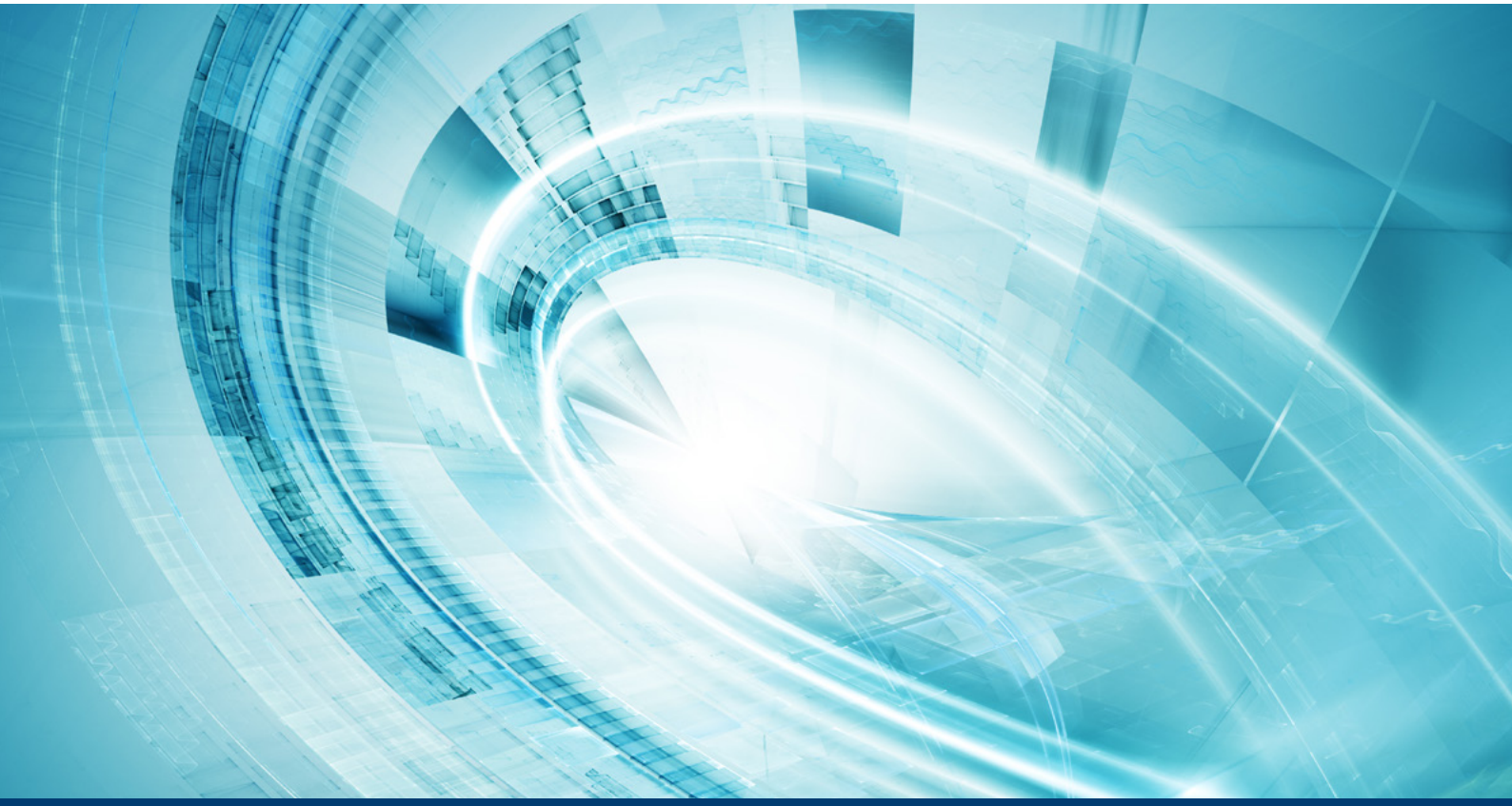


インテル® MPI ライブラリーを利用した PYTHON* の並列プログラミング

パフォーマンスを向上するためのガイドラインとツール

Artem Ryabov インテル コーポレーション ソフトウェア開発エンジニア
Alexey Malhanov インテル コーポレーション ソフトウェア開発エンジニア

ハイパフォーマンス・コンピューティング (HPC) はこれまで、Fortran や C のようなコンパイラー型言語を利用してソフトウェア開発が行われてきました。ところが最近では、Python* のようなインタープリター型高水準言語でも、モジュールを利用して、モデリング、シミュレーション、設計などの科学 / エンジニアリング・アプリケーションを簡単に開発できるようになりました。では、Python* で並列アプリケーションを作成することは可能なのでしょうか。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

この記事では、分散メモリーシステムの HPC で最もよく使用されているプログラミング・モデルである、メッセージ・パッシング・インターフェイス (MPI) と Python* を組み合わせた並列プログラミングの方法について説明します。スクリプト言語と **MPI** 間のインターフェイスでよく発生する問題は、パフォーマンスに関するものです。

まず、ネイティブ MPI 実装 (**インテル® MPI ライブラリー**) の Python* ラッパーとして **mpi4py** を使用する、並列 Python* アプリケーションの開発ガイドラインを説明します。mpi4py は、インテル® Distribution for Python* (**2017** 以降) に含まれています。さらに、mpi4py を使用して記述した MPI アプリケーションのパフォーマンス向上に役立つ手法と、**インテル® Trace Analyzer & Collector** のようなインテルのツールを使用して並列 Python* アプリケーションを解析する方法について説明します。

MPI for Python*

MPI for Python* (mpi4py) は、MPI にオブジェクト指向のアプローチを提供します。このインターフェイスは、標準 C++ MPI バインドの MPI 構文とセマンティクスを Python* に変換するように設計されています。

一般に、mpi4py は、Python* オブジェクトのシリアル化機能を使用して、任意のビルトインまたはユーザー定義の Python* オブジェクトと通信します。特に大きなバッファーを転送する場合、シリアル化によりメモリーやプロセッサーの利用に著しいオーバーヘッドが発生します。幸い、mpi4py は、一部の型 (文字列や数値配列など) で提供される標準 Python* メカニズムを備えたオブジェクトの直接通信をサポートしています。この機能は、C 側で、関連データを含む連続するメモリーバッファーへのアクセス (アドレスと長さ) を提供します。この機能を、複雑なメモリーレイアウトを記述するユーザー定義の MPI データ型を構築する機能とともに利用すると、オーバーヘッドを気にすることなく多次元数値配列を含む多くのアルゴリズムを Python* で直接実装して、コンパイルした Fortran、C、C++ コードとほぼ同じ速度で実行することができます。

使用法

図 1 は、mpi4py で記述された簡単な “Hello World” の例です。

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
name = MPI.Get_processor_name()
if rank == 0:
  print "Hello world: rank %d of %d running on %s" % (rank, size, name)
  for i in xrange(1, size):
    rank, size, name = comm.recv(source=i, tag=1)
    print "Hello world: rank %d of %d running on %s" % (rank, size, name)
else:
  comm.send((rank, size, name), dest=0, tag=1)
```

1 サンプル・アプリケーション

この例で、**comm** はコミュニケーター・オブジェクトです。MPI 関数 **Get_size()**、**Get_rank()**、**send()**、**recv()** は、このコミュニケーター・オブジェクトのメソッドです。

図 1 の例は**図 2** のコマンドを使用して実行します。

```
$ source <intel_mpi>/bin/mpivars.sh

$ source <intel_python>/bin/pythonvars.sh

$ mpirun -ppn 1 -n 2 -hosts node01,node02 python helloworld.py
Hello world: rank 0 of 2 running on node01
Hello world: rank 1 of 2 running on node02
```

2 コマンド

mpivars.sh スクリプトは、インテル® MPI ライブラリーの環境を設定します。**pythonvars.sh** は、インテル® Distribution for Python* の環境変数を設定します。**mpirun** ユーティリティーは、MPI アプリケーションを起動します。

パフォーマンス

前述したように、mpi4py には 2 種類の通信があります。

- **シリアル化ベースのジェネリック Python* オブジェクトの通信。**mpi4py では、**send()**、**recv()**、**bcast()** のように、(コミュニケーター・オブジェクトの) 小文字のメソッドです。送られるオブジェクトが通信呼び出しのパラメーターとして渡され、受け取ったオブジェクトが戻り値になります。
- **バッファー型オブジェクトの通信。**Send()、Recv()、Bcast()、Scatter()、Gather() のように、メソッド名が大文字で始まります。一般に、これらの呼び出しのバッファー引数は、**[data, MPI.DOUBLE]** や **[data, count, MPI.DOUBLE]** のように、リスト / タプルを使用して明示的に指定する必要があります (前者はバイトサイズのデータと MPI データ型の範囲を使用してカウントを定義します)。

ジェネリック Python* オブジェクトの通信は、オブジェクトのシリアル化による追加のメモリー /CPU オーバーヘッドが発生し、パフォーマンスに影響することがあります。バッファー形式のオブジェクトの通信は、C に近い速度で行われるため、パフォーマンスへの影響はほとんどありません。

パフォーマンス向上のため、可能な場合は Python* インターフェイス・メソッドに大文字の MPI 通信を使用することを推奨します。

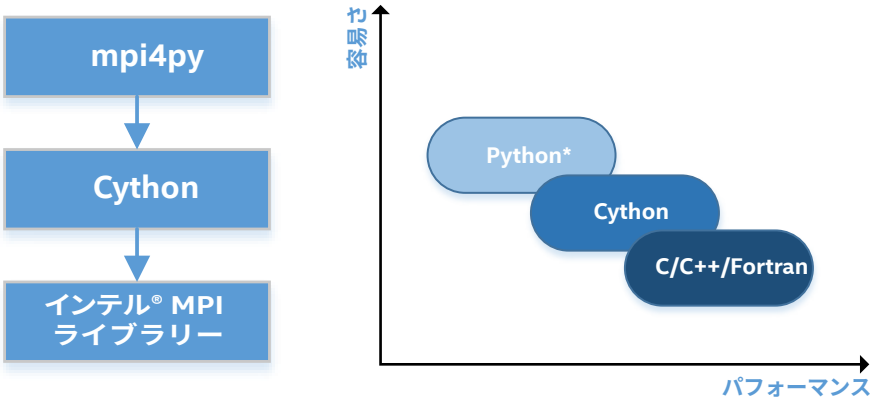
MPI for Python* とインテル® MPI ライブラリーのパフォーマンス向上のためのもう 1 つの推奨事項は、**I_MPI_SHM_LMT=shm** 環境変数を指定して共有メモリーの大規模メッセージ転送メカニズムをオンにすることです (詳細は、**インテル® MPI ライブラリーのドキュメント**を参照)。

mpi4py とネイティブ MPI 実装

mpi4py のオーバーヘッドをネイティブ MPI 実装と比較すると、興味深いことが分かります。

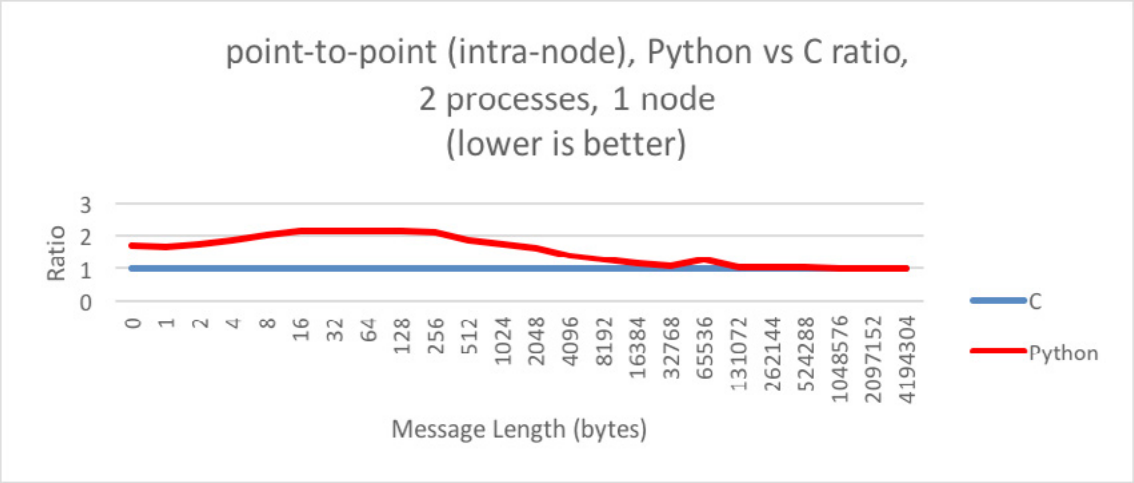
mpi4py は Cython 言語を使用して実装されています。Cython は Python* 言語に似ていますが、Cython は C 関数呼び出しと C 型の変数宣言およびクラス属性をサポートしているため、コンパイラーは Cython コードから非常に効率の良い C コードを生成することができます。このことから、Cython は外部 C ライブラリーと高速な C モジュールをラップして Python* コードの実行を高速化する理想的な言語と考えられています。Python* で MPI コードを記述すると、mpi4py は、その部分を Cython を使用した C の呼び出しに変換します。

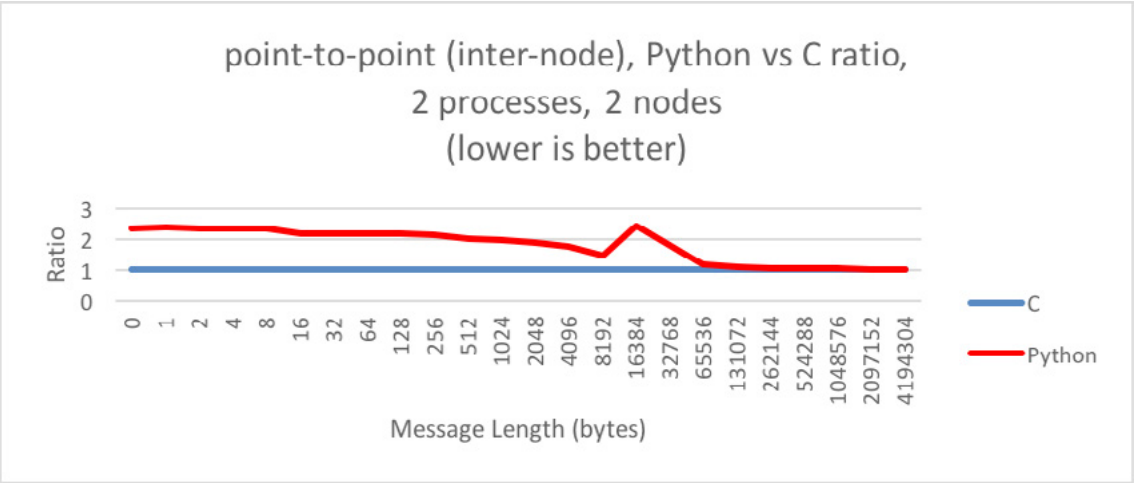
図で示すと、**図 3** のようになります。



3 mpi4py の仕組み

MPI ポイントツーポイント操作のパフォーマンスを測定した結果は、この仕組みを裏付けるものとなりました **(図 4)**。同様の結果は集合操作にも当てはまります。





ポイントツーポイントのパフォーマンス測定には、**osu_latency** ベンチマークを使用しました。

システム構成 : ハードウェア : インテル® Xeon® プロセッサー E5-2699 v4 (2.20GHz)、64GB RAM、インターコネクト : インテル® Omni-Path ホスト・ファブリック・インターフェイス。ソフトウェア : Red Hat* Enterprise Linux* 7.1、ベータ版 インテル® C++ コンパイラー 2017、ベータ版 インテル® MPI ライブラリー 2017、**OSU Micro-Benchmarks**。各実行で I_MPI_SHM_LMT=shm 環境変数を使用。

4 パフォーマンス・テストの結果

解析結果から、MPI for Python* のパフォーマンスはかなり優れていることがわかります。mpi4py のパフォーマンスは小さな / 中くらいのメッセージサイズではネイティブ MPI 実装よりも (最大 2.5 倍) 遅くなっていますが、大きなサイズではほぼ同じです。

アプリケーション

Python* の柔軟性と MPI の優れた通信効率を利用するアプリケーションの数は急激に増加しています。有名なプロジェクトの 1 つは、PETSc (Portable Extensible Toolkit for Scientific Computations) 向けの Python* バインド、**petsc4py** です。PETSc は、偏微分方程式によりモデル化された科学アプリケーションを**並列計算**で解くためのデータ構造とルーチンのセットで、MPI をサポートしています。

MPI for Python* は、ParaView マルチプラットフォーム・データ解析 / 視覚化アプリケーションのコンポーネントとして、ユーザーが並列でフィルターを実行する必要がある場合に利用されます。また、天体物理学問題を解くための統合科学環境である **yt プロジェクト**でも、MPI for Python* を利用しています。

インテル® Trace Analyzer & Collector を使用したプロファイル

MPI for Python* アプリケーションは、インテル® Trace Analyzer & Collector を使用してプロファイルすることができます **(図 5)**。

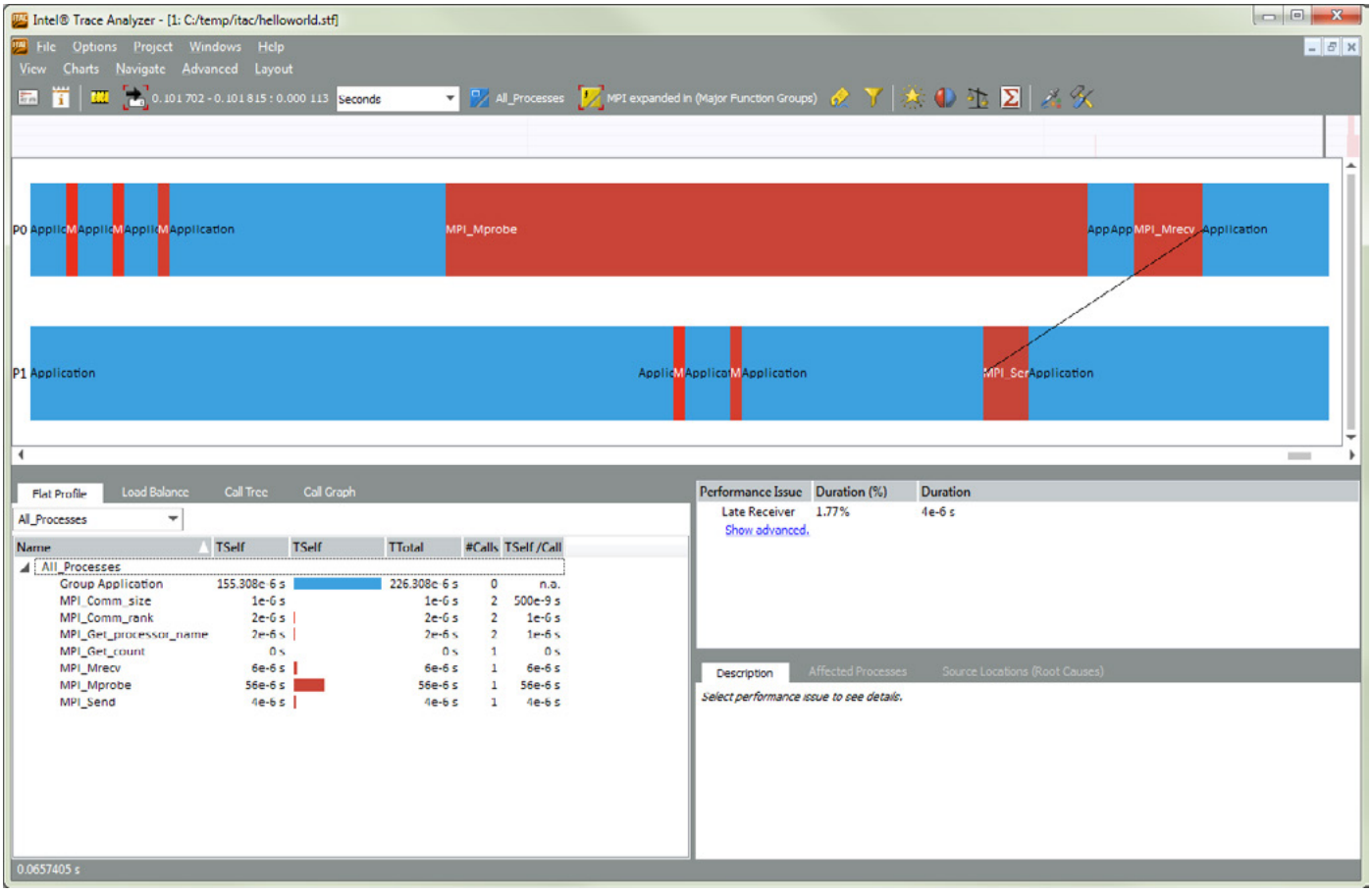
```
$ source <itac _ dir>/bin/itacvars.sh
$ export VT _ LOGFILE _ NAME=helloworld.stf
$ export VT _ LOGFILE _ FORMAT=SINGLESTF
$ export LD _ PRELOAD="<itac _ dir>/intel64/slib/libVT.so <impi _ dir>/intel64/lib/release _ mt/libmpi.so"

$ mpirun -ppn 1 -n 2 -hosts node01,node02 python helloworld.py
Hello world: rank 0 of 2 running on node01
Hello world: rank 1 of 2 running on node02
[0] Intel(R) Trace Collector INFO: Writing tracefile helloworld.stf in <dirname>
```

5 インテル® Trace Analyzer & Collector を使用したプロファイル

VT_LOGFILE_NAME および **VT_LOGFILE_FORMAT** 変数はオプションです。これらの変数は、主にトレースファイルを利用した操作に使用されます。図 5 のコマンドを実行すると、トレースファイル **helloworld.stf** および補助ファイル **helloworld.prot** が生成されます。トレースファイルの確認には、**traceanalyzer** ユーティリティーを使用します **(図 6)**。

```
$ traceanalyzer helloworld.stf &
```

6 traceanalyzer ユーティリティー



7 インテル® Trace Analyzer & Collector を使用した MPI for Python* アプリケーションのプロファイル

インテル® Trace Analyzer & Collector は低レベルの MPI 呼び出しをすべて表示するため、その一部はオリジナルの MPI for Python* コードに存在しないことに注意してください **(図 7)**。この例では、Python* **`recv()`** から **`MPI_Mprobe()/MPI_Mrecv()`** が生成されています。

関連情報

1. **インテル® MPI ライブラリー**
2. **インテル® Distribution for Python***
3. **インテル® Trace Analyzer & Collector**
4. **mpi4py と VTK (英語)**
5. **mpi4py の公式ページ (英語)**
6. **PETSc (英語)**
7. **yt プロジェクトの公式ページ (英語)**

インテル® ソフトウェア製品 技術ウェビナー

intel Software

1 時間で学ぶコードの現代化。

最新の技術ウェビナーを追加しました。
インテルのエキスパートが、より高速で、より優れた
HPC アプリケーションの作成方法を解説します。

2016 年春 / 夏の技術ウェビナーシリーズ

ハイブリッド・プログラムの 理解を深めよう >

インテル® TBB とヘテロジニアス・コンピューティング >

インテル® Fortran コンパイラーを利用して効率良く並列処理を最適化する (英語) >

Spark* クラスターで高速なデータ・アプリケーションをビルドする (英語) >

OpenMP*: メモリー共有型並列プログラミングの先 (英語) >

SDLT を利用して効率良くベクトル化する (英語) >

ベクトル化するか、それともせずに滅びるか : 最新のインテル® AVX SIMD 向けにチューニング (英語) >

インテル ® MKL を利用して Python* のパフォーマンスを向上する (英語) >

競合、デッドロック、メモリーの問題を解決する (英語) >

NUMA のワークロードへの影響 : インテル® VTune™ Amplifier XE (英語) >

インテル® VTune™ Amplifier XE で Python* アプリケーションのパフォーマンスを解析する (英語) >

インテル® IPP を利用してコードを高速化する (英語) >

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。
© 2016 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴ、VTune 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。
* 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。