開発者用 Knights Landing† ガイド

第 2 世代インテル® Xeon Phi™ プロセッサーの概要

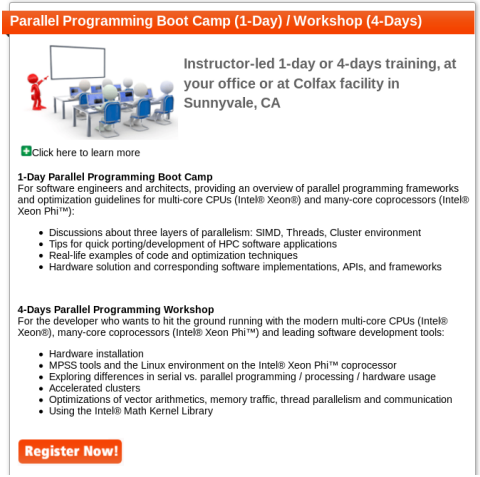
Colfax International — @colfaxintl

2016 年 4 月 - 改訂 1.2

†開発コード名

このドキュメントについて

このドキュメントは、Colfax International 社の Web ベースのトレーニング、「第 2 世代インテル® Xeon Phi™ プロセッサーの概要: 開発者用 Knights Landing† ガイド」の資料です。



colfaxresearch.com/knl-webinar/

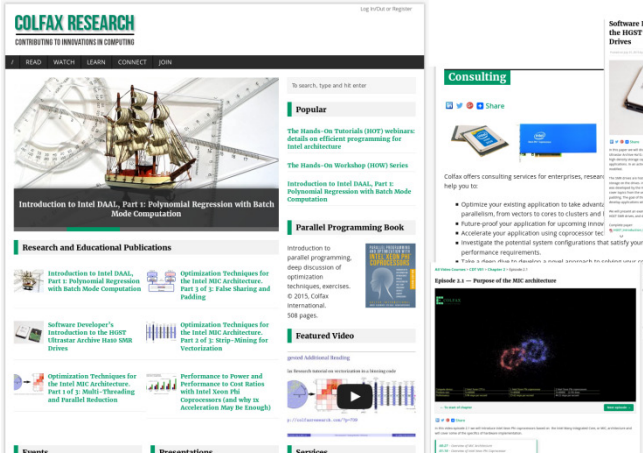
†開発コード名

法務上の注意書き

本トレーニングの準備には最善を尽くしていますが、Colfax International は、内容の正確さや完全性について、いかなる表明または保証もいたしません。また、いかなる責任も負いません。特に、商品適格性または特定目的への適合性の黙示的保証はいたしません。本資料に含まれる情報またはプログラムにより、直接的または間接的に生じた一切の損失、間接的または結果的損害、あるいは損失の申し立てについて、本資料の発行元は一切責任を負いません。販売担当者または販売促進資料は、一切の保証またはその追加・延長を行うものではありません。

関連文書およびビデオ

Colfax Research



http://colfaxresearch.com/

(登録済みの方は参考書が $10 割引になります)

HOW シリーズ: 無料ウェビナー (英語)

ご興味がある方はこちらからサインアップしてください:

colfaxresearch.com/how-series

Developer Access Program (DAP)

Knights Landing† 早期アクセスシステム受注中

詳細は、dap.xeonphi.com をご覧になるか、

dap@colfax-intl.com までお問い合わせください

†開発コード名

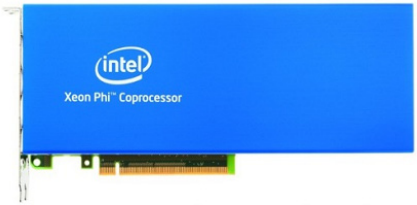
§2. インテル® アーキテクチャー: 現在と将来

インテル® アーキテクチャー

インテル® Xeon® プロセッサー

現在: Broadwell†
次世代: Skylake†

第 1 世代インテル® Xeon Phi™ コプロセッサー



現在: Knights Corner (KNC)†

第 2 世代インテル® Xeon Phi™ プロセッサー*

* ソケットとコプロセッサーのバージョン

次世代: Knights Landing (KNL)†

マルチコア・アーキテクチャー

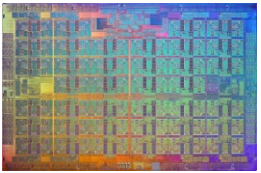
インテル® メニー・インテグレーテッド・コア (インテル® MIC) アーキテクチャー

† 開発コード名

インテル® Xeon Phi™ プロセッサー (第 2 世代)

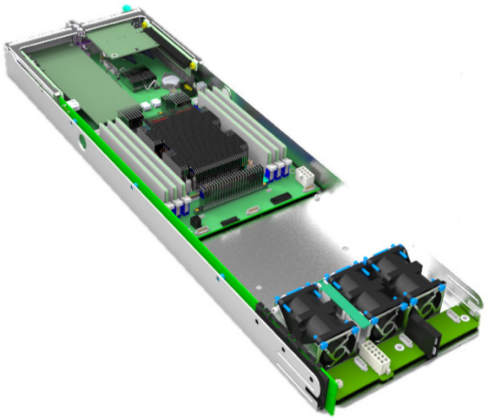
第 2 世代インテル® MIC アーキテクチャー
計算集約型アプリケーション向けのプラットフォーム

- ブート可能なホスト・プロセッサーまたは
  コプロセッサー
- 3+ TFLOP/秒 DP
- 6+ TFLOP/秒 SP
- 最大 16GiB MCDRAM
- MCDRAM 帯域幅 ≈ DDR4 の 5 倍
- インテル® Xeon® プロセッサーとバイナリー互換
- 公開情報



標準の CPU フォームファクター

- ブート可能なホスト・プロセッサー
  - “ホスト” は不要
    KNL† プロセッサー上で OS を実行
  - 一般的な OS をサポート
  - PCIe ボトルネックなし
- 最大 384GiB DDR4 RAM への直接アクセス
  - 最大 ≈ 90GB/秒の DDR4 帯域幅
- PCIe バスへのアクセス

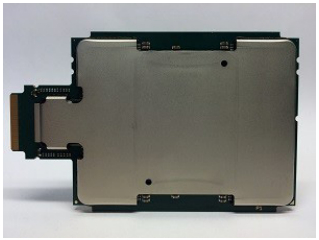


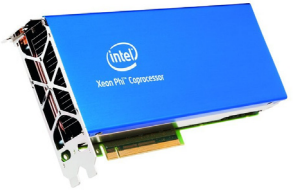
†開発コード名

今後リリース予定の製品

KNLF: ファブリック付き KNL†

- CPU 上にファブリックを統合
  - インテル® Omni-Path アーキテクチャー
- ソケット・マウント・プロセッサー





*KNC† のイメージ

KNL† コプロセッサー

- PCIe アドインカード
  - ホストが必要
- 1 システムに複数の KNL† を搭載可能

†開発コード名

§3. コアとスレッド

並列処理の重要性

KNL† をスチームエンジンに例えると...

火室が多数ある

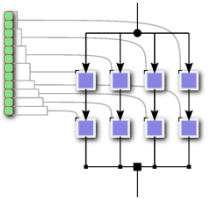
KNL† でシングルスレッド・コードを実行することは
1 人の火夫が 72 個のスチームエンジンに
燃料を投入しているようなもの

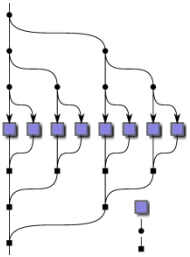
†開発コード名

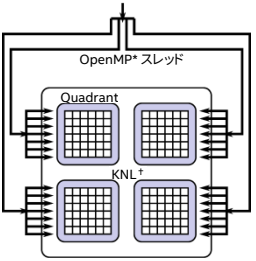
マルチスレッドの実装
コアの能力を引き出すにはコアを利用しなければならない

スレッド化のフレームワーク:

- OpenMP*
- インテル® TBB
- インテル® Cilk™ Plus
- Pthread
- MPI とのハイブリッド
- 基本、インテル® Xeon® プロセッサーでサポートされる手法すべてに対応







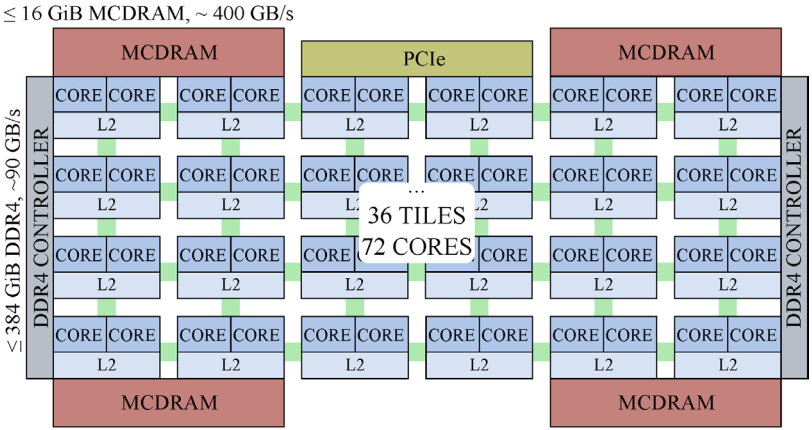
†開発コード名

KNL† コアの機能

†開発コード名

KNL† ダイの構成: タイル

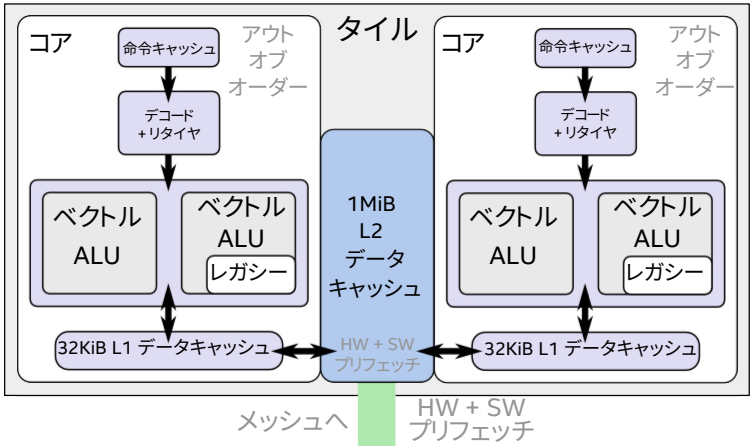
- 最大 36 タイル、タイルごとに 2 つの物理コア (合計 76 コア)
- メッシュ・インターコネクトによる通信、分散 L2 キャッシュ



†開発コード名

KNL† コア

- 4-way ハイパースレッディング (最大 4 × 72 = 288 論理プロセッサー)
- L2 キャッシュはタイル上の 2 つのコアで共有される



†開発コード名

第 1 世代 (KNC†) よりも寛容なコア

インテル® Atom™ プロセッサーのコア (Silvermont† マイクロアーキテクチャー) ベース

- **アウトオブオーダー・コア:**
  レイテンシーの長い操作においてより優れたレイテンシーのマスキング
- **シングルスレッドからの連続命令:**
  必要なスレッド数が ≈ 70 に減少 (KNC† では ≈ 120)
- **高度な分岐予測:**
  分岐予測ミスにより無駄になるサイクル数が減少

一般に、最適化されていないコードに対してより寛容

†開発コード名

パフォーマンスに関する考察

アフィニティーと cpuinfo

隣接するスレッドは隣接する/同じメモリー位置を操作することがよくある

→ スレッドピニングにより L2 キャッシュを共有させる

`cpuinfo` (インテル® MPI ライブラリーの機能) を利用してキャッシュを共有しているコアを特定できる

```
user@knl% cpuinfo
// ...cpuinfo  の出力...//
L2 1 MB (0,1,64,65,128,129,192,193)(2,3,66,67,130,131,194,195)(4,5,68, ...
```

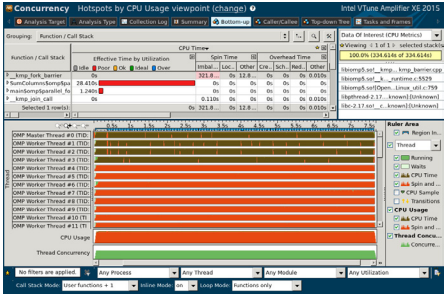
`KMP_AFFINITY` (インテル® コンパイラー) または `OMP_PROC_BIND` (GCC コンパイラー) を使用する

```
user@knl% export KMP_AFFINITY=compact
user@knl% export OMP_PROC_BIND=close
```

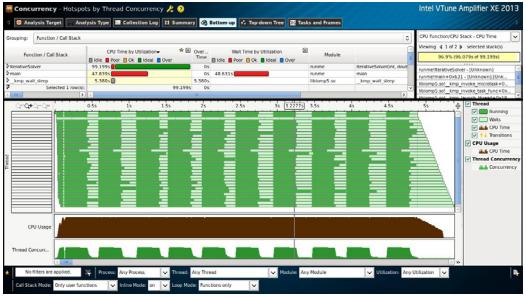
スレッドのチューニング

マルチスレッド・コードでも、次のような問題に注意が必要...

- 並列処理の不足



- ロード・インバランス



詳細は、colfaxresearch.com/how-series をご覧ください

§4. ベクトル化

スカラーコードと火夫

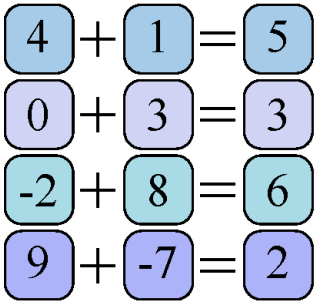
KNL[†] でベクトル命令を利用しないことは
スチームエンジンにスプーンで燃料を
投入しているようなもの

[†]開発コード名

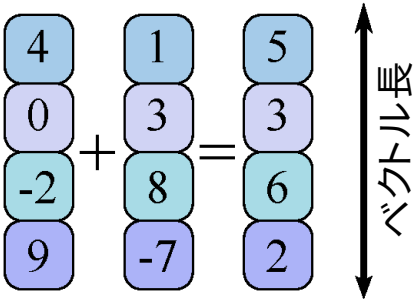
ショートベクトルのサポート

ベクトル命令 – SIMD (Single Instruction Multiple Data) 並列処理の実装の 1 つ

スカラー命令



ベクトル命令



KNL† 上のベクトル命令

†開発コード名

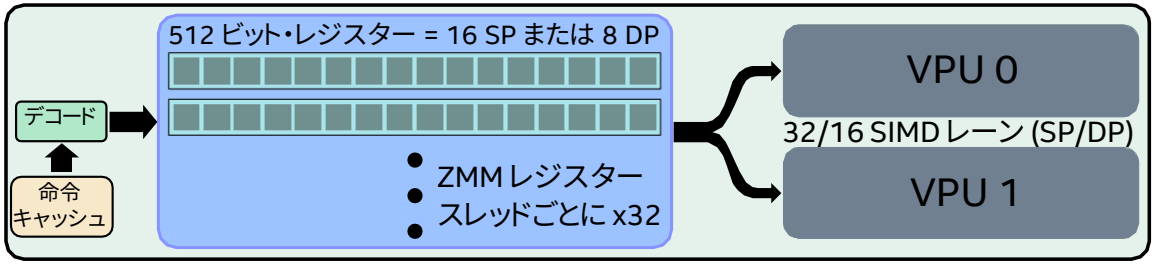
デュアル VPU

KNL† 上の各コアには 2 つのベクトル演算ユニット (VPU) がある

ベクトル化されていないコードのペナルティー

SP → 512 ビット・レジスター/32 ビット × 2 VPU = **32** SIMD レーン

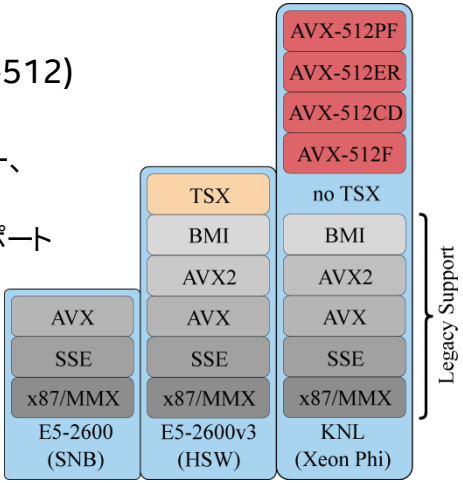
DP → 512 ビット・レジスター/64 ビット × 2 VPU = **16** SIMD レーン



†開発コード名

KNL† でサポートされるベクトル命令セット

- インテル® アドバンスト・ベクトル・エクステンション 512 (インテル® AVX-512)
  - 512 ビット・ベクトル・レジスター
  - ハードウェアによるギャザー/スキャッター、DP 超越関数サポートなど
  - GCC などの他社製コンパイラーでもサポート
- インテル® AVX2 以前
  - レガシーモード操作
  - インテル® Xeon® プロセッサーとバイナリー互換
  - IMCI (KNC†) は**含まない**



†開発コード名

インテル® AVX-512 の機能

Knights Landing†: 最初のインテル® AVX-512 対応プロセッサー

- インテル® AVX-512F (基本命令)
  - 多くのインテル® AVX2 命令の 512 ビット・レジスター拡張
- インテル® AVX-512CD (競合検出命令)
  - 効率良い競合検出 (例: ビニング)
- インテル® AVX-512ER (指数および逆数命令)
  - 超越関数 (exp、rcp、および rsqrt) のサポート
- インテル® AVX-512PF (プリフェッチ命令)
  - スキャッター/ギャザー用のプリフェッチ

†開発コード名

インテル® AVX-512 サポートの確認方法

/proc/cpuinfo でインテル® AVX-512 のフラグを確認

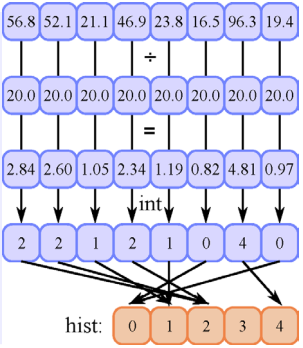
```
user@knl% cat /proc/cpuinfo
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm
constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc  aperfmperf
eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 fma  cx16 xtpr pdcm
sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer xsave avx  f16c rdrand
lahf_lm abm 3dnowprefetch arat epb xsaveopt pln pts dtherm  tpr_shadow vnmi
flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2  erms  avx512f
rdseed adx  avx512pf avx512er avx512cd
```

C/C++ 関数呼び出しにより確認することも可能: こちらのブログをご覧ください

†開発コード名

インテル® AVX-512CD: ヒストグラム

```
1  // Colfax 問題集 4.04 の worker.cc
2  void Histogram(const float* age, int* const hist,
3                 const int n, const float group_width,
4                 const int m) {
5
6    for (int i = 0; i < n; i++)  {
7      const int j= (int) ( age[i] / group_width );
8      hist[j]++;
9    }
10 }
```




```
user@knl% cat worker.optrpt
....
リマーク ...: ベクトル化のサポート: 分散 (scatter) が生成されました (変数 hist:)
リマーク ...: ベクトル化のサポート: 集約 (gather) が生成されました (変数 hist:)
リマーク #15300: ループがベクトル化されました
```

†開発コード名

インテル® AVX-512ER: 超越関数

超越関数のサポート

- 指数
- 逆数
- 逆平方根

精度の向上

- 倍精度
- 相対誤差の最大値:
- 2^{-23} (exp)
- 2^{-28} (rcp および rsqrt)

出典: APOD

†開発コード名

プログラミングに関する考察

インテル® AVX-512 の使用: 2 つのアプローチ

自動ベクトル化:

- コンパイラーによるベクトル化
- 移植性が高い: 再コンパイルするだけ
- ディレクティブを利用してチューニング

```
double A[vec_width], B[vec_width];
// ...

// このループは自動ベクトル化される
for(int i = 0; i <
  vec_width; i++)
  A[i]+=B[i];
```

```
double A[vec_width], B[vec_width];
// ...
// 明示的なベクトル化
__m512d A_vec = _mm512_load_pd(A);
__m512d B_vec = _mm512_load_pd(B);
A_vec = _mm512_add_pd(A_vec,B_vec);
_mm512_store_pd(A,A_vec);
```

明示的なベクトル化:

- 組込み関数によるベクトル化
- 組込み関数を完全に制御
- 移植性が制限される

インテル® コンパイラーのインテル® AVX-512 サポート

インテル® コンパイラー 15.0 以上はインテル® AVX-512 命令セットをサポート

```
user@knl% icc -v
icc 16.0.1 (gcc 4.8.5 互換)
user@knl% icc -help
// ...出力の一部抜粋...//
-x<code>
     ...
     MIC-AVX512
     CORE-AVX512
     COMMON-
     AVX512
```

- -xMIC-AVX512: KNL† 向け (F、CD、ER、PF をサポート)
- -xCORE-AVX512: 将来のインテル® Xeon® プロセッサー向け (F、CD、DQ、BW、VL をサポート)
- -xCOMMON-AVX512: KNL† およびインテル® Xeon® プロセッサー共通 (F、CD をサポート)

†開発コード名

GCC のインテル® AVX-512 サポート

GCC 4.9.1 以上はインテル® AVX-512 命令セットをサポート

```
user@knl% g++ -v
gcc 4.9.2 (GCC)
user@knl% g++ foo.cc -mavx512f -mavx512er -mavx512cd -mavx512pf
```

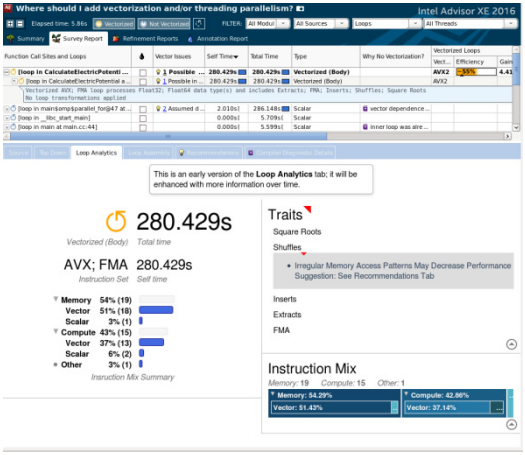
基本的な自動ベクトル化のサポート: -O2 または -O3 を追加

```
1 // ...foo.cc ...//
2 for(int i=0; i < n; i++)
3    B[i] = A[i] + B[i];
```

```
user@knl% g++ -s foo.cc -mavx512f -03
user@knl% cat  foo.s
...
vmovapd  -16432(%rbp,%rax), %zmm0
vaddpd  -8240(%rbp,%rax), %zmm0, %zmm0
vmovapd  %zmm0, -8240(%rbp,%rax)
```

パフォーマンスに関する考察

コードがベクトル化されていても、チューニングによりパフォーマンスを
さらに向上できる可能性がある



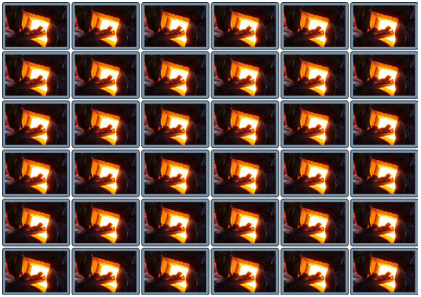
- 十分な並列処理
  - ベクトル命令のレイテンシーをカバーするには連続するベクトル命令が必要
- ループのパイプライン化とアンロール
  - VPU が 2 つあるため、パイプライン・ステージが 2 倍
- より優れたベクトル化パターン
  - ユニットストライドを含むレイテンシーの長い操作やマスクなし操作は避ける

§5. メモリー・アーキテクチャー

燃料の場所は?

データがなければコアは処理を実行できない

出典: wikipedia



コアにデータを渡すため、KNL† メモリーを効率良く使用する必要がある

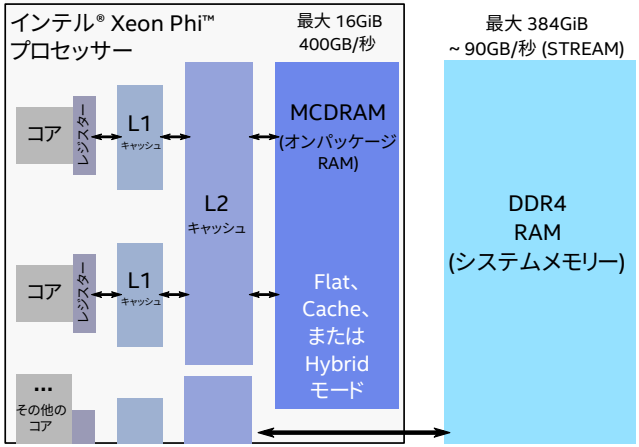
†開発コード名

KNL† の MCDRAM

†開発コード名

KNL† のメモリー構成

- オンパッケージの MCDRAM とシステム DDR4 (ソケット) への直接アクセス
- MCDRAM を Cache、Flat、または Hybrid モードで利用する

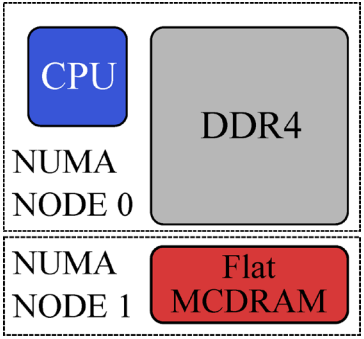


†開発コード名

MCDRAM メモリーモード

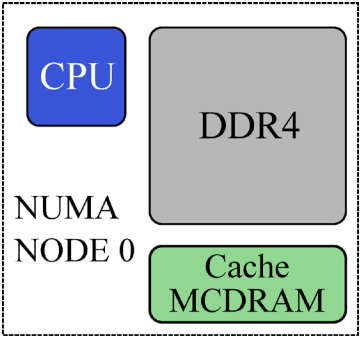
Flat モード

- MCDRAM は NUMA ノードとして扱われる
- MCDRAM の使用はユーザーが制御



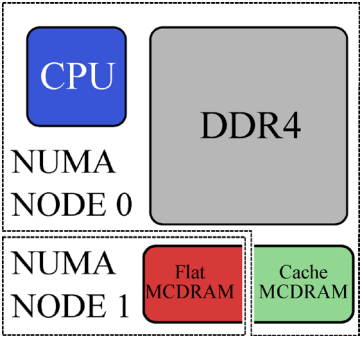
Cache モード

- MCDRAM は LLC として扱われる
- 自動で MCDRAM が使用される



Hybrid モード

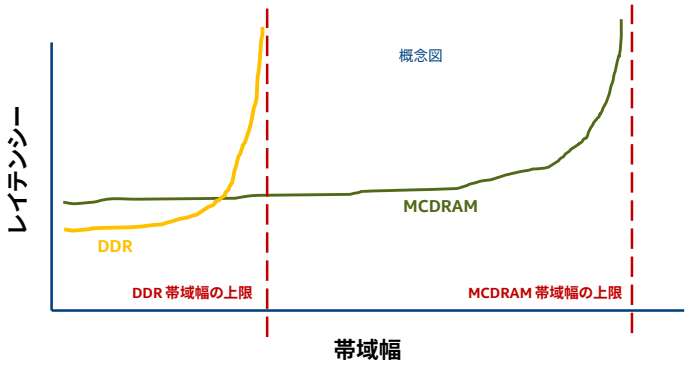
- Flat モードと Cache モードの組み合わせ
- 比率は BIOS で指定可能



†開発コード名

MCDRAM のパフォーマンス

DDR と MCDRAM の帯域幅とレイテンシー



負荷が低い場合は MCDRAM のほうが DDR よりも高レイテンシーで、
負荷が高い場合は MCDRAM のほうが DDR よりも低レイテンシー

出典: インテル、ISC 2015 KNL† 基調講演 (pdf)

†開発コード名

MCDRAM を利用するプログラミング

Numactl

- システムの NUMA ノードに関する情報を取得

```
user@knl% # All-to-All の Flat モード
user@knl% numactl -H
available: 2 nodes (0-1)
node 0 cpus: ...all cpus ...
node 0 size: 98207 MB
node 0 free: 94798 MB
node 1 cpus:
node 1 size: 16384 MB
node 1 free: 15991 MB
```

- アプリケーションを MCDRAM にバインド (Flat/Hybrid)

```
user@knl% gcc myapp.c -o runme -mavx512f -O2
user@knl% numactl -membind 1 ./runme
// ...MCDRAM  上で実行中のアプリケーション...//
```

Memkind ライブラリーと hbwmalloc

hbwmalloc と Memkind ライブラリーにより手動で MCDRAM 上に割り当てることができる

```
#include <hbwmalloc.h>
const int n = 1<<10;
// MCDRAM への割り当て
double* A = (double*) hbw_malloc (sizeof(double)*n);
// _mm_malloc の代替はない。posix_memalign を使用
double* B;
int ret = hbw_posix_memalign((void*) B, 64, sizeof(double)*n);
.....
// hbw_free で解放
hbw_free(A); hbw_free(b);
```

Fortran での割り当て

```
REAL, ALLOCATABLE :: A(:)
!DEC$ ATTRIBUTES FASTMEM :: A
ALLOCATE (A(1:1024))
```

Memkind ライブラリーと hbwmalloc を利用してコンパイル

C/C++ アプリケーションをコンパイルする場合:

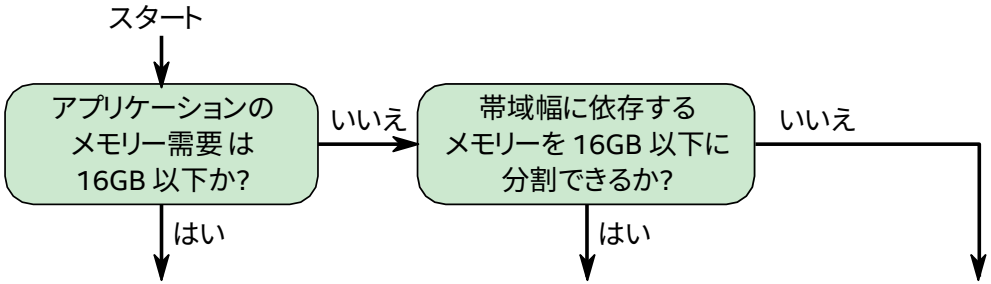
```
user@knl% icpc -lmemkind foo.cc -o runme
user@knl% g++  -lmemkind  foo.cc -o runme
```

Fortran アプリケーションをコンパイルする場合:

```
user@knl% ifort  -lmemkind foo.f90 -o runme
user@knl% gfortran    -lmemkind  foo.f90 -o  runme
```

Memkind ライブラリーのオープンソース版は、
http://memkind.github.io/memkind/ から入手可能

帯域幅に依存するアプリケーションのフローチャート



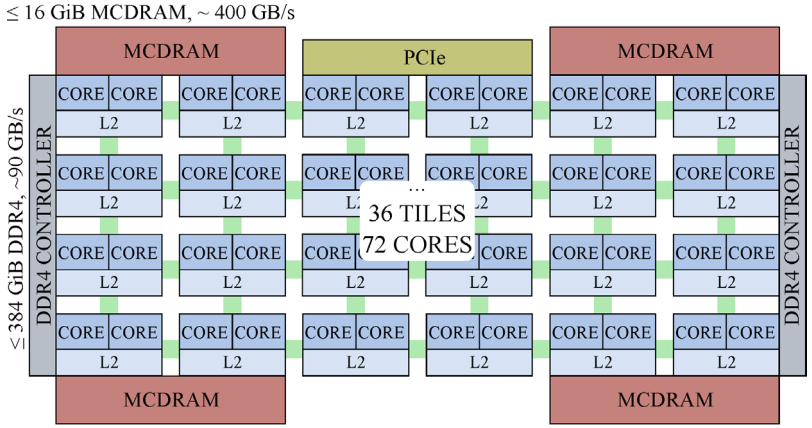
numactl	Memkind	Cache モード
• プログラム全体を MCDRAM で実行 • コード変更不要	• 手動で帯域幅に依存するメモリーを MCDRAM に割り当て • Memkind の呼び出しを追加する必要あり	• OS に MCDRAM の使用法を任せる • コード変更不要

KNL† 上のクラスターモード

†開発コード名

KNL⁺ のダイ構成

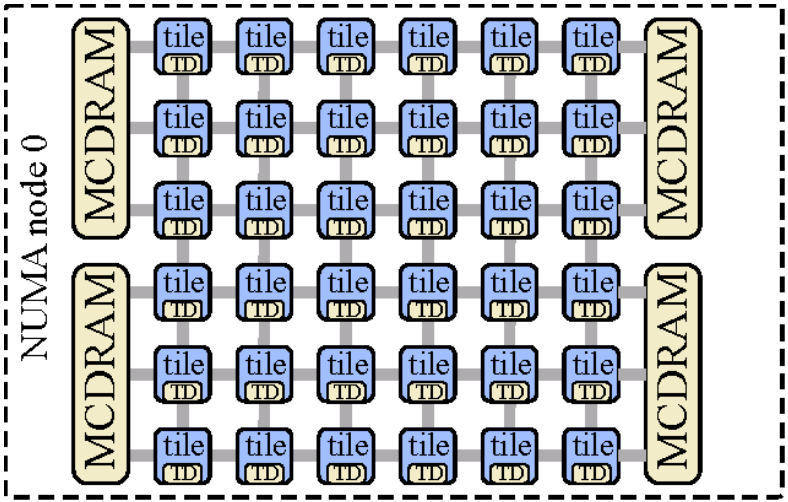
- メッシュ・インターコネクトによりデータの局所性要件が緩和される
- メッシュでの All-to-All、Quadrant 、Sub-NUMA ドメイン通信



⁺開発コード名

クラスターモード: All-to-All

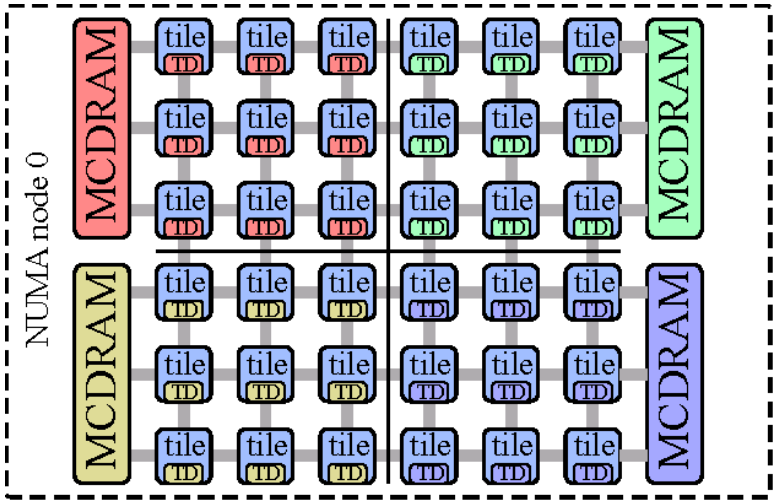
分散タグ・ディレクトリー (TD) とメモリーの間にアフィニティーなし



†開発コード名

クラスターモード: Quadrant/Hemisphere

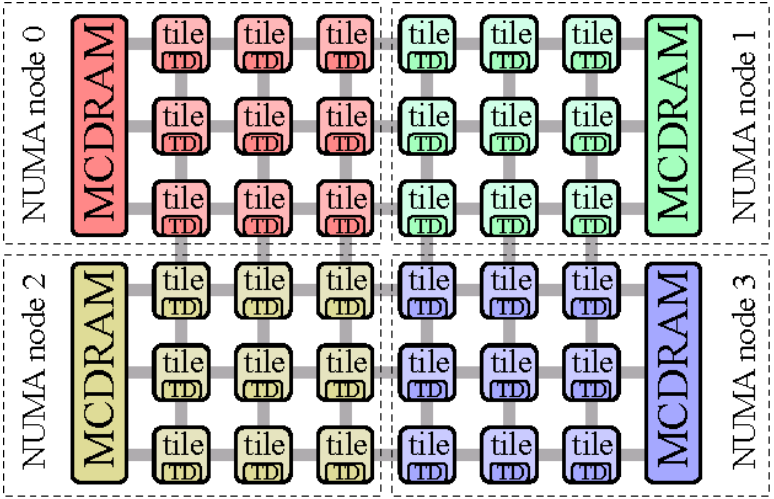
タグ・ディレクトリー (TD) とメモリーが同じ Quadrant にある



†開発コード名

クラスターモード: SNC-4/SNC-2

4 つの NUMA ノードとして扱われる (クアッドソケット・システムに似ている)



†開発コード名

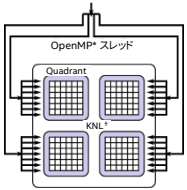
クラスターモードを利用するプログラミング

クラスターモードの利用法

スレッド・アフィニティーとメモリー/ディレクトリーのアフィニティーを一致させる

入れ子並列処理 (OpenMP*)

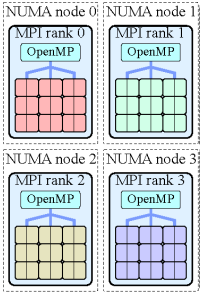
```
1  #pragma omp parallel
2  {
3    // ...
4  #pragma omp parallel
5    {
6      // ...
7    }
8  }
```




```
user@knl% OMP_NUM_THREADS=4,72
user@knl% OMP_NESTED=1
```

MPI + OpenMP*

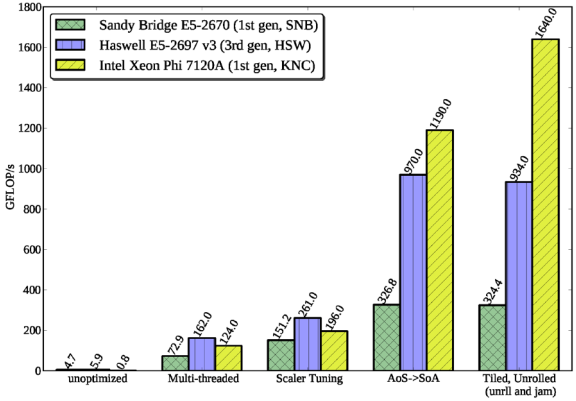
```
1  stat = MPI_Init();
2  // ...
3  #pragma omp parallel
4  {
5      // ...
6  }
7  // ...
8  MPI_Finalize();
```




```
user@knl% mpirun -host knl \
>   -np 4 ./myparallel_app
```

§6. コード最適化の重要性

インテル® アーキテクチャー上での N 体シミュレーション



† 開発コード名

KNL† 向けコードの準備

最良の方法は KNC† 向けにコードを最適化すること

関連情報

HOW シリーズ: 無料ウェビナー (英語)

ご興味がある方はこちらからサインアップしてください:

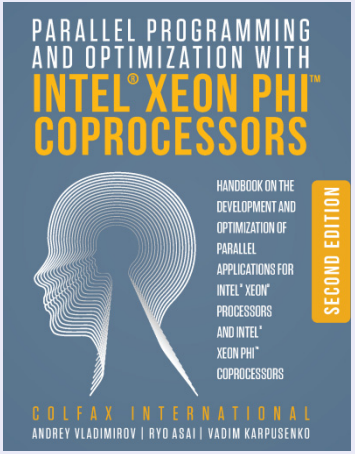
colfaxresearch.com/how-series

参考書

ISBN: 978-0-9885234-0-1 (508 ページ、電子版または印刷版)

Parallel Programming
and Optimization with
Intel® Xeon Phi™
Coprocessors

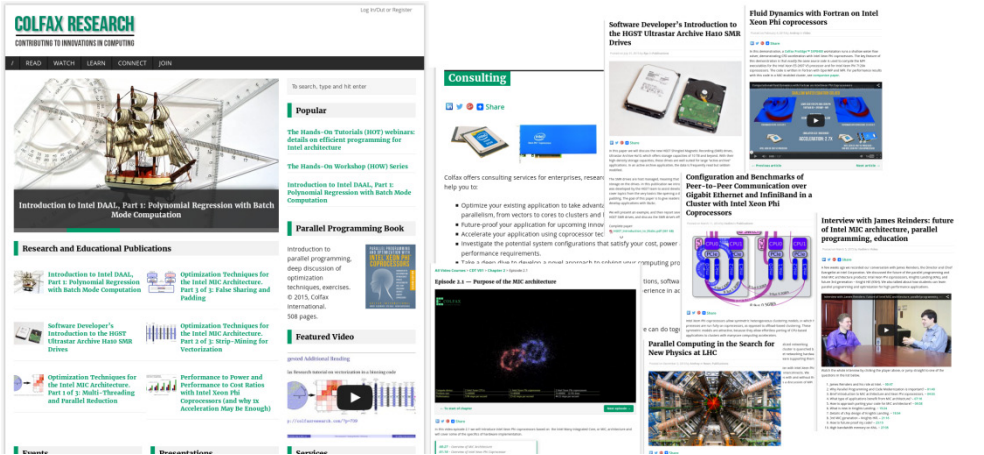
インテル® Xeon® プロセッサーおよび
インテル® Xeon Phi™ コプロセッサー向けの
並列アプリケーションの開発と
最適化に関するハンドブック



http://xeonphi.com/book

§7. まとめ

Colfax Research



http://colfaxresearch.com/

(登録済みの方は参考書が $10 割引になります)

Developer Access Program (DAP)

Knights Landing† 早期アクセスシステム受注中

詳細は、dap.xeonphi.com をご覧になるか、

dap@colfax-intl.com までお問い合わせください

†開発コード名

要点

KNL† は**高度な並列性**を備えた KNC† の後継製品であり
パフォーマンスと**使いやすさ**が向上しています

†開発コード名

ありがとうございました!