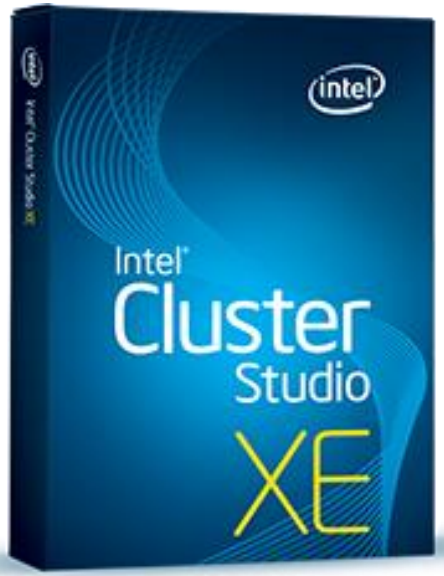
インテル® Xeon Phi™ コプロセッサー搭載システムの紹介およびオフロード・プログラミングとネイティブ実行の概要

インテル® ソフトウェア開発製品の紹介

インテル® ソフトウェア開発製品



Advanced Performance

C++ および Fortran コンパイラー
インテル® MKL/インテル® IPP ライブラリーと解析ツール
IA ベース・マルチコア・ノード上の Windows* および Linux* 開発者向け

Distributed Performance

MPI クラスターツールと C++ および Fortran コンパイラー、インテル® MKL/インテル® IPP ライブラリーと解析ツール
IA ベースのクラスター上の Windows* および Linux* 開発者向け

インテル® ソフトウェア開発製品
（インテル Xeon ® Phi™ コプロセッサー対応ツール）

インテル® Paralel Studio XE 2013

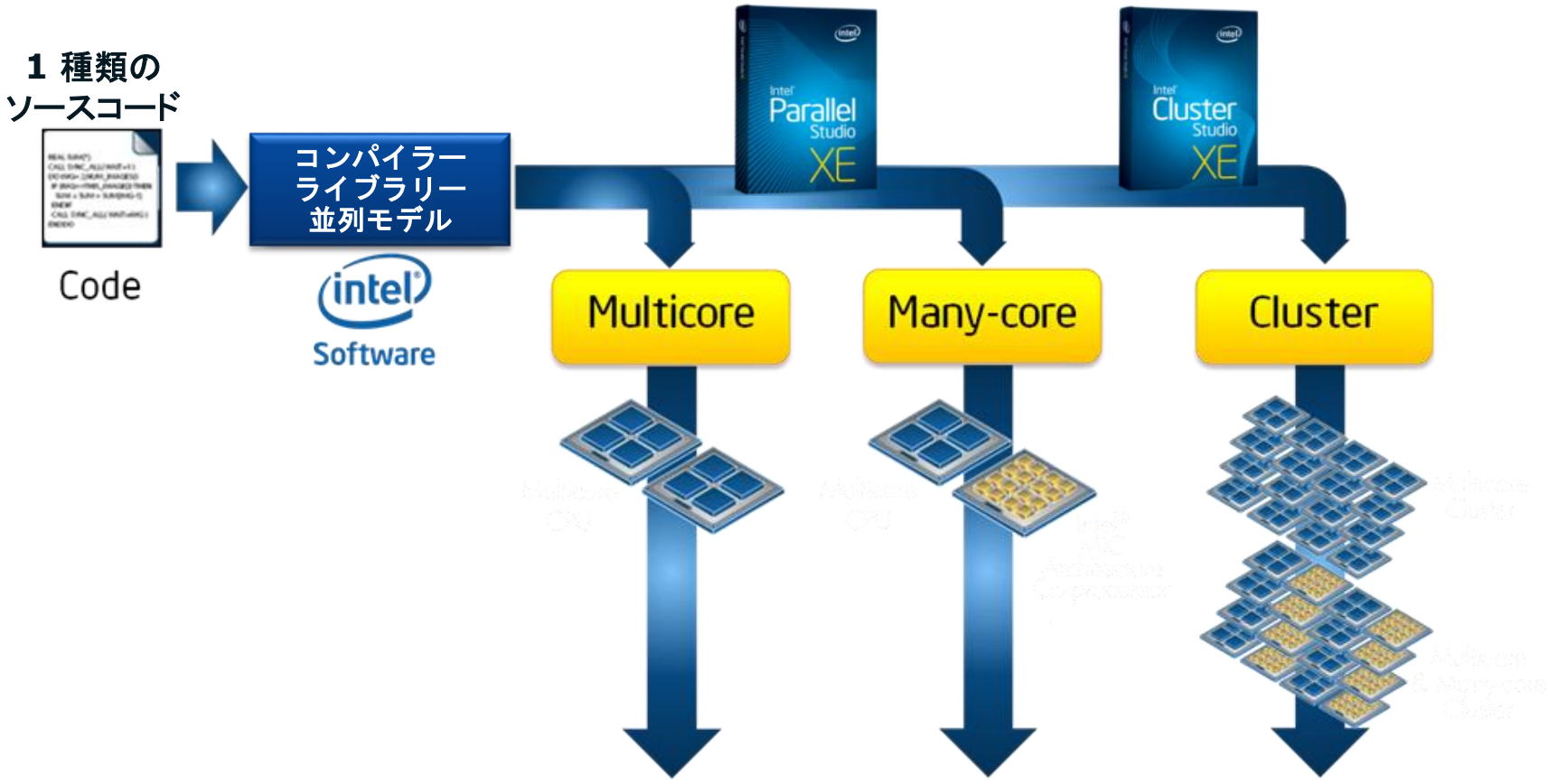
- インテル® Advisor XE
- ✓ インテル® C++ コンパイラー
- ✓ インテル® Fortran コンパイラー
- ✓ インテル® MKL
- インテル® IPP
- ✓ インテル® TBB
- インテル® Inspector XE
- ✓ インテル® VTune ™ Amplifier XE

インテル® Cluster Studio XE 2013

- インテル® Advisor XE
- ✓ インテル® C++ コンパイラー
- ✓ インテル® Fortran コンパイラー
- ✓ インテル® MKL
- インテル® IPP
- ✓ インテル® TBB
- インテル® Inspector XE
- ✓ インテル® VTune ™ Amplifier XE
- ✓ インテル® MPI ライブラリー
- ✓ インテル® Trace Analyzer/Collector

- 詳細は各製品のリリースノートやドキュメント等をご参照ください

インテル® ソフトウェア開発製品の活用



共通のソースコードから複数のプラットフォームへの対応が可能

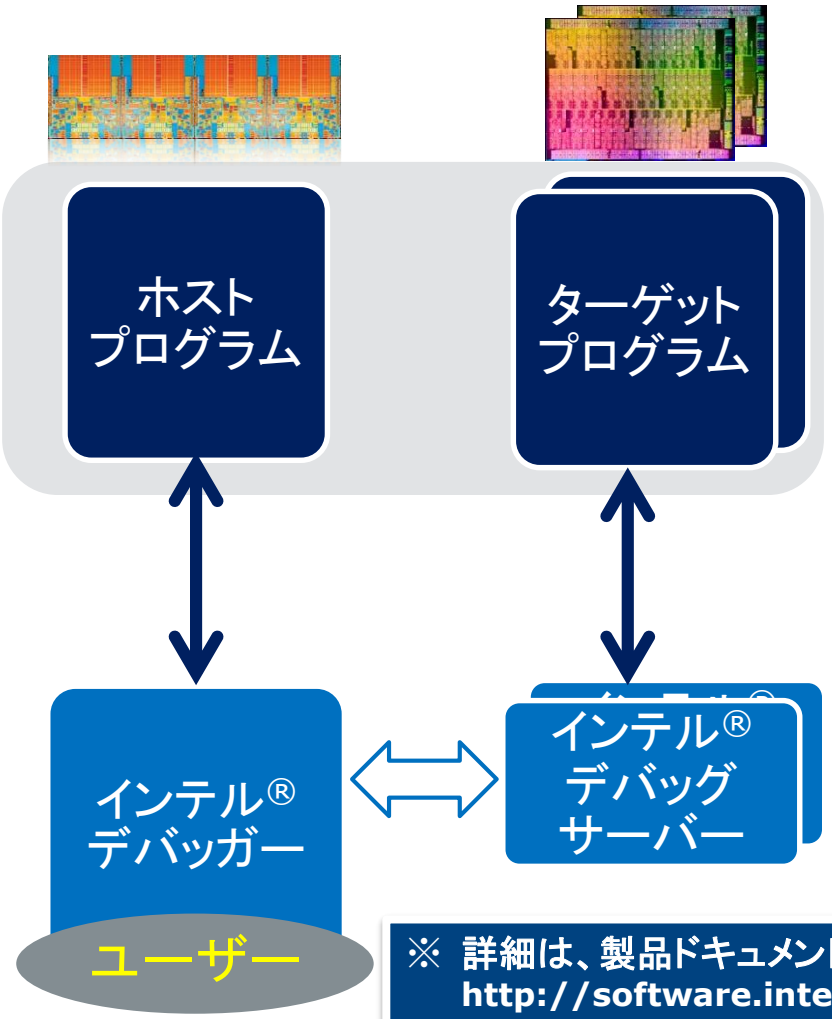
インテル® Composer XE

- ヘテロジニアス・コンパイルを提供するインテル® コンパイラー
- インテル® MIC 対応数値演算ライブラリー（MKL）
- インテル® MIC 対応インテル® デバッガー
- 並列化モデル言語拡張およびライブラリー

インテル® Composer XE でサポートする並列化モデル

† ノート：インテル® IPP は現在インテル® Xeon Phi™ コプロセッサー非対応

インテル® デバッガー（IDB）



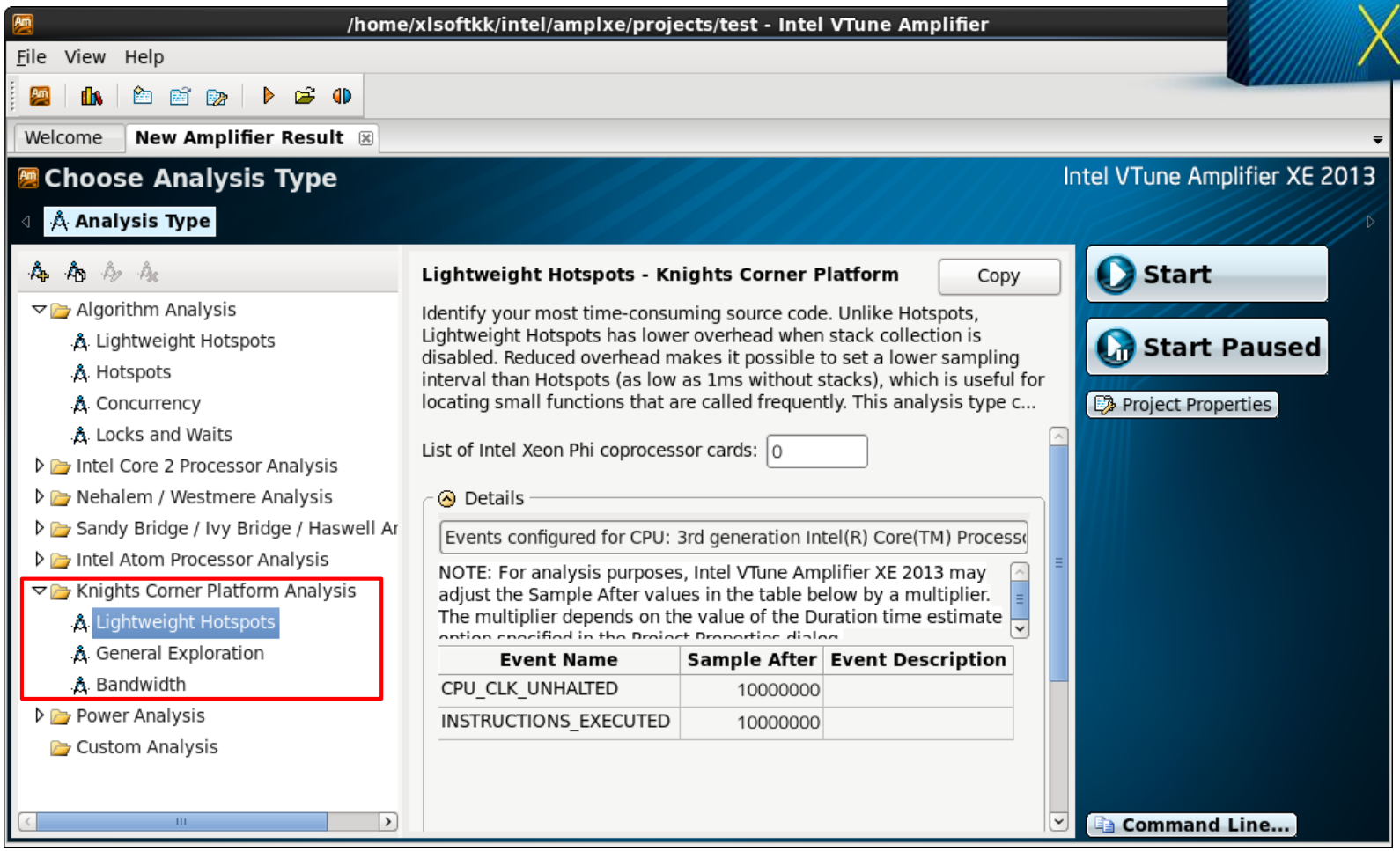
- ホストとターゲットを同時にデバッグ
- ホストのアプリケーションのデバッグを開始すると、ターゲットのアプリケーションは自動的にデバッグモードに入る
- ホストとターゲットのデバッグには、ホスト側のデバッガーを利用する
- デバッガーは、ホストとターゲットのプログラムを同期して、停止/再開できる
- 双方で C++ と FORTRAN を完全にサポート
- 将来のバージョンでは、1 つの GUI の中に 1 つの仮想アプリケーションのみを表示できるようにする
- 1 つ以上のオフロードカードをカバーするように拡張可能

※ 備考：ターゲットとは、インテル® Xeon Phi™ コプロセッサーを指す

※ 詳細は、製品ドキュメントまたは以下のサイトをご参照ください。
http://software.intel.com/en-us/articles/debugging-on-intel-xeon-phi-coprocessor-use-case-overview
http://software.intel.com/en-us/articles/debugging-intel-xeon-phi-coprocessor-targeted-applications-on-the-command-line

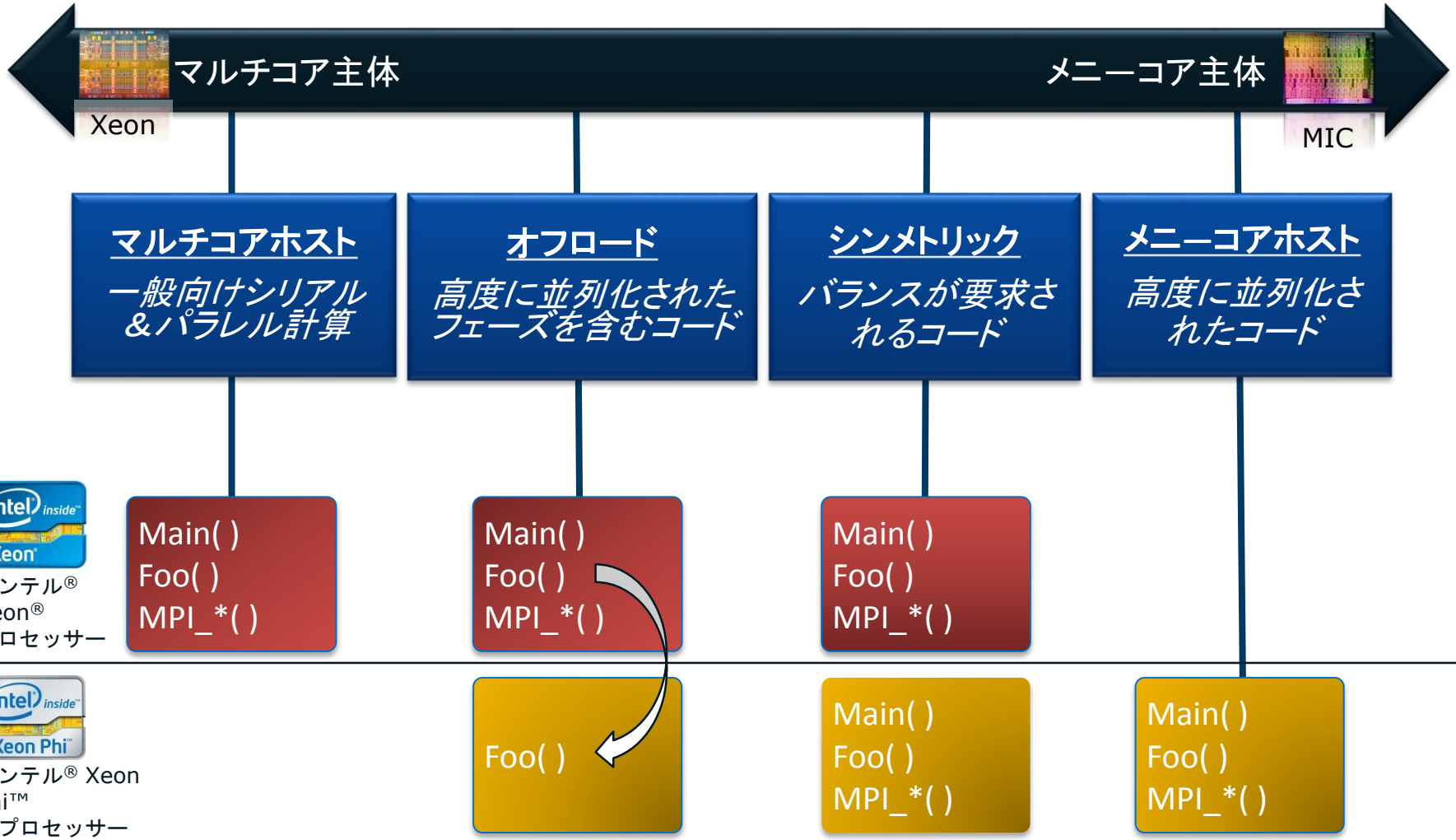
インテル® VTune™ Amplifir XE

インテル® VTune™ Amplifier XE パフォーマンス・プロファイラーは、オフロード実行およびネイティブ実行モデルにおけるインテル® MIC のパフォーマンスを解析します。



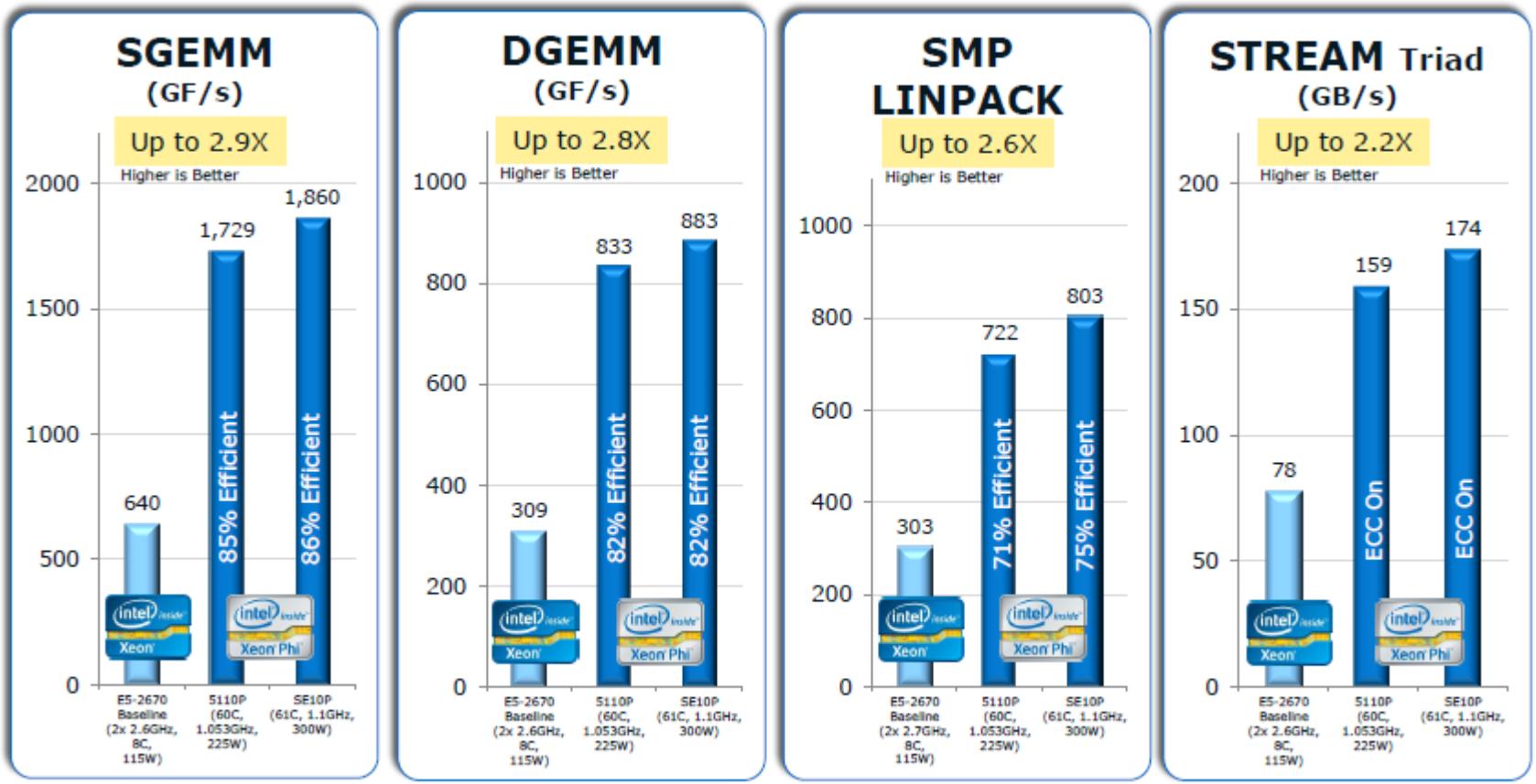
インテル® MIC 対応インテル® MPI

- 柔軟なプログラミング／実行モデルを提供



インテル® Xeon Phi™ コプロセッサー向け プログラミング概要

ベンチマーク

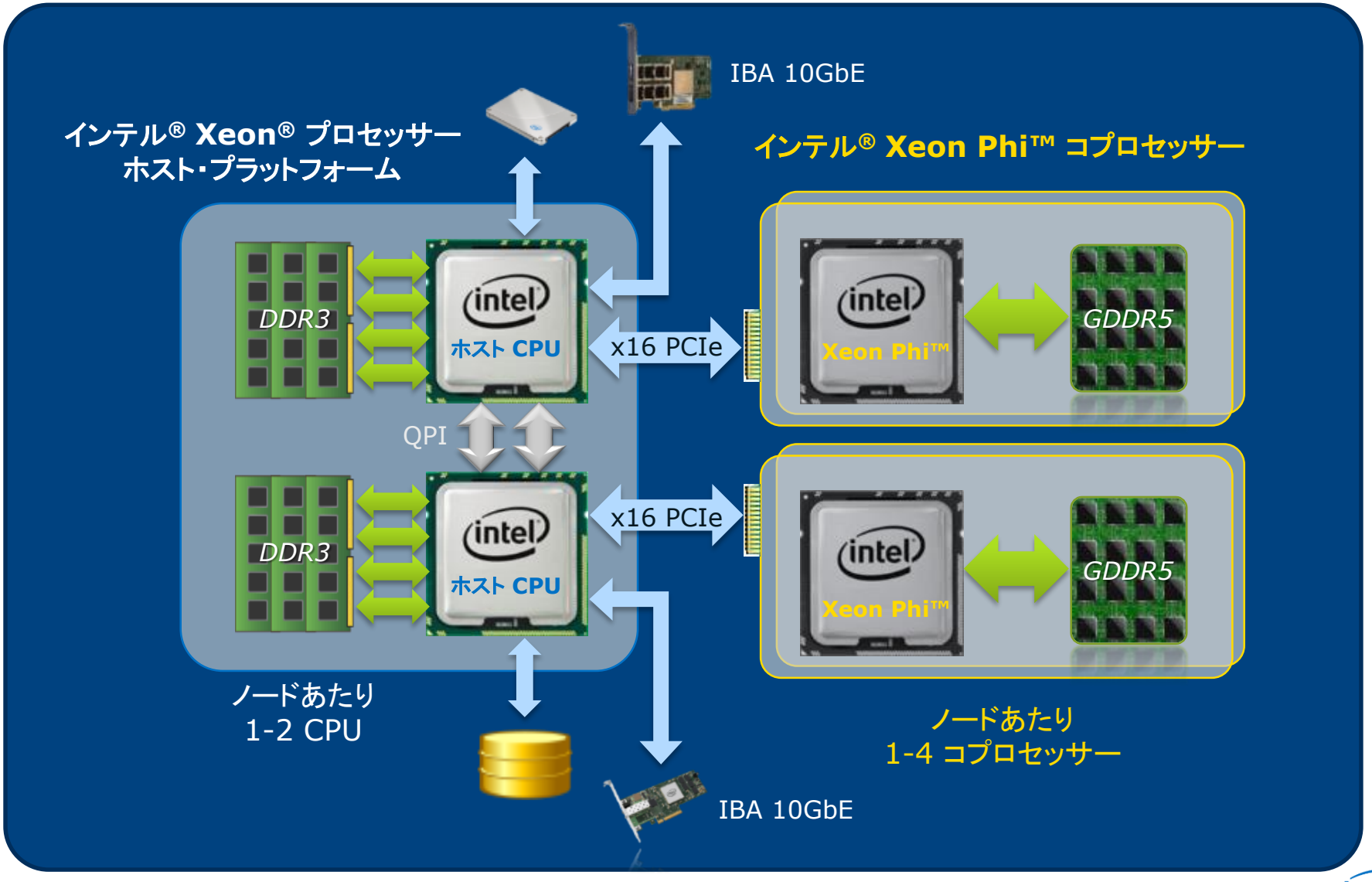


Coprocessor results: Benchmark run 100% on coprocessor, no help from Intel® Xeon® processor host (aka native)

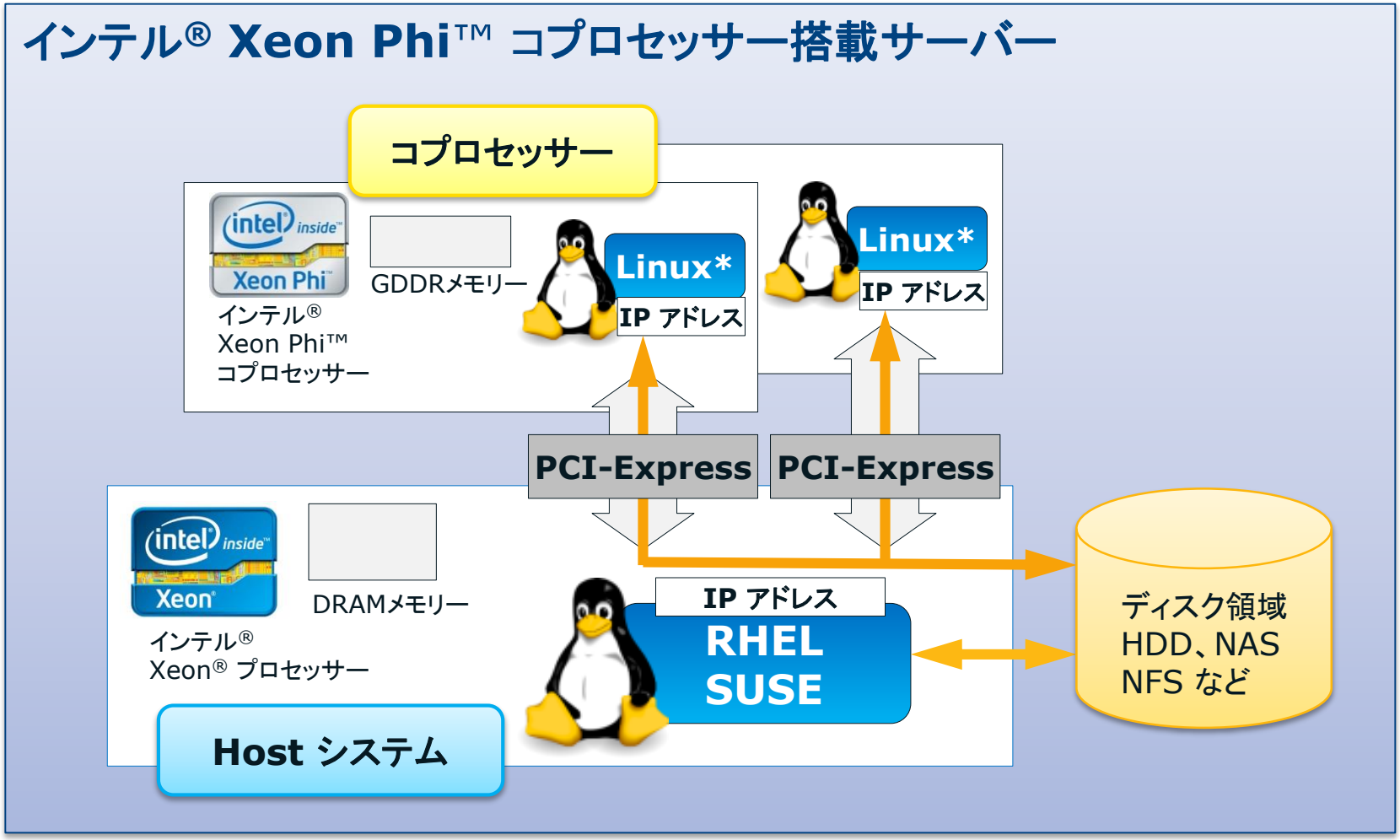
ベンチマーク詳細は以下のサイトをご参照ください。

- http://www.intel.com/content/dam/www/public/us/en/documents/performance-briefs/xeon-phi-product-family-performance-brief.pdf
- http://software.intel.com/en-us/intel-mkl#pid-12768-1295

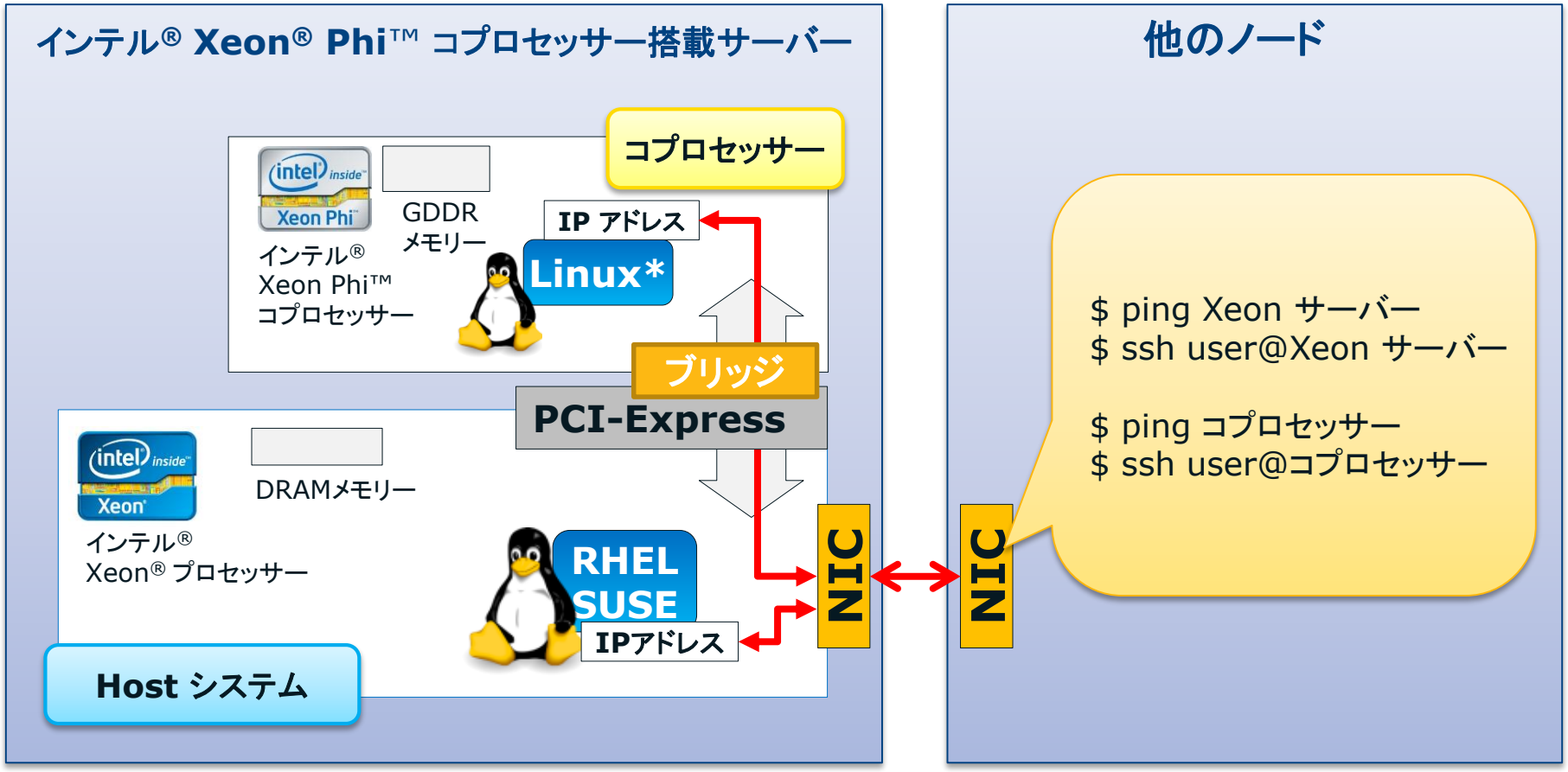
インテル® Xeon Phi™ コプロセッサー プラットフォーム概要



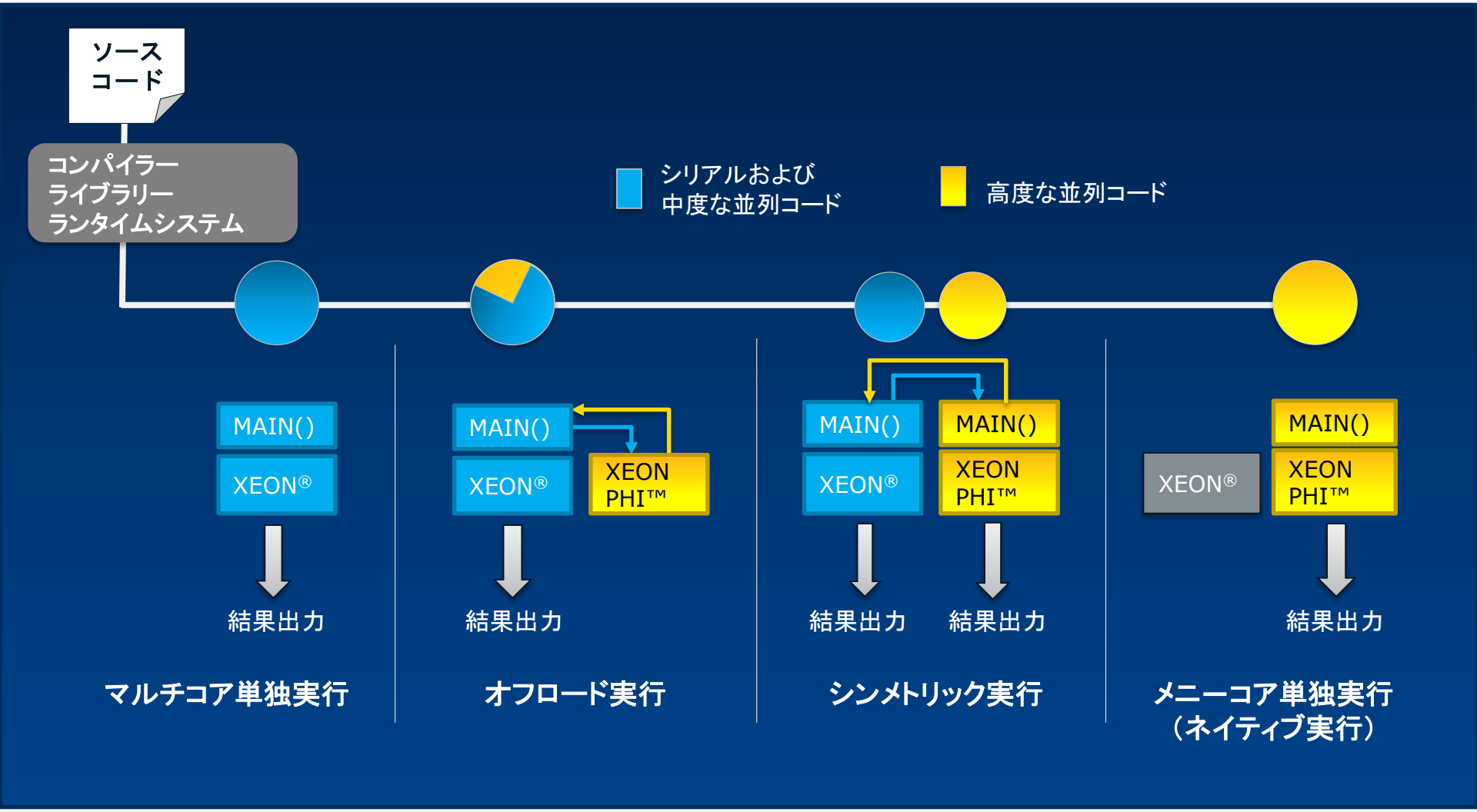
インテル® Xeon Phi™ コプロセッサーを利用したシステム構成例



インテル® Xeon Phi™ コプロセッサーを利用したシステム構成例（複数ノードの構成例）



実行モデルの概要



ヘテロジニアス・プログラミングについて

～はじめに～

- **CPU とコプロセッサーの命令セットは似ていますが同一ではありません。**
- **CPU とコプロセッサーはメインメモリーを共有しません。**

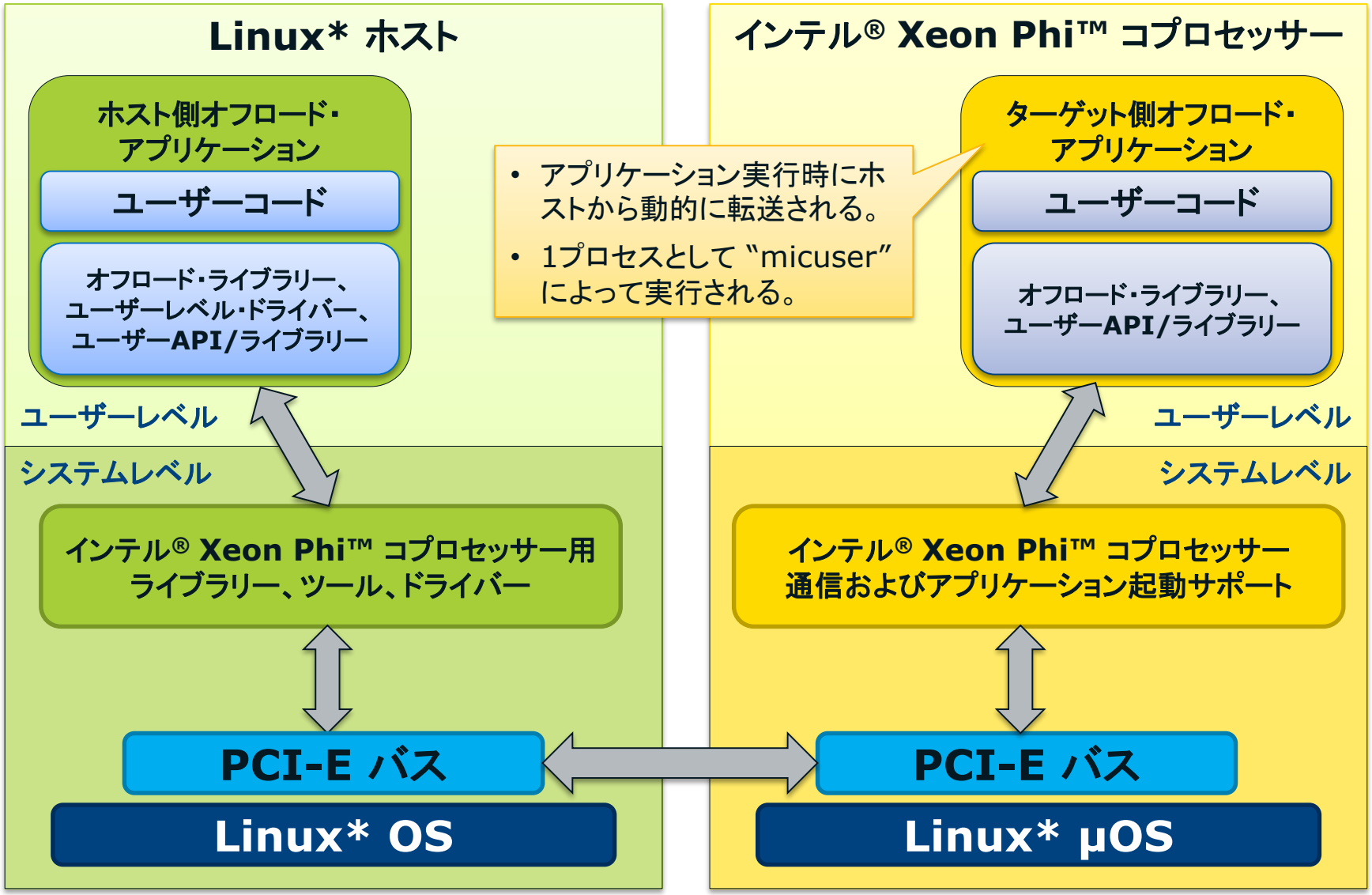
「オフロード実行」と「ネイティブ実行」

- **オフロード実行**：プログラムの特定の処理をインテル® 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**：インテル® Xeon Phi™ コプロセッサー上でのみプログラムを実行させるモデル

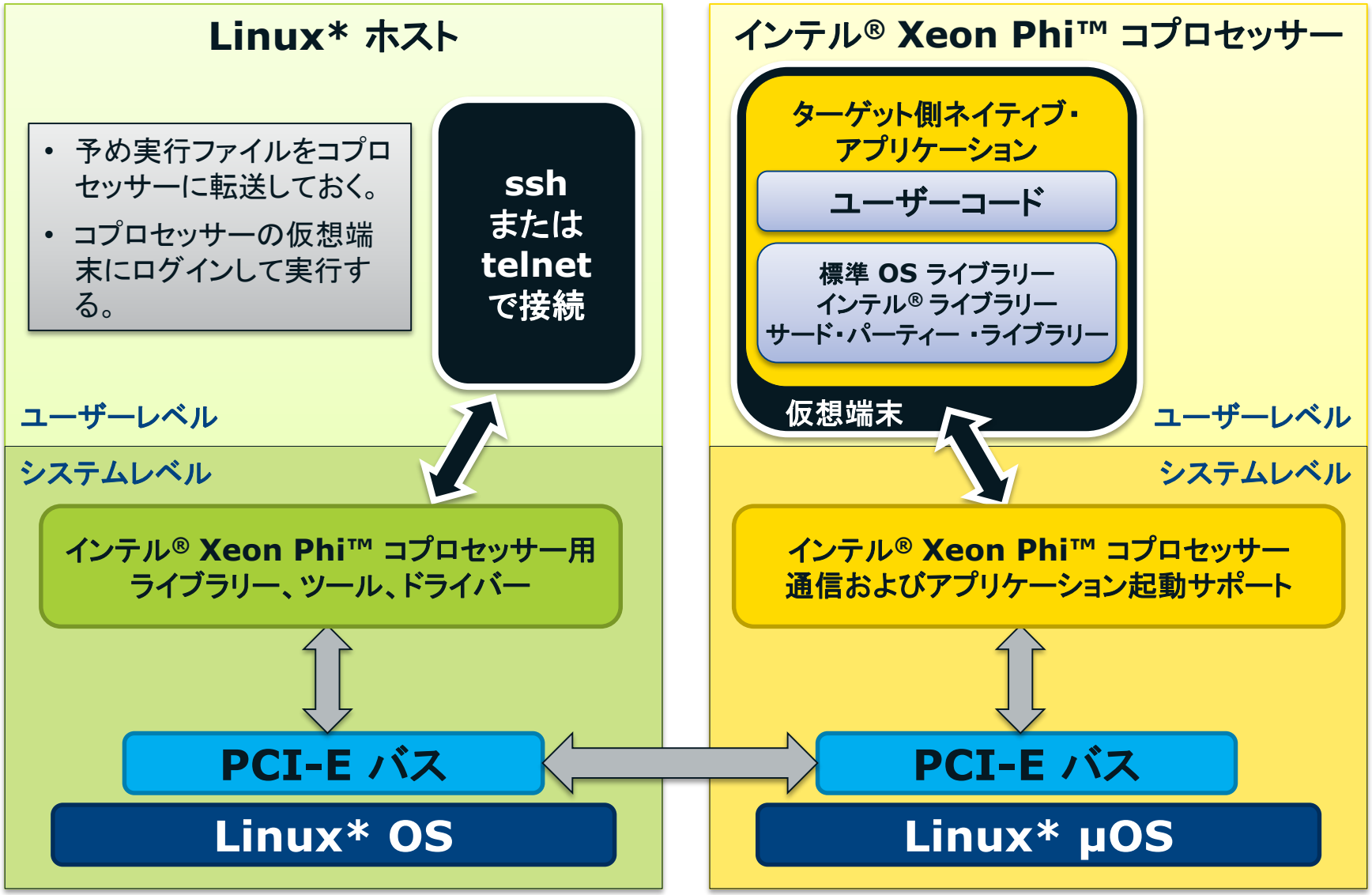
インテル® コンパイラーは、

- **CPU とコプロセッサー両方のバイナリーを生成することができます。**
- **オフロード実行用プログラムを記述するための言語拡張が提供されます。**

ソフトウェア構成概要（オフロード実行モデル）



ソフトウェア構成概要（ネイティブ実行モデル）



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

非共有型オフロード宣言子

	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 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 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 MIC 用にファイル
		全体またはコードブロックを指定

データ転送	#pragma offload_transfer target(mic)	データ転送開始（非同期） データ転送開始と完了（同期）

	Fortran シンタックス	意味
オフロード宣言子	!dir$ omp offload <指示句> <ステートメント>	次の OpenMP parallel 構文 を MIC で実行
	!dir$ offload <指示句> <ステートメント>	次のステートメントを MIC で実行
変数／関数用 キーワード	!dir$ attributes offload:<mic> :: <ret-name> もしくは <var1,var2,...>	CPU および MIC 用に関数／ 変数コンパイル

非共有型オフロード宣言子(続き)

節	シンタックス	意味
ターゲット指定	target(name[:#])	実行する 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 MIC へコピー
出力	out (var-list modifiersopt)	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 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 MIC に存在
非同期オフロード	signal(signal-slot)	非同期オフロードの開始
非同期オフロード	wait(signal-slot)	非同期オフロードの完了待ち

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

非共有型オフロード修飾子の例

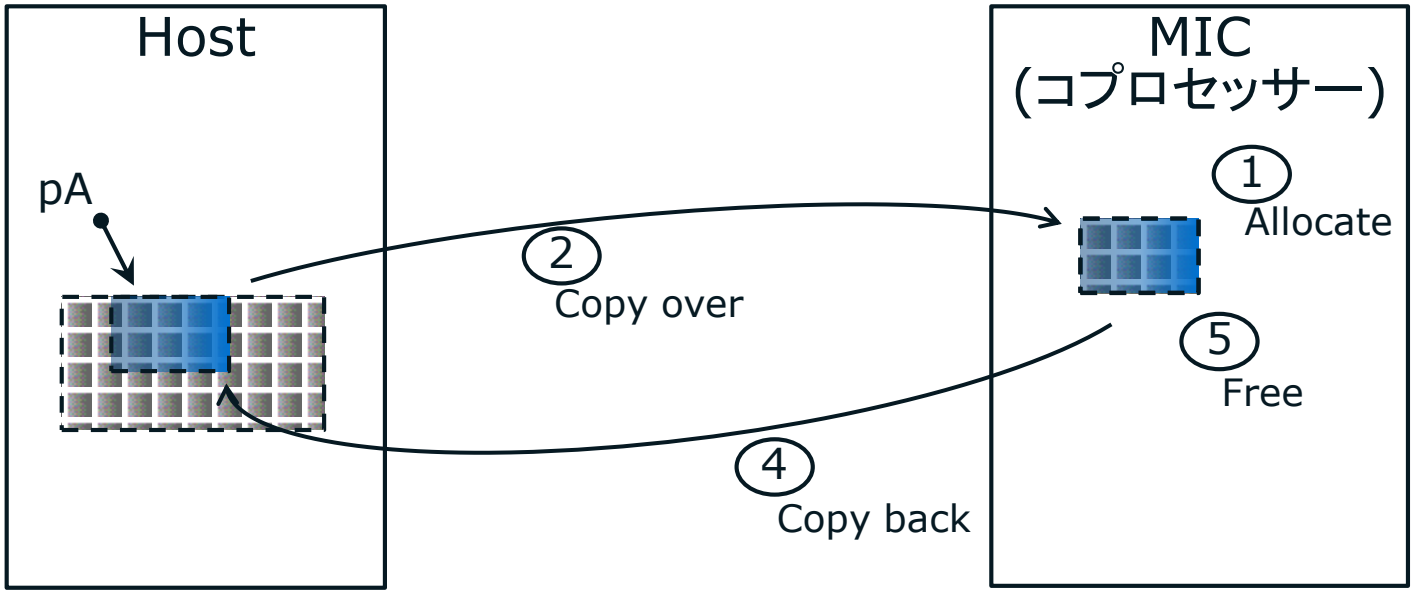
```
float reduction(float *data, int numberOf)
{
   float ret = 0.f;
   #pragma offload target(mic) in(data:length(numberOf))
   {
      #pragma omp parallel for reduction(+:ret)
      for (int i=0; i < numberOf; ++i)
         ret += data[i];
   }
   return ret;
}
```

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タイプをすでに知っている。

インテル® コンパイラーによるコンパイルと実行例:

```
$ source /opt/intel/bin/compilervars.sh  intel64          ← 環境設定
$ icc sample_offload.c –O2 –openmp –o sample_offload      ← コンパイル
$ ./sample_offload                                        ← 実行
```

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 MIC(コプロセッサー) に領域が確保される
  ② in 変数は Host から MIC に転送される。
- オフロードの終了時:
  ④ out 変数は MIC から Host に転送される。
  ⑤ 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)

```
#pragma offload inout(pA:length(n))
{...}
```
3

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” 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。

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:

```
# source /opt/intel/bin/compilervars.sh   intel64
# icc  -mmic  -openmp   pi.c -o pi.out.mic

# scp   pi.out.mic   root@mic0:
# scp   /opt/intel/composerxe/lib/mic/libiomp5.so   root@mic0:
```

コプロセッサーへのログインとプログラムの実行例:

```
# ssh   root@mic0
# export   LD_LIBRARY_PATH=./
# ./pi.out.mic
```

※青文字は、NFS (ネットワークファイルシステム) などでファイルを共有していない場合に、実行する必要があるコマンドです。

ネイティブ実行モデルのコンパイル／実行例 (micnativeloadex ツールの利用)

```
$ source /opt/intel/bin/compilervars.sh  intel64
$
$ icc -mmic -openmp omp_app.cpp
$
$ export SINK_LD_LIBRARY_PATH=/opt/intel/composerxe/lib/mic
$
$ /opt/intel/mic/bin/micnativeloadex a.out
```

ノート: SINK_LD_LIBRARY_PATH 環境変数に 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。
また “micnativeloadex a.out -l” で依存ライブラリーをチェック可能。

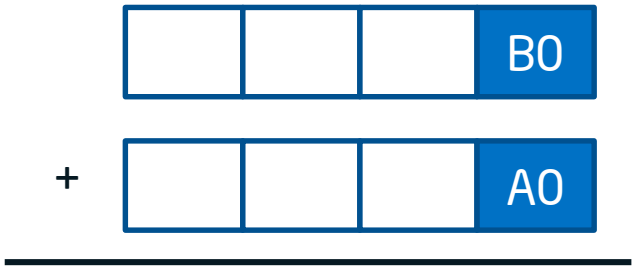
“micnativeloadex”ツールでネイティブ実行作業を簡略化できる。

性能を発揮させるために

```
float *restrict A, *B, *C;
for(i=0;i<n;i++){
C[i] = A[i] + B[i];
}
```

スカラーコードでは 1 要素ごとに処理される

```
addss %xmm1, %xmm2
```

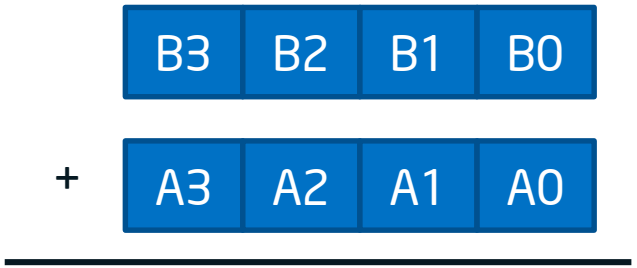


性能を発揮させるために (つづき)

```
float *restrict A, *B, *C;

for(i=0;i<n;i++){

C[i] = A[i] + B[i];

}
```

- [SSE] float 4 要素を一度に演算
  addps %xmm1, %xmm2
- [AVX] float 8 要素を一度に演算
  vaddps %ymm1, %ymm2, %ymm3
- [MIC] float 16 要素を一度に演算
  vaddps %zmm1, %zmm2, %zmm3

SIMD 命令
(Single Instruction Multiple Data)
は 1 命令で複数要素の演算を行う



スカラー演算とSIMD演算の比較

```
for (i=0; i<40000; i++)
  c[i] = a[i] + b[i];
```

スカラー演算
コード例

```
.B1.2::
  movss  xmm0, A[rcx+r9*4]
  addss  xmm0, B[rdx+r9*4]
  movss  C[r8+r9*4], xmm0
  inc    r9
  cmp    r9, 40000
  jl     .B1.2
```

1要素ずつ
40,000回の処理

SIMD 演算
コード例（AVX）

```
.B1.2::
  vmovups  ymm0, A[rcx+r9*4]
  vaddps   ymm1,ymm0,B[rdx+r9*4]
  vmovups  C[r8+r9*4], ymm1
  add      r9, 8
  cmp      r9, 40000
  jb       .B1.2
```

8要素ずつ
5,000回の処理

addss : Scalar Single-FP Add

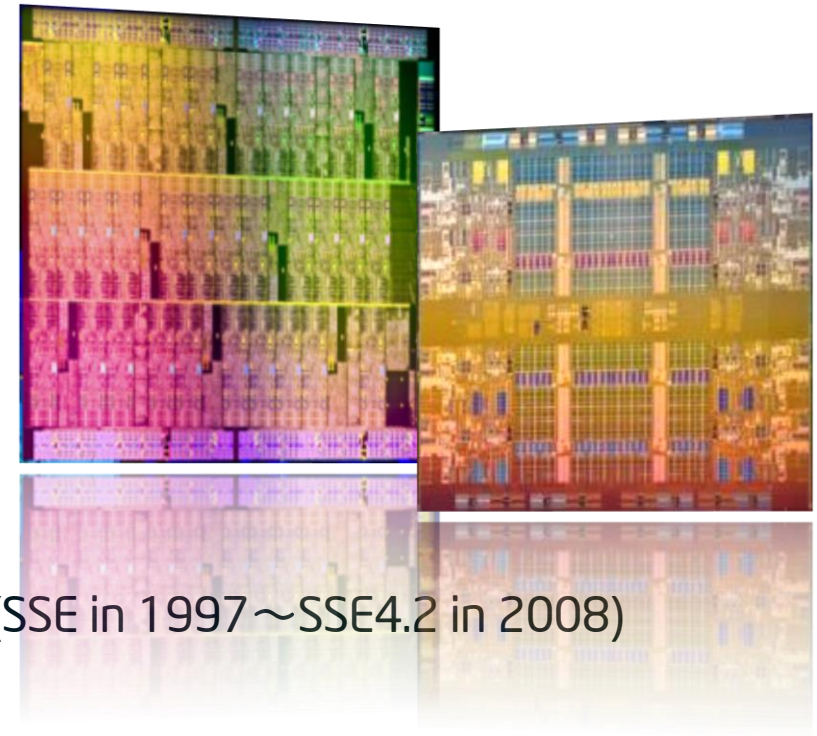
single precision FP data
scalar execution mode

vaddps : Packed Single-FP Add

single precision FP data
packed execution mode

ループを展開（アンロール）し、パックドSIMD命令を活用して
コードの処理効率を高める最適化技術をベクトル化と呼んでいる

SIMD の歴史



インテル® Pentirm® プロセッサー (1993)

MMX® (1997)

インテル® ストリーミング SMID Extensions (SSE in 1997～SSE4.2 in 2008)

インテル® AVX (AVX in 2011、AVX2 in 2013)

インテル® MIC アーキテクチャー (インテル® Xeon Phi™ コプロセッサー in 2012)

コアの演算性能を最大限に発揮させるためにはベクトル化が必須

ベクトル化オプション

オフロード宣言子が記述されている場合、
オフロードされる部分は自動的に -mmic と同じ
最適化がされる (-mmic を指定する必要がない)

インテル® Xeon® プロセッサー E5-2600 製品ファミリー +
インテル® Xeon Phi™ コプロセッサー向けのオフロードモデルのオプション例
$ icc hoge.c -O3 -xAVX

インテル® Xeon Phi™ コプロセッサー専用のオプション例
$ icc hoge.c -O3 -mmic

商標および最適化に関する注意事項について

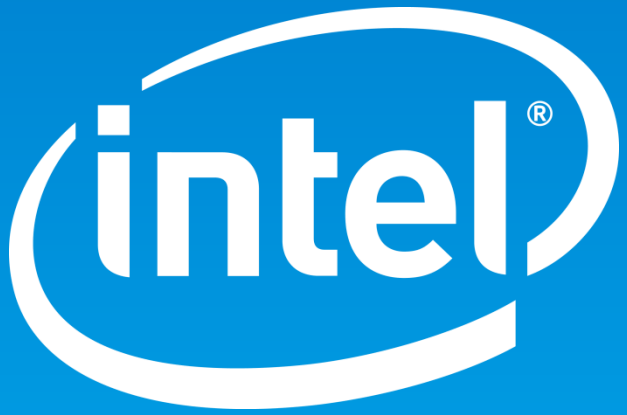
本資料の情報は、現状のまま提供され、本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスを許諾するものではありません。製品に付属の売買契約書『Intel ‘s Terms and Conditions of Sale』に規定されている場合を除き、インテルはいかなる責任を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証（特定目的への適合性、商品適確性、あらゆる特許権、著作権、その他知的財産権の非侵害性への保証を含む）に関してもいかなる責任も負いません。

性能に関するテストや評価は、特定のコンピューター・システム、コンポーネント、またはそれらを組み合わせて行ったものであり、このテストによるインテル製品の性能の概算の値を表しているものです。システム・ハードウェアの設計、ソフトウェア、構成などの違いにより、実際の性能は掲載された性能テストや評価とは異なる場合があります。システムやコンポーネントの購入を検討される場合は、ほかの情報も参考にして、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。インテル製品の性能評価についてさらに詳しい情報をお知りになりたい場合は、http://www.intel.com/performance/ (英語) を参照してください。

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


intel