オープンソース Python*、R、Julia ベースの HPC アプリケーションの高速化

Vipin Kumar E K
インテル コーポレーション
ソフトウェア & サービスグループ テクニカル・コンサルティング・エンジニア

インテル® コンパイラーとインテル® マス・カーネル・ライブラリー（インテル® MKL）によるパフォーマンスの向上

HPC コミュニティーでは、Python*、R、新しい Julia などのオープンソース言語に対する関心が高まっています。この記事では、GNU* コンパイラーと NumPy/SciPy、R、Julia 言語に同梱されているデフォルトの算術ライブラリーの代わりに、インテル® コンパイラーとインテル® マス・カーネル・ライブラリー（インテル® MKL）を利用してこれらの言語をビルドしインストールする方法を説明します。Linux* プラットフォーム上でこれらの言語をインテル® コンパイラーからビルドすることで、ベクトル化、OpenMP*、その他のコンパイラー機能を利用できるようになります。さらに、インテル® プラットフォーム用に高度に最適化されたインテル® MKL を介して、アプリケーションのパフォーマンスは大幅に向上するでしょう。

これらの言語には、インテル® コンパイラーやインテル® MKL が持つ最適化機能が含まれていません。その 1 つの機能はベクトル化です。ベクトル化により、インテル® アドバンスト・ベクトル・エクステンション（インテル® AVX1/AVX2）、インテル® Xeon Phi™ コプロセッサーの 512 ビット幅 SIMD、リリース予定のインテル® AVX-512 レジスターと命令のような、最新の SIMD ベクトルユニットを活用できます。インテル® ソフトウェア・ツールを導入すると、プロシージャー間の最適化、キャッシュとレジスターの効率良い使用、並列化などのインテル® アーキテクチャーの機能によって、プロセッサーのコアを最大限に利用できます。

インテル® Xeon Phi™ コプロセッサーがシステムに搭載されている場合、インテル® MKL の自動オフロード機能を有効にしてコプロセッサーを活用することで、パフォーマンスをさらに向上できます。GEMM、SYMM、TRSM、TRMM、LU、QR、コレスキー分解など、インテル® MKL の一部の BLAS および LAPACK 関数は、ホスト・プロセッサーとインテル® Xeon Phi™ コプロセッサー間で計算を自動的に分割できます。この機能を有効にするには、MKL_MIC_ENABLE=1 環境変数を設定します。問題サイズが大きくなるほど効果があります。

これらの言語のビルドを開始する前に、http://software.intel.com（英語）から最新のインテル® Parallel Studio XE またはインテル® Cluster Studio XE をダウンロードしてください。

インテル® ソフトウェア開発ツールのインストール・フォルダーから `compilervars.sh` を実行して、インテル® C/C++ コンパイラー、インテル® Fortran コンパイラー、インテル® MKL、インクルード・ファイル用の PATH、LD_LIBRARY_PATH、その他の環境変数をセットアップします。Linux* プラットフォームでは、インテル® コンパイラーおよびインテル® MKL のデフォルトのインストール先は /opt/intel/composer_xe_2013_sp1 です（インストールするバージョンによって異なります）。

64 ビットのインテル® アーキテクチャー（IA）向けに環境をセットアップするには、次のコマンドを実行します。

```
$source /opt/intel/composer_xe_2013_sp1/compilervars.sh intel64
```

32 ビットのインテル® アーキテクチャー（IA）向けの環境には、次のコマンドを実行します。

```
$source /opt/intel/composer_xe_2013_sp1/compilervars.sh ia32
```

インテル® MKL とインテル® コンパイラーを使用した Python* のインストール

Python* は、移植が容易で迅速に開発できる、オープンソースのインタープリター型オブジェクト指向言語です。Python* は記述が簡単であり、NumPy や SciPy のようなライブラリーをサポートしていることから非常にポピュラーで、科学技術計算コミュニティーでは、C/C++ および Fortran に続き最も広く使用されている言語の 1 つです。Python* が優れている点は、ほかの言語で記述されたソフトウェアへ容易に統合し、既存のプログラムを実行する制御言語として利用できることです。Python* は、異なるシステムを組み合わせる言語として動作します。

インテル® MKL とインテル® コンパイラーの最適化を使用した NumPy/SciPy のインストール

NumPy/SciPy は、科学技術計算や HPC（ハイパフォーマンス・コンピューティング）向けのオブジェクトとルーチンを提供する基本的なライブラリーです。インテル® MKL と統合することで、これらのライブラリーの BLAS および LAPACK のパフォーマンスを向上できます。インテル® Core™ i5 マシン（4 スレッド）でのテストでは、インテル® MKL で行列乗算を行った場合、オープンソースの ATLAS ライブラリー（4 スレッド）に比べてパフォーマンスが約 2 倍も向上しました。

NumPy は、可能であれば、ベクトルと行列の演算を BLAS および LAPACK 関数に自動的にマップします。インテル® MKL はこれらの標準インターフェイスに対応しているため、NumPy スクリプトを変更するだけでインテル® MKL の最適化機能の恩恵が得られます。

NumPy (http://numpy.scipy.org（英語）) は、Python* による科学技術計算に必須のパッケージであり、次のものが含まれます。

- 強力な N 次元の配列オブジェクト
- 高度な（ブロードキャスト）関数
- C/C++ と Fortran コードを統合するツール
- 優れた線形代数、フーリエ変換、乱数機能

科学的な用途に加えて、NumPy は汎用データの効率良い多次元コンテナーとして利用することもできます。

SciPy (http://www.scipy.org（英語）) には、統計、最適化、積分、線形代数、フーリエ変換、信号 / 画像処理、ODE ソルバー、その他のモジュールが含まれています。SciPy ライブラリーは、便利で高速な N 次元の配列操作を提供する NumPy に依存します。また、NumPy 配列で動作するように構築されており、数値積分用のルーチンや Python* ユーザー向けの最適化など、多数の使いやすく効率良い数値計算ルーチンを提供します。

NumPy および SciPy のソースコードは、http://www.scipy.org/Download（英語）からダウンロードできます。

最新バージョンの NumPy と SciPy の tar ファイルをダウンロードしたら、ファイルを展開してソース・ディレクトリーを作成します。その後、次の操作を行います。

1. numpy のルートフォルダー **`numpy-x.x.x`** に移動し、既存のファイルから `site.cfg` を作成します。
2. `site.cfg` を次のように編集します。
   a. 64 ビット用の NumPy をビルドする場合は、トップレベルの NumPy ディレクトリーの `site.cfg` に次のように指定します（インテル® Parallel Studio XE 2013 またはインテル® Composer XE 2013 バージョンのデフォルトパスにインテル® MKL をインストールした場合）。

```
[mkl]
library_dirs = /opt/intel/composer_xe_2013_sp1/mkl/lib/intel64
include_dirs = /opt/intel/mkl/include
mkl_libs = mkl_rt
lapack_libs =
```

   b. 32 ビット用の NumPy をビルドする場合は、次のように指定します。

```
[mkl]
library_dirs = /opt/intel/composer_xe_2013_sp1/mkl/lib/ia32
include_dirs = /opt/intel/mkl/include
mkl_libs = mkl_rt
lapack_libs =
```

3. numpy/distutils/intelccompiler.py の cc_exe を次のように変更します。

```
self.cc_exe = 'icc -O3 -xavx -ipo -g -fPIC -fp-model strict -fomit-frame-pointer
-openmp -DMKL_ILP64'
```

ここでは、**-O3** 最適化オプションを指定して、融合、アンロールとジャムのブロック、IF 文の折りたたみなど、より強力なループ変換を有効にしています。また、OpenMP* スレッド用の **-openmp** オプションを使用し、**-xavx** オプションでインテル® AVX 命令を生成するようにコンパイラーに指示します。プロセッサーのアーキテクチャーが不明な場合は、**-xHost** オプションを指定すると、コンパイル時にホストのプロセッサーが持つ最上位命令セットが使用されます。ILP64 インターフェイスを使用している場合は、**-DMKL_ILP64** コンパイラー・オプションを追加します。

プロセッサー固有のオプションの詳細は、`icc --help` を実行して確認できます。各種コンパイラー・オプションの詳細については、インテル® コンパイラーのドキュメントを参照してください。

インテル® Fortran コンパイラー向けのコンパイラー・オプションを使用するように、numpy-x.x.x/numpy/distutil/fcompiler/intel.py の Fortran コンパイラー設定ファイルを変更します。

32 ビットおよび 64 ビットの場合、次のオプションを指定します。

```
ifort -xhost -openmp -fp-model strict -i8 -fPIC
```

次のように、インテル® コンパイラーで NumPy をコンパイルしインストールします (32 ビット・プラットフォームの場合は "**intelem**" を "**intel**" に変更します)。

```
$python setup.py config --compiler=intelem build_clib
--compiler=intelem build_ext --compiler=intelem install
```

次のように、インテル® コンパイラーで SciPy をコンパイルしインストールします (32 ビット・プラットフォームの場合は "**intelem**" を "**intel**" に変更します)。

```
$python setup.py config --compiler=intelem --fcompiler=intelem
build_clib --compiler=intelem --fcompiler=intelem build_ext
--compiler=intelem --fcompiler=intelem install
```

次のように、**LD_LIBRARY_PATH** 環境変数をエクスポートして、インテル® MKL とインテル® コンパイラーのライブラリーのパスをセットアップします。

64 ビット・プラットフォームの場合 :

```
$export
LD_LIBRARY_PATH=/opt/intel/composer_xe_2013_sp1/mkl/lib/intel64:/opt/
intel/composer_xe_2013_sp1/lib/intel64:$LD_LIBRARY_PATH
```

32 ビット・プラットフォームの場合 :

```
$export
LD_LIBRARY_PATH=/opt/intel/composer_xe_2013_sp1/mkl/lib/ia32:/opt/intel/
composer_xe_2013_sp1/lib/ia32:$LD_LIBRARY_PATH
```

インテル® MKL とインテル® Composer XE をデフォルト以外のディレクトリーにインストールした場合、**LD_LIBRARY_PATH** 変数が正しく動作しないことがあります。この問題を回避するには、**LD_RUN_PATH** 変数を設定した環境内で Python*、NumPy、SciPy をビルドします。例えば、32 ビット・プラットフォームでは、次のようにパスを設定します。

```
$export LD_RUN_PATH=/opt/intel/composer_xe_2013_sp1/lib/ia32:/opt/
intel/composer_xe_2013_sp1/mkl/lib/ia32
```

注 : パフォーマンスの観点から、および NumPy が CBLAS を使用することから、Fortran 形式（列優先）ではなく、デフォルトの 'C' 形式（行優先）の配列を使用することを推奨します。最新の情報はこちらをご覧ください。

インテル® コンパイラーとインテル® MKL を使用した R のインストール

R は、統計計算、計算生物学、ビッグデータ・アナリティクスで頻繁に使用される複雑な計算を行う、高水準プログラミング言語です。R には、行列演算、数値積分、高度な統計などを実行する高レベルな関数が含まれており、さまざまな科学技術計算アプリケーションで利用できます。

ここでは、インテル® MKL の BLAS と LAPACK ライブラリーを介して R のパフォーマンスを改善する方法を説明します。ほかのインテル® MKL 関数を用いる場合は、ラッパーを利用します。インテル® Core™ i7 マシン（4 コア、SMT 有効）上でインテル® MKL とインテル® コンパイラーの最適化を有効にして、標準 R ベンチマークの 3000x3000 行列のコレスキー分解を実行したところ、GNU* ツールの場合に比べて約 20 倍もスピードアップしました。

インテル® MKL の BLAS と LAPACK ライブラリーを使用するように R を設定するには、設定時に **-with-blas** オプションを指定します。

インテル® コンパイラーの最適化を活かすには、まず、設定プロセスで使われる AR および LD 変数をエクスポートして、インテル® コンパイラーのリンカーとライブラリー・アーカイブ・ツール用の環境変数を設定します。

```
$export AR="xiar"

$export LD="xild"
```

インテル® MKL と OpenMP* ライブラリーのパスを設定します。

```
$MKL_LIB_PATH='/opt/intel/composer_xe_2013_sp1/mkl/lib/intel64'

$OMP_LIB_PATH='/opt/intel/lib/intel64'

$export LD_LIBRARY_PATH=${MKL_LIB_PATH}:${OMP_LIB_PATH}

$MKL=" -L${MKL_LIB_PATH} -L${OMP_LIB_PATH} -lmkl_intel_lp64 -
lmkl_intel_thread -lmkl_core -liomp5 -lpthread"
```

インテル® コンパイラーとコンパイラーの最適化機能を使用するように、R.x.x.x ソースのルートフォルダーに含まれている **config.site** ファイルを変更します。

1. 次のように C/C++ および Fortran コンパイラーとオプション関連行を変更して、config.site を編集します。

```
CC='icc -std=c99'
CFLAGS='-O3 -ipo -xavx -openmp'
F77='ifort'
FFLAGS='-O3 -ipo -xavx -openmp'
CXX='icpc'
CXXFLAGS='-O3 -ipo -xavx -openmp'
$./configure --with-blas="$MKL" --with-lapack
```

   デフォルトのスレッド数はシステムの物理コアの数と同じですが、OMP_NUM_THREADS または MKL_NUM_THREADS を設定して制御できます。

2. ログファイル config.log を調べて、インテル® MKL が構成テスト中に動作していたかどうかを確認します。

```
configure:29075: checking for dgemm_ in -lmkl_intel_lp64
-lmkl_intel_thread -lmkl_core -liomp5 -lpthread

configure:29096: icc -std=c99 -o conftest -O3 -ipo -openmp -xHost
-I/usr/local/include  -L/usr/local/lib64 conftest.c -lmkl_intel_lp64
-lmkl_intel_thread -lmkl_core -liomp5 -lpthread  -lifport -lifcoremt
-limf -lsvml -lm -lipgo -liomp5 -lirc -lpthread -lirc_s -ldl -lrt -ldl -lm  >&5

conftest.c(210): warning #266: function "dgemm_" declared implicitly
  dgemm_()
  ^

configure:29096: $? = 0
configure:29103: result: yes
configure:29620: checking whether double complex BLAS can be used
configure:29691: result: yes
configure:29711: checking whether the BLAS is complete
```

3. 次に、make を実行し、インテル® コンパイラーとインテル® MKL を使用して R をビルドしインストールします。

```
$make && make install
```

4. インテル® MKL ライブラリーが動的にリンクされている場合、ldd コマンドを実行してインテル® MKL とインテル® コンパイラーのライブラリーが R にリンクされていることを確認します。インテル® コンパイラーを使用して、インテル® MKL が正常にリンクされた場合、出力は次のようになります。

```
$ ldd ./bin/exec/R
        linux-vdso.so.1 =>  (0x00007fff2a1ff000)
        libmkl_intel_lp64.so =>
/opt/intel/composer_xe_2013_sp1.0.080/mkl/lib/intel64/libmkl_intel_lp64.so
(0x00007f7d89bf9000)
        libmkl_intel_thread.so =>
/opt/intel/composer_xe_2013_sp1.0.080/mkl/lib/intel64/libmkl_intel_thread.so
(0x00007f7d88b5e000)
        libmkl_core.so =>
/opt/intel/composer_xe_2013_sp1.0.080/mkl/lib/intel64/libmkl_core.so
(0x00007f7d87630000)
        libiomp5.so =>
/opt/intel/composer_xe_2013_sp1.0.080/compiler/lib/intel64/libiomp5.so
(0x00007f7d87317000)
        libpthread.so.0 => /lib64/libpthread.so.0
(0x00000036ec600000)
        libifport.so.5 =>
/opt/intel/composer_xe_2013_sp1.0.080/compiler/lib/intel64/libifport.so.5
(0x00007f7d870c8000)
        libifcoremt.so.5 =>
/opt/intel/composer_xe_2013_sp1.0.080/compiler/lib/intel64/libifcoremt.so.5
(0x00007f7d86d59000)
        libimf.so =>
/opt/intel/composer_xe_2013_sp1.0.080/compiler/lib/intel64/libimf.so
(0x00007f7d8689a000)
        libsvml.so =>
/opt/intel/composer_xe_2013_sp1.0.080/compiler/lib/intel64/libsvml.so
(0x00007f7d85e3f000)

.............
```

R のパフォーマンスを検証するため、クアッドコアのインテル® Core™ i7-975 プロセッサー エクストリーム・エディション（8MB LLC、3.33GHz）、メモリー 16GB、RHEL* 6 x86_64 のシステムで、R benchmarks サイト（英語）の R-benchmark-25.R 使ってテストしました。最初に R-2.15.3 をインストールし、GNU* ツールチェーンを使用してベンチマークを実行しました。

インテル® Core™ i7 3770K クアッドコア（8MB LLC、3.50 GHz）、メモリー 16GB、Ubuntu* 13.1 のシステム上でインテル® MKL を使用して標準 R ベンチマーク R-2.15.3 を実行したところ、大幅にパフォーマンスが向上しました。次の結果で、最後の項目はベンチマークのすべてのテストの合計時間を示しています（単位はすべて秒）。

	GNU*	インテル® MKL & インテル® コンパイラー
2800x2800 行列のクロス積 (b = a' * a)	9.4276667	0.2966667
3000x3000 行列のコレスキー分解	3.7346666	0.1713333
15 種類の全テストの合計時間	29.1816667	5.4736667

インテル® MKL を使用した Julia のインストール

最後に、HPC コミュニティーで評判が高まっている Julia 言語のビルドとインストール方法を説明します。Julia は、ハイパフォーマンスな数値計算と科学技術計算に焦点を当てた高水準の動的プログラミング言語ですが、汎用プログラミングにも効果があります。Julia もインタープリター型言語ですが、コンパイラー型言語と同等の高い機能を提供するよう設計されています。Julia の実装設計は、コア言語プラットフォームとして多重ディスパッチを含み、高度な型システム、より速いコードを生成する動的な型推測、実行時の型に応じて生成されるコードの特殊化などに対応します。インテル® Core™ i7 マシン（4 コア）でのレベル 3 BLAS パフォーマンス・テストでは、インテル® MKL を使用した場合、デフォルトのインストールに比べてパフォーマンスが 1.5 ～ 2 倍に向上しました。

Julia のソースコードは GIT hub（https://github.com/JuliaLang/julia/（英語））からダウンロードできます

インテル® MKL 環境をセットアップするには、64 ビットと 8 バイト整数（ILP64）をサポートするようにビルドします。

次のようにインテル® 64 および ilp64 用の引数を渡して `mklvars.sh` スクリプトを実行し、インテル® MKL 環境変数を設定します。

```
$source /opt/intel/mkl/bin/mklvars.sh intel64 ilp64
```

32 ビット・アーキテクチャーの場合、環境変数を設定する引数として ia32 を指定します。

```
$source /opt/intel/mkl/bin/mklvars.sh ia32
```

今回のケースでは、64 ビットと ILP64 アーキテクチャーを使用するには、インテル® MKL のインターフェイス・レイヤー変数を次のように設定する必要がありました。

```
$export MKL_INTERFACE_LAYER=ILP64
```

次に、Julia のルートフォルダーにある Make.inc を編集するか、Make.user ファイルを作成して次の変更を加えます。

`USE_MKL=0` 変数を `USE_MKL=1` に変更します。

`USE_INTEL_IMF=1` を追加します。

インテル® MKL 環境を設定してインテル® MKL を使用するように Make.inc ファイルを変更したら、次のコマンドを実行してインテル® MKL 対応の Julia をビルドします。

```
$make install
```

make に引数を渡してインテル® MKL 対応の Julia をビルドすることもできます。

```
$make USE_MKL=1 install
```

インテル® コンパイラーの最適化機能を利用するには、次のコマンドで Julia をビルドします。

```
$make -j 4 CC=icc CFLAGS="-O3 -xavx" CXX=icpc CXXFLAGS="-O3
-xavx"  FC=ifort FFLAGS="-O3 -xavx" USE_MKL=1 USE_INTEL_IMF=1
```

ここでは、-j 4 を渡して 4 つのスレッドでビルドを実行していることに注意してください。この値は、システムで利用可能なスレッド数に応じて変更してかまいません。

インテル® MKL 対応の Julia ソースをリビルドするには、Julia の /deps フォルダーから OpenBLAS、ARPACK、SuiteSparse の依存関係を削除して、次のコマンドを実行します。

```
$make cleanall testall
```

インテル® ソフトウェア・ツールに対応した Julia のパフォーマンスを検証するには、Julia/test/perf/blas と Julia/test/perf/lapack フォルダーに含まれている各種パフォーマンス・テストを実行します。

まとめ

Python*、R、Julia は、高速なプログラミングと可読性に優れたコードの開発を可能にするポピュラーな言語です。HPC、アナリティクス、科学技術計算分野の開発者は、これらの言語をインテル® ソフトウェア・ツールを使用してビルドすることで、最新のインテル® アーキテクチャーの機能を活用することができます。システム上のコアのすべての機能を利用するには、インテル® コンパイラーのベクトル化オプションと OpenMP* のベクトル化サポートにより、SIMD/AVX ベクトル命令を活用します。コンパイラーは、プロシージャー間の最適化やその他のパフォーマンス最適化機能によって、複数ファイルの最適化を行います。インテル® MKL ライブラリーは、優れたアルゴリズム、ベクトル化、OpenMP* スレッド化を通じて最新のインテル® アーキテクチャー向けに最適化された、ハイパフォーマンスな業界標準の数値演算ライブラリーです。条件付き数値再現性機能を用いると、再現可能な結果を得ることもできます。インテル® コンパイラーを使用して簡単な作業を行うだけで、ハイパフォーマンスなインテル® MKL 対応の Python*、R、Julia をビルドしインストールすることができます。システムにインテル® Xeon Phi™ コプロセッサーが搭載されている場合、インテル® MKL のホスト・プロセッサーとコプロセッサー間で計算を分割する機能が自動的に有効になり、パフォーマンスがさらに向上するでしょう。

インテル® コンパイラーを使用して簡単な作業を行うだけで、ハイパフォーマンスなインテル® マス・カーネル・ライブラリー（インテル® MKL）対応の Python*、R、Julia をビルドしインストールすることができます。

Parallel Universe 17 号の全文はこちらでご覧いただけます。