レシピ: インテル® Xeon® プロセッサーとインテル® Xeon Phi™ プロセッサー上でシングルノードで動作する NAMD をビルドする

この記事は、インテル® デベロッパー・ゾーンに公開されている「Recipe: Building NAMD on Intel® Xeon® and Intel® Xeon Phi™ Processors on a Single Node」の日本語参考訳です。

クラスターで動作する NAMD のビルドについては、「**レシピ: インテル® Xeon® プロセッサーとインテル® Xeon Phi™ プロセッサー上でマルチノードで動作する NAMD をビルドする**」を参照してください。

目的

このレシピは、優れたパフォーマンスを達成するため、インテル® Xeon Phi™ プロセッサーおよびインテル® Xeon® プロセッサー E5 ファミリーで NAMD (スケーラブルな分子動力学コード) を準備、ビルド、実行する手順を説明します。

はじめに

NAMD は、大規模な生体分子系のハイパフォーマンス・シミュレーション向けに設計された並列分子動力学コードです。Charm++* 並列オブジェクトをベースに、NAMD は典型的なシミュレーションでは数百コア、大規模なシミュレーションでは 50 万を超えるコアにスケーリングします。NAMD は、シミュレーションの設定と軌道解析にポピュラーな分子グラフィックス・プログラム VMD を使用しますが、AMBER、CHARMM*、X-PLOR* とファイル互換です。

NAMD は、ソースコードを含め、無料で提供されています。自身で NAMD をビルドするか、さまざまなプラットフォーム向けのバイナリーをダウンロードすることができます。インテル® Xeon Phi™ プロセッサーとインテル® Xeon® プロセッサー E5 ファミリーで NAMD をビルドする方法を以下に示します。NAMD の詳細は、http://www.ks.uiuc.edu/Research/namd/ (英語) を参照してください。

インテル® Xeon® プロセッサー E5-2697 v4 (開発コード名 Broadwell (BDW))、インテル® Xeon Phi™ プロセッサー 7250 (開発コード名 Knights Landing (KNL))、およびインテル® Xeon® Gold 6148 プロセッサー (開発コード名 Skylake (SKX)) ベースのクラスター向けに NAMD をビルドして実行する

コードのダウンロード

1. http://www.ks.uiuc.edu/Development/Download/download.cgi?PackageName=NAMD (英語) から最新の NAMD ソースコードをダウンロードします。
2. Charm++* 6.7.1 をダウンロードします。
   a. NAMD ソースコードのナイトリービルドから Charm++* を取得できます。
   b. または、http://charmplusplus.org/download/ (英語) から別途ダウンロードできます。

3. http://www.fftw.org/download.html (英語) から FFTW3 をダウンロードします。

ここでは、バージョン 3.3.4 を使用します。

4. http://www.ks.uiuc.edu/Research/namd/utilities/ (英語) から ApoA1 および STMV ワークロードをダウンロードします。

バイナリーのビルド

1. コンパイル環境を設定します。
```
CC=icc; CXX=icpc; F90=ifort; F77=ifort
export CC CXX F90 F77
source /opt/intel/compiler/<version>/compilervars.sh intel64
```

2. FFTW3 をビルドします。
   a.
```
cd <fftw_root_path>
```
   b.
```
./configure --prefix=<fftw_install_path> --enable-single --
disable-fortran CC=icc
```
   SKX † の場合は -xCORE-AVX512、KNL † の場合は -xMIC-AVX512、BDW † の場合は -xCORE-AVX2 を使用します。
   c.
```
make CFLAGS="-O3 -xMIC-AVX512 -fp-model fast=2 -no-prec-div -
qoverride-limits" clean install
```

3. Charm++* のマルチコアバージョンをビルドします。
   a.
```
cd <charm_root_path>
```
   b.
   src/arch/multicore-linux64/conv-mach.h で CMK_TIMER_USE_RDTSC タイマーを設定し、ほかのタイマーの設定を解除します。
```
define CMK_TIMER_USE_RDTSC                                  1
define CMK_TIMER_USE_GETRUSAGE                              0
define CMK_TIMER_USE_SPECIAL                                0
define CMK_TIMER_USE_TIMES                                  0
```
   c.
```
./build charm++ multicore-linux64 iccstatic --with-production "-
O3 -ip"
```

4. NAMD をビルドします。
   a. arch/Linux-x86_64-icc を次のように変更します (CPU タイプに応じて適切な FLOATOPTS オプションを選択します)。

```
NAMD_ARCH = Linux-x86_64
CHARMARCH = multicore-linux64-iccstatic

# KNL† の場合
FLOATOPTS = -ip -xMIC-AVX512  -O3 -g -fp-model fast=2 -no-prec-
div -qoverride-limits -DNAMD_DISABLE_SSE

# SKX† の場合
FLOATOPTS = -ip -xCORE-AVX512  -O3 -g -fp-model fast=2 -no-prec-
div -qoverride-limits -DNAMD_DISABLE_SSE

# BDW† の場合
```

```
FLOATOPTS = -ip -xCORE-AVX2  -O3 -g -fp-model fast=2 -no-prec-div
-qoverride-limits -DNAMD_DISABLE_SSE

CXX = icpc -std=c++11 -DNAMD_KNL
CXXOPTS = -static-intel -O2 $(FLOATOPTS)
CXXNOALIASOPTS = -O3 -fno-alias $(FLOATOPTS) -qopt-report-
phase=loop,vec -qopt-report=4
CXXCOLVAROPTS = -O2 -ip
CC = icc
COPTS = -static-intel -O2 $(FLOATOPTS)
```

b. NAMD をコンパイルします。

i.
```
./config Linux-x86_64-icc --charm-base <charm_root_path> --
charm-arch multicore-linux64- iccstatic --with-fftw3 --fftw-
prefix <fftw_install_path> --without-tcl --charm-opts -verbose
```

ii.
```
gmake -j
```

その他のシステム設定

1. **KNL†のカーネル設定を変更します:** “nmi_watchdog=0 rcu_nocbs=2-271 nohz_full=2-271”。**以下に、設定の変更方法の例を示します (これは、システムごとに異なります)。**

   a. 念のため、最初にオリジナルの grub.cf**g のコピーを作成します。**
   ```
   cp /boot/grub2/grub.cfg /boot/grub2/grub.cfg.ORIG
   ```

   b. “/etc/default/grub” **に次の行を追加します。**
   ```
   “GRUB_CMDLINE_LINUX”: nmi_watchdog=0 rcu_nocbs=2-271 nohz_full=2-
   271
   ```

   c. 新しい設定を保存します。
   ```
   grub2-mkconfig -o /boot/grub2/grub.cfg
   ```

   システムを再起動します。ログイン後、“cat /proc/cmdline” **で設定を確認します。**

2. **それぞれのワークロードの *.namd ファイルで、次の行を変更します。**
   numsteps 1000
   outputtiming 20
   outputenergies 600

NAMD の実行

- **SKL†/BDW†(それぞれ ppn = 40 / ppn = 72):**
  ```
  ./namd2 +p $ppn apoa1/apoa1.namd +pemap 0-($ppn-1)
  ```
- **KNL†(ppn = 136 (コアごとに 2 ハイパースレッド)、MCDRAM フラットモード (キャッシュに似たパフォーマンス)):**
  ```
  numactl -p 1 ./namd2 +p $ppn apoa1/apoa1.namd +pemap 0-($ppn-1)
  ```

KNL†の例

```
numactl -p 1 <namd_root_path>/Linux-KNL-icc/namd2 +p 136 apoa1/apoa1.namd
+pemap 0-135
```

インテルの Salesforce リポジトリーで報告されたパフォーマンス結果 (ns/day、値が大きいほうが良い):

ワークロード	2x インテル® Xeon® プロセッサー E5-2697 v4 18 コア 2.30GHz (ns/day)	インテル® Xeon Phi™ プロセッサー 7250 bin1 (ns/day)	インテル® Xeon Phi™ プロセッサー 7250 と 2x インテル® Xeon® プロセッサー E5-2697 v4 の比較 (スピードアップ)
STMV	0.45	0.55	1.22 倍
ApoA1	5.5	6.18	1.12 倍

ワークロード	2x インテル® Xeon® Gold 6148 プロセッサー 20 コア 2.40GHz (ns/day)	インテル® Xeon Phi™ プロセッサー 7250 と 2x インテル® Xeon® Gold 6148 プロセッサーの比較 (スピードアップ)
STMV	0.73	1.44 倍
オリジナルの ApoA1	7.68	1.43 倍
ApoA1	8.70	1.44 倍

システム構成

プロセッサー	インテル® Xeon® プロセッサー E5-2697 V4	インテル® Xeon® Gold 6148 プロセッサー	インテル® Xeon Phi™ プロセッサー 7250
ステッピング	1 (B0)	1 (B0)	1 (B0) Bin1
ソケット数/TDP	2S/290W	2S/300W	1S/215W
周波数/コア数/スレッド数	2.30GHz/36/72	2.40GHz/40/80	1.40GHz/68/272
DDR4	8x16GB 2400MHz (128GB)	12x16GB 2666MHz (192GB)	6x16GB 2400MHz
MCDRAM	N/A	N/A	16GB フラットモード
クラスター/スヌープモード/メモリーモード	Home	Home	クワドラント/フラット
インテル® ターボ・ブースト・テクノロジー	オン	オン	オン
BIOS	GRRFSDP1.86B0271.R00.1510301446		GVPRCRB1.86B.0010.R02.1608040407
コンパイラー	ICC-2017.0.098	ICC-2016.4.298	ICC-2017.0.098
OS	Red Hat* Enterprise Linux* 7.2	Red Hat* Enterprise Linux* 7.3	Red Hat* Enterprise Linux* 7.2
	(3.10.0-327.e17.x86_64)	(3.10.0-514.6.2.0.1.el7.x86_64.knl1)	(3.10.0-327.22.2.el7.xppsl_1.4.1.3272._86_64)

著者紹介

Alexander Bobyr は、インテルの INNL ラボの CRT アプリケーション・エンジニアで、HPC とソフトウェア・ツールのサポートとフィードバックを提供しています。SPEC* HPG のテクニカル・エキスパート兼代表を務めており、モスクワ発電工学研究所 (工科大学) からインテリジェント・システム管理の学士号と人工知能の修士号を取得しています。

Mikhail Shiryaev は、インテルのソフトウェア & サービスグループ (SSG) のソフトウェア開発エンジニアで、クラスター・ツール・チームの一員として、インテル® MPI ライブラリーとインテル® MLSL の開発に取り組んでいます。主に、ハイパフォーマンス・コンピューティング、分散システム、分散型ディープラーニングに関心があります。ロシア、ニジニ・ノヴゴロドのロバチェフスキー州立大学からソフトウェア・エンジニアリングの学士号と修士号を取得しています。

Smahane Douyeb は、インテルのソフトウェア & サービスグループ (SSG) のソフトウェア・アプリケーション・エンジニアで、競争力をテストするため、さまざまな HPC プラットフォーム向けのレシピとベンチマークを実行し、検証しています。いくつかのインテル® プラットフォーム上で HPC Python* アプリケーションの最適化にも取り組んでいます。オレゴン工科大学からソフトウェア・エンジニアリングの学士号を取得しています。主席エンジニアを目指して成長し、学習しています。

†開発コード名

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。