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

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 for Multi-node Runs」の日本語参考訳です。

シングルノードで動作する 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™ プロセッサー上でシングルノードで動作する 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. Open Fabric Interfaces (OFI) をダウンロードします。NAMD は、マルチノード・バージョンの Charm++*/OFI を使用します。
    - IFS パッケージによってインストールされる OFI ライブラリーを使用するか、手動でダウンロードしてビルドすることができます。

- インストールされている OFI のバージョンをチェックするには、"f_info --version" コマンドを使用します (ここでは、**OFI1.4.2** を使用)。
- OFI ライブラリーは、https://github.com/ofiwg/libfabric/releases (英語) からダウンロードできます。

3. 次のサイトから OFI 対応の Charm++* をダウンロードします。
   http://charmplusplus.org/download/ (英語)
   または
   http://charm.cs.illinois.edu/gerrit/charm.git (英語) - git クローン

4. 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。

5. 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. OFI ライブラリーをビルドします (インストールされている OFI ライブラリーを使用する場合はこのステップをスキップします)。
   1. `cd <libfabric_root_path>`
   2. `./autogen.sh`
   3. `./configure --prefix=<libfabric_install_path> --enable-psm2`
   4. `make clean && make -j12 all && make install`
   5. LD_PRELOAD または LD_LIBRARY_PATH を設定して、カスタム OFI を使用することもできます。

```
export LD_LIBRARY_PATH=<libfabric_install_path>/lib:${LD_LIBRARY_PATH}
mpiexec.hydra …
```

   または

```
LD_PRELOAD=<libfabric_install_path>/lib/libfabric.so mpiexec.hydra …
```

3. FFTW3 をビルドします。
   1. `cd <fftw_root_path>`
   2. `./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。
   3. `make CFLAGS="-O3 -xMIC-AVX512 -fp-model fast=2 -no-prec-div -qoverride-limits" clean install`
4. Charm++* のマルチノード・バージョンをビルドします。
   1. `cd <charm_root_path>`
   2. `src/arch/ofi-linux-x86_64/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_RTC                                   0
#define CMK_TIMER_USE_RDTSC                                 1
#define CMK_TIMER_USE_GETRUSAGE                             0
```

```
#define CMK_TIMER_USE_SPECIAL                                   0
#define CMK_TIMER_USE_TIMES                                     0
#define CMK_TIMER_USE_BLUEGENEL                                 0
```

src/arch/ofi-linux-x86_64/conv-mach-smp.h ファイルで CMK_TIMER_USE_GETRUSAGE の設定を解除します。

```
#undef CMK_TIMER_USE_GETRUSAGE
#undef CMK_TIMER_USE_SPECIAL
#define CMK_TIMER_USE_GETRUSAGE                                 0
#define CMK_TIMER_USE_SPECIAL                                   0
```

3. `./build charm++ ofi-linux-x86_64 icc smp --basedir <libfabric_root_path> --with-production “-O3 -ip” -DCMK_OPTIMIZE`

5. **NAMD をビルドします。**
   1. **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)
```

   2. **NAMD をコンパイルします。**
      1. `./config Linux-x86_64-icc --charm-base <charm_root_path> --charm-arch ofi-linux-x86_64-smp-icc --with-fftw3 --fftw-prefix <fftw_install_path>--without-tcl --charm-opts -verbose`
      2. `cd Linux-x86_64-icc`
      3. `make clean && gmake -j`

6. **memopt NAMD バイナリーをビルドします。**

上記の BDW†/KNL†の NAMD ビルドと同様ですが、config に "-with-memopt" を追加します。

その他の設定

STMV および ApoA1 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

バイナリーの実行

1. 起動環境を設定します。
    1. source /opt/intel/compiler/<version>/compilervars.sh intel64
    2. source /opt/intel/impi/<version>/intel64/bin/mpivars.sh
    3. "hosts" ファイルでバイナリーを実行するホスト名を指定します。
    4. export MPIEXEC="mpiexec.hydra -hostfile ./hosts"
    5. export PSM2_SHAREDCONTEXTS=0 (PSM2 < 10.2.85 の場合)
2. タスクを起動します。例えば、N ノード、ノードごとに 1 プロセス、PPN コアの場合、次のコマンドを実行します。
    1. $MPPEXEC -n N -ppn 1 ./namd2 +ppn (PPN-1) <workload_path> +pemap 1-(PPN-1) +commap 0

       BDW† の場合 (PPN=72):
       $MPPEXEC -n 8 -ppn 1 ./namd2 +ppn 71 <workload_path> +pemap 1-71 +commap 0

       KNL† の場合 (PPN=68、ハイパースレッド無効):
       $MPPEXEC -n 8 -ppn 1 ./namd2 +ppn 67 <workload_path> +pemap 1-67 +commap 0

       KNL† の場合 (コアごとに 2 ハイパースレッド):
       $MPPEXEC -n 8 -ppn 1 ./namd2 +ppn 134 <workload_path> +pemap 0-66+68 +commap 67

    2. MCDRAM をフラットモードに設定した KNL†の場合:
       $MPPEXEC -n N -ppn 1 numactl -p 1 ./namd2 +ppn (PPN-1) <workload_path> +pemap 1-(PPN-1) +commap 0

留意事項

マルチコードで優れたスケーリングを達成するには、通信スレッドの数を増やします (1、2、4、8、13、17)。例えば、次のコマンドは、ノードごとに 17 プロセス、プロセスごとに 8 スレッドの N 個の KNL†ノード (7 ワーカースレッドと 1 通信スレッド) を指定します。

```
$MPPEXEC -n $(($N*17)) -ppn 17 numactl -p 1 ./namd2 +ppn 7 <workload_path>
+pemap 0-67,68-135:4.3 +commap 71-135:4
```

基本的な Charm++*/OFI オプション (NAMD パラメーターとして追加)

- +ofi_eager_maxsize: (デフォルト: 65536) バッファーされたパスと RMA パス間のしきい値。
- +ofi_cq_entries_count: (デフォルト: 8) 各 fi_cq_read() 呼び出しで完了キューから読み取るエントリーの最大数。
- +ofi_use_inject: (デフォルト: 1) バッファー送信を使用するかどうか。
- +ofi_num_recvs: (デフォルト: 8) 事前にポストされた受信バッファーの数。
- +ofi_runtime_tcp: (デフォルト: オフ) 初期化フェーズ中に OFI EP 名をすべてのノード間で交換します。
  デフォルトでは、PMI と OFI の両方で交換が行われます。このフラグがオンの場合は、PMI でのみ交換が行われます。

例:

```
$MPPEXEC -n 2 -ppn 1 ./namd2 +ppn 1 <workload_path> +ofi_eager_maxsize 32768
+ofi_num_recvs 16
```

最大 128 のインテル® Xeon Phi™ プロセッサー・ノードで構成されるクラスターで最高のパフォーマンスを達成 (ns/day、値が大きいほうが良い)

ワークロード/ノード (2HT)	1	2	4	8	16
stmv (ns/day)	0.55	1.05	1.86	3.31	5.31
ワークロード/ノード (2HT)	8	16	32	64	128
stmv.28M (ns/day)	0.152	0.310	0.596	1.03	1.91

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います。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。