス名です

シングルコアで 35 ギガ FLOPS を超えるパフォーマンスを達成するには、FMA 演算による高度にベクトル化されたコードが必要

©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。

2 番目のサンプルコード: 2 つのスレッドで実行

ここでの目標は、コアのピーク性能に近づくため 1 つのコアで少なくてももう 1 つのスレッドを有効にすること



OpenMP* を使用して配列 fa[] と fc[] を
2 つのスレッドに分割する

- スレッド化を有効にするため簡単で強力な OpenMP* の parallel for 指示句を利用
- 2 つの OpenMP* API を次のように呼び出してスレッド数とアフィニティーを設定

```
omp_set_num_threads(2);
kmp_set_defaults("KMP_AFFINITY=compact");
```

©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。

2 番目のサンプルコード (helloflops2.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>

double dtime() {

    double tseconds = 0.0;
    struct timeval mytime;
    gettimeofday(&mytime,(struct timezone*)0);
    tseconds = (double)(mytime.tv_sec + mytime.tv_usec*1.0e-6);
    return( tseconds );
}

#define FLOPS_ARRAY_SIZE (1024*1024)
#define MAXFLOPS_ITERS 100000000
#define LOOP_COUNT 128

// Floating point ops per inner loop iteration

#define FLOPSPERCALC 2

// define some arrays - 64 byte aligned for fastest cache access

float fa[FLOPS_ARRAY_SIZE] __attribute__((align(64)));
float fb[FLOPS_ARRAY_SIZE] __attribute__((align(64)));
```

```c
int main(int argc, char *argv[] )  {
        int i,j,k;
        double tstart, tstop, ttime;
            double gflops = 0.0;
            float a=1.1;

omp_set_num_threads(2);
kmp_set_defaults("KMP_AFFINITY=compact");
#pragma omp parallel
#pragma omp master
    numthreads = omp_get_num_threads();
     printf("Initializing¥r¥n");
#pragma omp parallel for
        for(i=0; i<FLOPS_ARRAY_SIZE; i++){
             fa[i] = (float)i + 0.1;
                fb[i] = (float)i + 0.2;
        }

        printf("Starting Compute¥r¥n");

        tstart = dtime();
#pragma omp parallel for private(j,k)
     for (i=0; i<numthreads; i++)
            int offset = i*LOOP_COUNT;
        for(j=0; j<MAXFLOPS_ITERS; j++)  {
         for(k=0; k<LOOP_COUNT; k++)  {
                fa[k+offset]=a*fa[k+offset]+ fb[k+offset];
            }
         }
         tstop = dtime();

        // # of gigaflops we just calculated
        gflops = (double)(1.0e-9*LOOP_COUNT*
                                MAXFLOPS_ITERS*FLOPSPERCALC);

        // elasped time
        ttime = tstop - tstart;

        if ((ttime) > 0.0) {
            printf("GFlops = %10.3lf, Secs = %10.3lf,
                      GFlops per sec = %10.3lf¥r¥n",
                      gflops, ttime, gflops/ttime);
        }
        return( 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。

2 番目のコードをコンパイルして実行 (ネイティブモデル)

> icl /Qmic helloflops2.c -qopt-report=3 –qopt-report-phase=vec –O3 –openmp –o helloflops2

ここでは、まだ内側のループがベクトル化できたかどうかレポートを得ることを忘れないでください。もしベクトル化されていない場合、前述の -no-vec オプションを取り忘れていないか確認してください

```
Intel Parallel Studio XE 2015 Composer Edition Intel(R) 64 Visual Studio 2012
C:¥Users¥kiyo¥Desktop¥Book¥Ch2>micrun helloflops2
Initializing
Starting Compute on 2 threads
GFlops =     51.200, Secs =      1.462, GFlops per sec =     35.030
```

micrun コマンドは、micnativeloadex コマンドへのエリアス名です

2 つのスレッドで実行すると、計算は各サイクルで行われ、シングルコア効率の 99.5% を達成

©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 番目のサンプルコード: より多くのスレッドで実行

サンプルコード 2 は、1 コアで 2 つのスレッドを動作して 1 コアの最大 FLOPS 値を求めようとしているため、この設定を削除;

```
omp_set_num_threads(2);
kmp_set_defaults("KMP_AFFINITY=compact");
```

代わりに、環境変数 OMP_NUM_THREADS と KMP_AFFINITY でスレッド数とアフィニティーを調整し、いろいろな設定で動作を確認

©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 番目のサンプルコード (helloflops3.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>

double dtime() {

    double tseconds = 0.0;
    struct timeval mytime;
    gettimeofday(&mytime,(struct timezone*)0);
    tseconds = (double)(mytime.tv_sec + mytime.tv_usec*1.0e-6);
    return( tseconds );
}

#define FLOPS_ARRAY_SIZE (1024*1024)
#define MAXFLOPS_ITERS 100000000
#define LOOP_COUNT 128

// Floating point ops per inner loop iteration

#define FLOPSPERCALC 2

// define some arrays - 64 byte aligned for fastest cache access

float fa[FLOPS_ARRAY_SIZE] __attribute__((align(64)));
float fb[FLOPS_ARRAY_SIZE] __attribute__((align(64)));
```

```c
int main(int argc, char *argv[] )  {
        int i,j,k;
        double tstart, tstop, ttime;
           double gflops = 0.0;
           float a=1.1;

#pragma omp parallel
#pragma omp master
    numthreads = omp_get_num_threads();
         printf("Initializing¥r¥n");
#pragma omp parallel for
         for(i=0; i<FLOPS_ARRAY_SIZE; i++){
              fa[i] = (float)i + 0.1;
                 fb[i] = (float)i + 0.2;
         }

         printf("Starting Compute¥r¥n");

         tstart = dtime();
#pragma omp parallel for private(j,k)
     for (i=0; i<numthreads; i++)
             int offset = i*LOOP_COUNT;
         for(j=0; j<MAXFLOPS_ITERS; j++)  {
          for(k=0; k<LOOP_COUNT; k++)  {
                 fa[k+offset]=a*fa[k+offset]+ fb[k+offset];
             }
          }
          tstop = dtime();

        // # of gigaflops we just calculated
        gflops = (double)(1.0e-9*LOOP_COUNT*
                                MAXFLOPS_ITERS*FLOPSPERCALC);

        // elasped time
        ttime = tstop - tstart;

        if ((ttime) > 0.0) {
            printf("GFlops = %10.3lf, Secs = %10.3lf,
                      GFlops per sec = %10.3lf¥r¥n",
                      gflops, ttime, gflops/ttime);
        }
        return( 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。

3 番目のコードをコンパイルして実行 (ネイティブモデル)

> icl /Qmic helloflops3.c –openmp –qopt-report=3 –qopt-report-phase=vec -O3 -o helloflops3

helloflops3 を実行する前に、コプロセッサー・コマンド・プロンプトで次の設定を行う:

> set OMP_NUM_THREADS=2
> set KMP_AFFINITY=compact

192.168.1.100 - PuTTY

```
[root@mic0 ~]# ./helloflops3_xphi
Initializing
Starting Compute on 2 threads
GFlops =     51.200, Secs =      1.463, GFlops per sec =     34.985
[root@mic0 ~]#
```

©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 番目のコードを最大スレッドで実行 (ネイティブモデル)

環境変数を以下に変更:

% export OMP_NUM_THREADS=114

% export KMP_AFFINITY=scatter

この設定では、114 スレッド (2 x 57 コア) を要求し、スレッドがコア間で “散乱 (scatter)” するように指定する。そのため、OpenMP* ランタイムは最初に割り当てたコアから離れたコアに後続のスレッドを割り当てることを繰り返し、コア数より大きなスレッドは最初のコアから再び割り当てを行なう

192.168.1.100 - PuTTY

```
[root@mic0 ~]# ./helloflops3_xphi
Initializing
Starting Compute on 114 threads
GFlops =   2918.400, Secs =      1.631, GFlops per sec =   1789.004
[root@mic0 ~]#
```

テスト・プラットフォーム (57 コア) で 1.7 テラ FLOPS を越えた！
これは、89.1% 以上の効率を達成

©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。

4 番目のサンプルコード: オフロードモデルで実行

オフロードコードは、ホスト・プロセッサー用にコンパイルされ、必要に応じてオフロード宣言子が検出されてコプロセッサーが稼働して利用可能な時に、指示されたデータとコードが自動的にホストとコプロセッサー間で転送される

インテル® Xeon Phi™ コプロセッサー上で実行するコードの前に、2つの #pragma offload_target(mic) 宣言子を追加

©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。

4 番目のサンプルコード (helloflops3offload.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>

double dtime() {

    double tseconds = 0.0;
    struct timeval mytime;
    gettimeofday(&mytime,(struct timezone*)0);
    tseconds = (double)(mytime.tv_sec + mytime.tv_usec*1.0e-6);
    return( tseconds );
}

#define FLOPS_ARRAY_SIZE (1024*1024)
#define MAXFLOPS_ITERS 100000000
#define LOOP_COUNT 128

// Floating point ops per inner loop iteration

#define FLOPSPERCALC 2

// define some arrays - 64 byte aligned for fastest cache access

float fa[FLOPS_ARRAY_SIZE] __attribute__((align(64)));
float fb[FLOPS_ARRAY_SIZE] __attribute__((align(64)));
```

```c
int main(int argc, char *argv[] )  {
        int i,j,k;
        double tstart, tstop, ttime;
           double gflops = 0.0;
           float a=1.1;

#pragma offload target (mic)
#pragma omp parallel
#pragma omp master
    numthreads = omp_get_num_threads();
         printf("Initializing¥r¥n");
#pragma omp parallel for
         for(i=0; i<FLOPS_ARRAY_SIZE; i++){
              fa[i] = (float)i + 0.1;
                 fb[i] = (float)i + 0.2;
         }

         printf("Starting Compute¥r¥n");

         tstart = dtime();
#pragma offload target (mic)
#pragma omp parallel for private(j,k)
     for (i=0; i<numthreads; i++)
            int offset = i*LOOP_COUNT;
         for(j=0; j<MAXFLOPS_ITERS; j++)  {
          for(k=0; k<LOOP_COUNT; k++)  {
                 fa[k+offset]=a*fa[k+offset]+ fb[k+offset];
             }
          }
          tstop = dtime();

        // # of gigaflops we just calculated
        gflops = (double)(1.0e-9*LOOP_COUNT*
                                MAXFLOPS_ITERS*FLOPSPERCALC);

        // elasped time
        ttime = tstop - tstart;

        if ((ttime) > 0.0) {
            printf("GFlops = %10.3lf, Secs = %10.3lf,
                      GFlops per sec = %10.3lf¥r¥n",
                      gflops, ttime, gflops/ttime);
        }
        return( 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。

4 番目のコードをコンパイル (オフロードモデル)

> icl /Qopenmp /Qopt-report=3 /Qopt-report-phase=vec /O3 helloflops3offload.c /Fehelloflops3offload

/Qmic オプションを指定していないことに気づきましたか？
このコードはホスト・プロセッサーをターゲットとしており、#pragma offload 宣言子は、コプロセッサー向けのコードを生成するべき特定の場所を示す。ベクトル化レポートのオプションが指定されているが、コプロセッサーをターゲットとしたコード領域のレポートには、*MIC* 文字列が追加される

```
Intel Parallel Studio XE 2015 Composer Edition Intel(R) 64 Visual Studio 2012
*MIC* 最適化レポート開始: main(int, char **)
   remark #15542: ループ はベクトル化されませんでした: 内部ループがすでにベクトル化されています。
   remark #15542: ループ はベクトル化されませんでした: 内部ループがすでにベクトル化されています。
   remark #15300: ループがベクトル化されました。
```

©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。

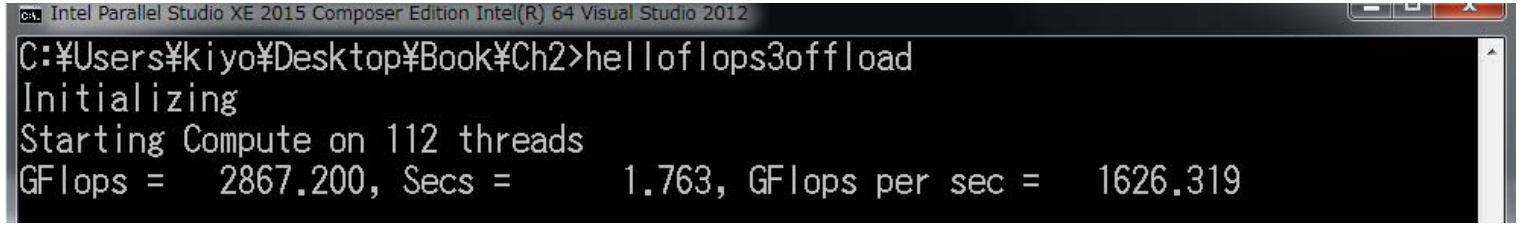
4 番目のコードを実行 (オフロードモデル)

プログラムの実行が期待通りになることを確実にするには、ホスト上でいくつかの環境変数の設定が必要。ホスト側のコマンド・プロンプトで次のように入力:

> set MIC_ENV_PREFIX=MIC

> set MIC_OMP_NUM_THREADS=112

> set MIC_KMP_AFFINITY=scatter



©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。

5 番目のサンプルコード: メモリー帯域の影響を考える

ここで利用するモデルの仕様:

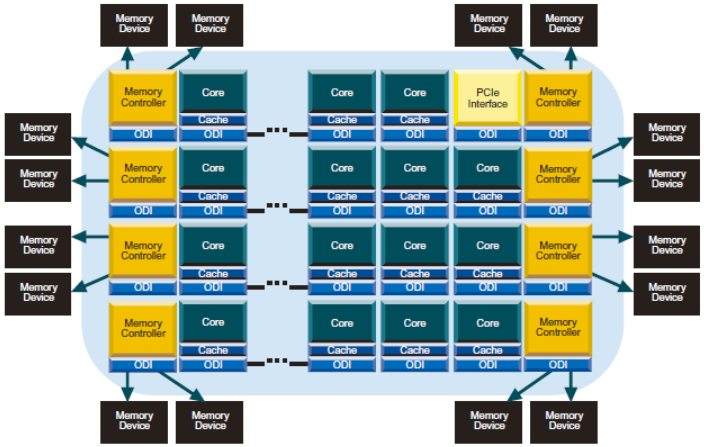
- 6 ギガバイトの容量と毎秒 5.0 ギガの (GT/s) 転送速度のメモリー・チャンネル・インターフェイスを持つ
- それぞれ 2 つのメモリーチャンネルにアクセスする 6 つのメモリー・コントローラーがある
- GDDR5 メモリーへの各メモリー・トランザクションは、4 バイト・データでチャンネルあたり 5.0GT/s (34 バイト) または 20GB/s 。合計 12 チャンネルで 240GB/s の最大転送速度を提供

メモリー・サブシステムのパフォーマンスは、ハードウェアの実装、”配線” 上の電磁ノイズ、およびその他の複雑なシステムの要因に影響を受る。そのため、理想的な状況であっても、最大転送速度の 50 から 60% 程度

©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。

サンプルコードについて

hellomem.c コードは、メモリー帯域幅を測定して最大化するために使用。このコードは、メインの内ループで単純なリードとライトのメモリー・アクセス・パターンを実行し、大きな 1 つの配列を数回に分けてコピーする



©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。

5 番目のサンプルコード (hellomem.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <omp.h>

double dtime() {

    double tseconds = 0.0;
    struct timeval mytime;
    gettimeofday(&mytime,(struct timezone*)0);
    tseconds = (double)(mytime.tv_sec + mytime.tv_usec*1.0e-
6);
    return( tseconds );
}

#define FLOPS_ARRAY_SIZE (1024*1024)
#define MAXFLOPS_ITERS 100000000
#define LOOP_COUNT 128

// Floating point ops per inner loop iteration

#define FLOPSPERCALC 2

// define some arrays - 64 byte aligned for fastest cache
access

REAL fa[BW_ARRAY_SIZE] __attribute__((align(64)));
REAL fb[BW_ARRAY_SIZE] __attribute__((align(64)));
REAL fc[BW_ARRAY_SIZE] __attribute__((align(64)));
```

```
int main(int argc, char *argv[] )  {
        int i,j,k;
        double tstart, tstop, ttime;
           double gflops = 0.0;
           float a=1.1;

           printf("Initializing¥r¥n");
#pragma omp parallel for
         for(i=0; i<FLOPS_ARRAY_SIZE; i++){
              fa[i] = (float)i + 0.1;
                 fb[i] = (float)i + 0.2;
         }

         printf("Starting Compute¥r¥n");
#pragma omp parallel
#pragma omp master
printf("Starting BW Test on %d threads¥r¥n",omp_get_num_threads());
         tstart = dtime();
         for (i=0; i<BW_ITERS; i++) {
#pragma omp parallel for
          for(k=0; k<BW_ARRAY_SIZE; k++) {
                fa[k] = fb[k];
          }
          }
          tstop = dtime();

        // # of gigaflops we just calculated
gbytes = (double)( 1.0e-9 * OPS_PERITER * BW_ITERS *
BW_ARRAY_SIZE*sizeof(REAL));

        // elasped time
        ttime = tstop - tstart;

        if ((ttime) > 0.0) {
          printf("Gbytes = %10.3lf, Secs = %10.3lf,"
          "Gbytes per sec - %10.3lf¥r¥n",
          gbytes, ttime, gbytes/time);
        }
        return( 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。

5 番目のコードをコンパイル (ネイティブモデル)

スレッドとコアへのコードのスケールをサポートするため、ここでは再び OpenMP* を利用

> icl -openmp /Qmic -qopt-report=3 /qopt-report-phase=vec -O3 hellomem.c -o hellomem

すべてのコアでコードを実行することもできるが、ここでは最初にスケーラビリティー曲線を確認します。hellomem をアップロード後、コプロセッサーで次のコマンドを実行:

export OMP_NUM_THREADS=1

192.168.1.100 - PuTTY

```
[root@mic0 ~]# ./hellomem_xphi
Initializing
Starting BW Test on 1 threads
Gbytes =   1024.000, Secs =    197.209, GBytes per sec =      5.192
[root@mic0 ~]#
```

©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。

5 番目のコードを異なるスレッド数で実行 (ネイティブモデル)

192.168.1.100 - PuTTY

```
[root@mic0 ~]# export KMP_AFFINITY=scatter
[root@mic0 ~]# export OMP_NUM_THREADS=2
[root@mic0 ~]# ./hellomem_xphi
Initializing
Starting BW Test on 2 threads
Gbytes =   1024.000, Secs =    101.078, GBytes per sec =     10.131
[root@mic0 ~]#
```

OMP_NUM_THREADS	Gbytes	Secs	GBytes per sec
2	1024.000	101.078	10.131
10	1024.000	21.632	47.337
20	1024.000	15.331	66.792
30	1024.000	11.281	90.772
40	1024.000	10.119	101.197
50	1024.000	9.866	103.795
55	1024.000	9.933	103.093
57	1024.000	10.052	101.875

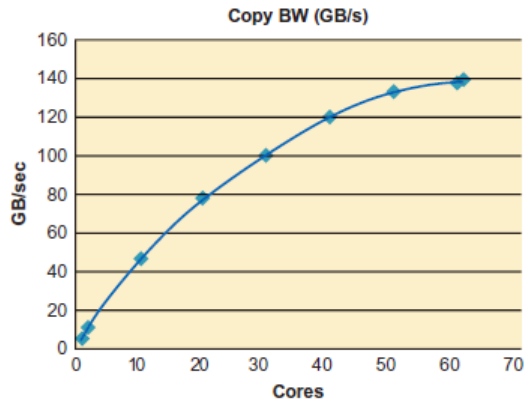
©2015 Intel Corporation. 無断での引用、転載を禁じます。* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

メモリー帯域幅に関する考察

コアの大部分を利用する時、毎秒ピークギガバイトの 40% に近い帯域幅を達成している

アプリケーションや用途に応じてさらなる改善をもたらす、いくつかのコンパイラー・スイッチ (オプション) が用意されている

また、インテル® Xeon Phi™ コプロセッサーとコプロセッサー用の Linux* オペレーティング・システムは、2MB メモリーページをサポートする



テスト・コプロセッサーでの実行結果をグラフにしたものです。30 から 40 コアあたりで、1 秒当たりのデータ転送量を示す曲線が平坦化し始めていることが見てとれる。そして、利用可能なコアの大部分を利用するまで転送量は増加し続けている

©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。

まとめ

- インテル® Xeon Phi™ コプロセッサーで利用可能な最高の計算性能を達成するには、ベクトル処理ユニットによってもたらされるローカルデータ並列性を活用するため、コードのベクトル化を確実にすることが重要
- 1 つの計算で乗算と加算を行っている場合、インテル® Xeon Phi™ コプロセッサーは、一般的な線形代数や行列乗算に基づく数学、技術計算をサポートするため乗算と加算の融合命令 (FMA)を持っている。コードでこの命令を広域に活用できた場合、コプロセッサーは計算ピーク性能に近い能力を発揮できる
- サンプルコードを改良することで、インテル® Xeon Phi™ コプロセッサーのピーク性能に近い性能を達成し、同様にコプロセッサーの高速メモリー・サブシステムを体験した。GDDR5 メモリーデバイスを利用することで、コプロセッサーは 240 GB/秒以上のピーク帯域幅を持ち、効率良いアクセス可能な帯域幅は、ピークメモリー帯域幅の 50 – 60% であることを指摘した
- 各コアは、32 個の内部メモリー管理バッファーを持ち、1 つのコアの帯域幅は 8 – 10 GB/秒に制限される
- これらの知識と簡単なメモリーコピーを行うサンプルコードを通じて、ピークメモリー帯域幅のおよそ 40% を達成することができた

©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。

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