第 2 世代インテル® Xeon Phi™ プロセッサー上のインテル® Distribution for Python*: デフォルト設定のパフォーマンス

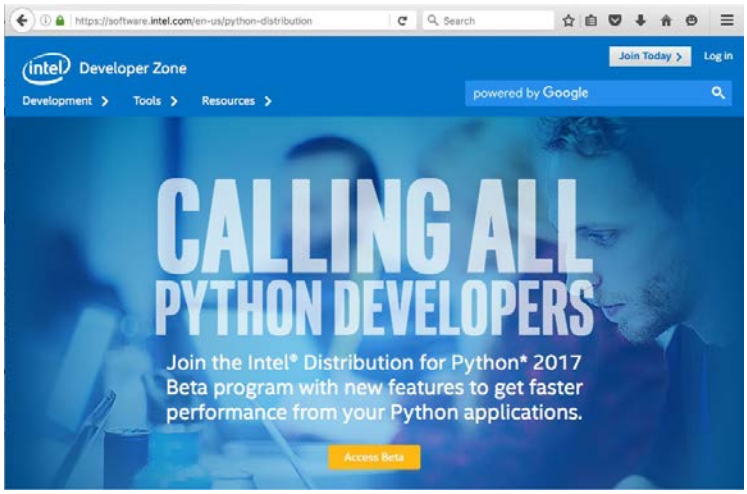
Tony Yoo、Ryo Asai、Andrey Vladimirov
Colfax International

2016 年 6 月 20 日

抄録

この文献は、第 2 世代インテル® Xeon Phi ™ プロセッサー (開発コード名 Knights Landing) 上でインテル® Distribution for Python* 2017 Beta を使用した計算アプリケーションの価値とパフォーマンスに関する報告です。SciPy* および NumPy* ライブラリーの LU 分解、コレスキー分解、特異値分解、および倍精度の一般的な行列-行列乗算ルーチンのベンチマークを使用して、高帯域幅メモリー (HBM) 向けのチューニング手法を示します。

目次



1. コンピューティングにおける Python* のケーススタディー

Python*[1] は、計算アプリケーションで人気のあるスクリプト言語です。科学計算向けの基本ツール、NumPy*[2] および SciPy*[3] ライブラリーにより、Python* アプリケーションは行列と線形代数方程式に対する操作を、簡潔で便利な基本線形代数 (BLAS) および線形代数パッケージ (LAPACK) 関数を使用して記述することができます。

BLAS と LAPACK で優れたパフォーマンスを発揮する第 2 世代インテル® Xeon Phi™ プロセッサー (開発コード名 Knights Landing または KNL) は、Python*/ NumPy*/ SciPy* アプリケーション向けのコンピューティング・プラットフォームとして最適です。しかし、標準の Python* ディストリビューションである CPython は、インテル® Xeon Phi™ プロセッサーのベースであるインテル® メニー・インテグレーテッド・コア (インテル® MIC) アーキテクチャーの利点をまだ活用することができません。

この問題に対応するため、インテル® Distribution for Python*[4] は、インテル® マス・カーネル・ライブラリー (インテル® MKL)[5] を利用して線形代数を高速化します。さらに、このディストリビューションは、インテル® スレッディング・ビルディング・ブロック (インテル® TBB)、インテル®データ・アナリティクス・アクセラレーション・ライブラリー (インテル® DAAL)、インテル® MPI ライブラリーとのインターフェイスを提供します。

この文献では、インテル® Xeon Phi™ プロセッサー上での BLAS および LAPACK の基本サブセット (LU 分解、コレスキー分解、特異値分解 (SVD)、および一般的な行列-行列乗算) の利便性とパフォーマンスについて報告します。CPython とインテル® Distribution for Python* のベンチマークを比較します。

2. ベンチマーク

ルーチンのベンチマーク・コードの主要部分をリスト 1 に示します。CPython では SciPy* および NumPy* ライブラリーを使用し、インテル® Distribution for Python* では SciPy* のみ使用しました。ループでは時間測定を 10 回繰り返しました。それぞれのベンチマークの前には、ウォームアップ計算があります。

インテル® Distribution for Python* 2017 Beta に含まれる Python* 2.7 と CentOS* 7.2 に含まれる標準ビルドの Python* 2.7.5 を使用しました。64 コア@ 1.30GHz、96GB DDR4、16GB MCDRAM (HBM: 高帯域幅メモリー) 搭載のインテル® Xeon Phi™ プロセッサー 7210 ベースのシステムでテストしました。HBM は、個別の NUMA ノードでプログラマーにはアドレス指定可能なメモリーとして見える、フラットモードで使用しました (詳細はこちらの文献を参照してください)。

```
# LU 分解ベンチマーク
import time
from scipy.linalg import \
         lu_factor as lu_factor
...
start = time.time()
LU,piv = lu_factor(A, overwrite_a=True, \
         check_finite=False)
stop = time.time()
```

```
# コレスキー分解ベンチマーク
from scipy.linalg import \
          cholesky as cholesky
...
start = time.time()
L = cholesky(A, overwrite_a=True, \
         check_finite=False)
stop = time.time()
```

```
# SVD ベンチマーク
from scipy.linalg import \
         svd as svd
...
start = time.time()
U, s, V = svd(A)
stop = time.time()
```

```
# DGEMM ベンチマーク
from scipy.linalg.blas import \
         dgemm as dgemm
...
start = time.time()
C=dgemm(alpha=1.0, a=A, b=B, c=C, \
         overwrite_c=1, trans_b=1)
stop = time.time()
```

リスト 1: ベンチマーク・コードの抜粋

[1] www.python.org (英語)
[2] numpy.org (英語)
[3] www.scipy.org (英語)
[4] www.isus.jp/python-distribution/
[5] www.isus.jp/intel-mkl/

ここでは、環境変数を使用してインテル® MKL の実行環境をチューニングしていません。このテストにより、インテル® MKL がデフォルトで選択したスレッド数とスレッド・アフィニティーのほうが、ほかの設定よりも優れたパフォーマンスをもたらすことが証明されました。

しかし、高帯域幅メモリーを活用するため、次に示すように `numactl` ツールを使用してベンチマークを実行することで、アプリケーション全体を HBM に配置しました。

```
user@knl% numactl -m 1 benchmark-script.py
```

リスト 2: `numactl` でアプリケーションを HBM にバインド

3. 結果と考察

行列サイズ N = 5000 のベンチマークを図 1 に示します。その他のベンチマークは、付録 A にあります。

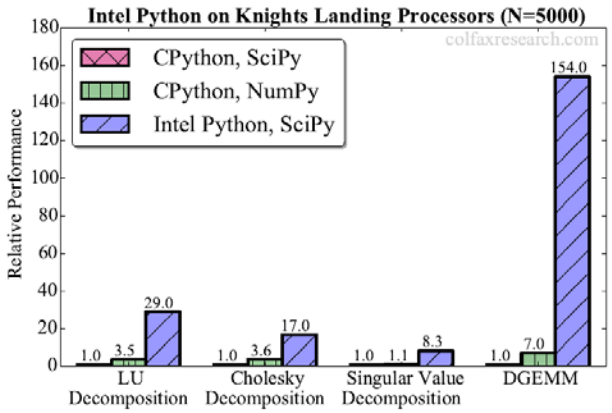


図 1: インテル® Distribution for Python* によるパフォーマンス・ゲイン

インテル® Distribution for Python* は、コードを変更せずに、デフォルトの設定で大幅なパフォーマンス・ゲインをもたらしました。

インテル® MKL ライブラリーによって報告された計算時間 (環境変数 `MKL_VERBOSE=1` を設定して取得) は、Python* 呼び出しの時間よりも短く、その差は DGEMM では 0.2% ~ 25% (大きな行列のほうが小さい)、LU では 23% ~ 60%、SVD では 4% ~ 7%、コレスキー分解では 65% を超えました。このテストでは、デフォルト設定のパフォーマンスを評価することが目的のため、このオーバーヘッドの原因は調査していません。BLAS および LAPACK に依存するアプリケーションでは、C や Fortran などのコンパイル言語の実装により利点が得られる可能性があります。

また、CPython では、テストしたすべてのルーチンで NumPy* のほうが SciPy* よりも高速でした (DGEMM では最大 7 倍)。しかし、このパフォーマンスの差は、インテル® Distribution for Python* により得られるスピードアップ (DGEMM では最大 150 倍) と比較するとわずかです。DGEMM では、N = 5000 の場合、倍精度で 1.85 TFLOP/秒 (付録 A を参照) を達成しました。これは、テストに使用したプロセッサーの理論的なピーク・パフォーマンスの 70% にあたります。このことから、インテル® アーキテクチャーのパフォーマンスを最大限に引き出すためには、インテル® MKL の使用が重要であることは明らかです。

インテル® Distribution for Python* によるパフォーマンスの最適化は、インテル® Xeon Phi™ プロセッサーに限られたことではありません。インテルの報告では、汎用インテル® Xeon® プロセッサー[6] でも同様のハイパフォーマンスが得られることを示しています。これは、その性質または実装が原因で高度に最適化されていないアプリケーションにとって朗報です。インテル® Xeon® プロセッサーの高速なクロック速度で実行するコアは、インテル® Xeon Phi™ プロセッサーの高度な並列処理向けに設計されたコアよりもシリアル・ワークロードに適しています。

インテル® MKL とインテル® Distribution for Python* は無料で利用することができます。[7]このことからも、インテル® Xeon Phi™ プロセッサー上で実行するパフォーマンスが重要な計算アプリケーションには、CPython よりもインテルのソフトウェア・スタックを推奨します。

この文献とダウンロード可能なベンチマーク・コードは、colfaxresearch.com/isc16-intel-python (英語) にあります。

[6] インテル® Distribution for Python* のページを参照してください。
[7] www.isus.jp/products/psxe/free_mkl/

A. その他のベンチマーク

各ベンチマーク関数のさまざまな問題サイズでのパフォーマンスを以下に示します。DGEMM では、相対パフォーマンスに加えて、絶対パフォーマンスも示しています。セクション 3 にある、この関数のオーバーヘッドに関するコメントを参照してください。

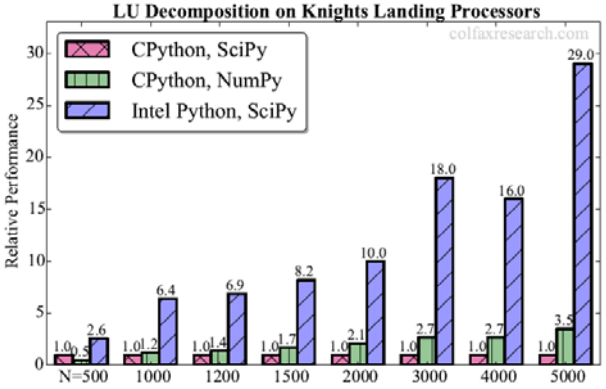


図 2.

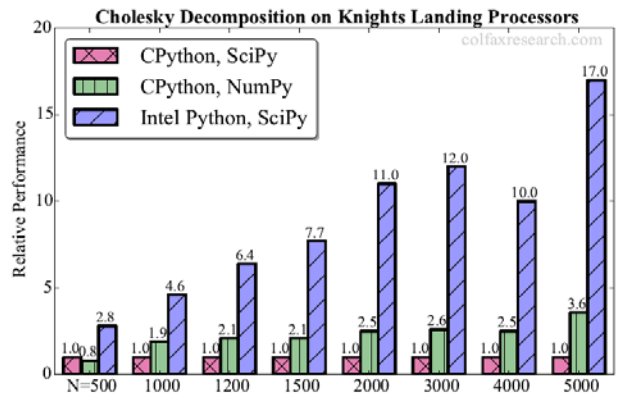


図 3.

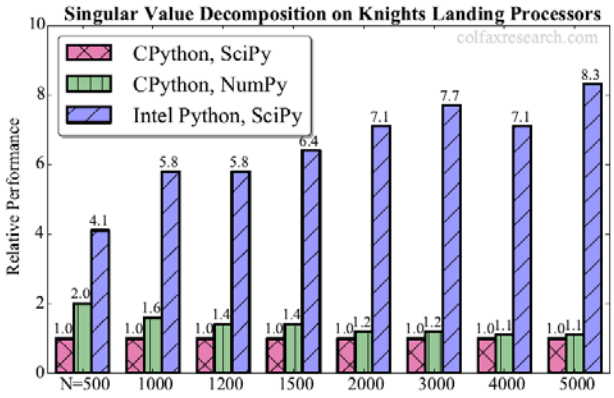


図 4.

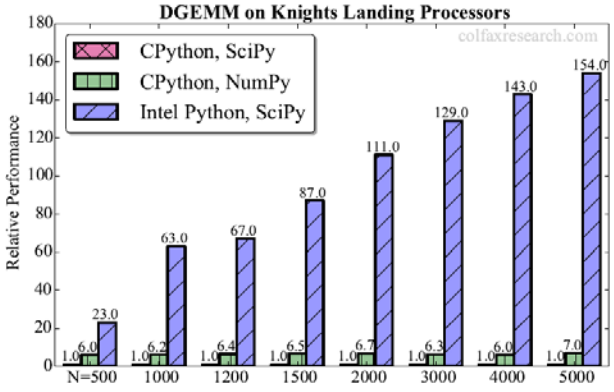


図 5.

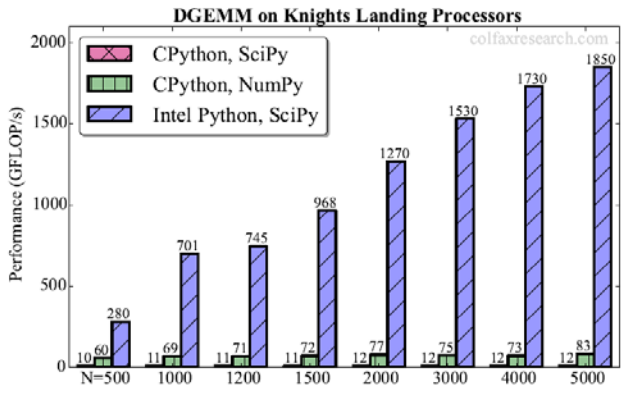


図 6.