AMBER22 update 1

ウェブページ

http://ambermd.org/

バージョン

Amber22 update 1, AmberTools 22 update 4

ビルド環境

- GCC 10.3.1 (gcc-toolset-10)
- CUDA 11.6
- OpenMPI 4.1.4 (HPC-X 2.11)
    - 実際のビルド時には OpenMPI 4.1.5 (HPC-X 2.13.1 を使用)していたが、問題が発生したため、runtime だけを HPC-X 2.11 に切り替え
    - 以下の手順では HPC-X 2.11 で記述。

ビルドに必要なファイル

- Amber22.tar.bz2
- AmberTools22.tar.bz2
- patch-cmake-python (miniconda のかわりに miniforge を使用)

```
--- cmake/UseMiniconda.cmake.org        2022-05-27 09:43:57.000000000 +0900
+++ cmake/UseMiniconda.cmake    2022-05-27 09:56:28.000000000 +0900
@@ -84,11 +84,14 @@
             endif()
       endif()

-      set(MINICONDA_INSTALLER_FILENAME "Miniconda${PYTHON_MAJOR_RELEASE}-${MINICONDA_VERSION}-${CONTINUUM_SYSTEM_NAME}-${CONTINUUM_BITS}.${INSTALLER_SUFFIX}")
+      #set(MINICONDA_INSTALLER_FILENAME "Miniconda${PYTHON_MAJOR_RELEASE}-${MINICONDA_VERSION}-${CONTINUUM_SYSTEM_NAME}-${CONTINUUM_BITS}.${INSTALLER_SUFFIX}")
+   ## mkamiya: assume x86_64 Linux...
+      set(MINICONDA_INSTALLER_FILENAME "Miniforge${PYTHON_MAJOR_RELEASE}-${MINICONDA_VERSION}-Linux-x86_64.sh")

       # location to download the installer to
       set(MINICONDA_INSTALLER ${MINICONDA_DOWNLOAD_DIR}/${MINICONDA_INSTALLER_FILENAME})
-      set(INSTALLER_URL "http://repo.continuum.io/miniconda/${MINICONDA_INSTALLER_FILENAME}")
+      #set(INSTALLER_URL "http://repo.continuum.io/miniconda/${MINICONDA_INSTALLER_FILENAME}")
+      set(INSTALLER_URL "https://github.com/conda-forge/miniforge/releases/download/${MINICONDA_VERSION}/${MINICONDA_INSTALLER_FILENAME}")

       # If we've already downloaded the installer, use it.
       if(EXISTS "${MINICONDA_INSTALLER}")
```

ビルド手順

```
#!/bin/sh

VERSION=22
TOOLSVERSION=22

MINIFORGE_VERSION="4.11.0-4" # ad hoc custom version

# amber22 + AmberTools22
INSTALL_DIR="/apl/amber/22u1"
WORKDIR="/work/users/${USER}/work"
TARBALL_DIR="/home/users/${USER}/Software/AMBER/22"

PATCHX=${TARBALL_DIR}/patch-cmake-python
```

```
PARALLEL=12

#-----------------------------------------------------------------------
module purge
module load gcc-toolset/10
module load openmpi/4.1.4-hpcx/gcc10
module load cuda/11.6

export LANG=C
export LC_ALL=C
ulimit -s unlimited

# install directory has to be prepared before running this script
if [ ! -d $WORKDIR ]; then
  echo "Create $WORKDIR before running this script."
  exit 1
fi

# build directory must be empty
if [ "$(ls -A $WORKDIR)" ]; then
  echo "Target directory $WORKDIR not empty"
  exit 2
fi
# install directory must be empty
if [ "$(ls -A $INSTALL_DIR)" ]; then
  echo "Target directory $INSTALL_DIR not empty"
  exit 2
fi

# prep files
cd $WORKDIR
if [ -d amber${VERSION}_src ]; then
  mv -f amber${VERSION}_src amber_erase
  rm -rf amber_erase &
fi

bunzip2 -c ${TARBALL_DIR}/Amber${VERSION}.tar.bz2 | tar xf -
bunzip2 -c ${TARBALL_DIR}/AmberTools${TOOLSVERSION}.tar.bz2 | tar xf -

# prep python and update
cd amber${VERSION}_src
export AMBERHOME=${WORKDIR}/amber${VERSION}_src

patch -p0 < $PATCHX
sed -i -e "s/latest/${MINIFORGE_VERSION}/" cmake/PythonInterpreterConfig.cmake

# CPU serial with installation of tests
echo "[CPU serial edition]"
mkdir build_cpu_serial && cd build_cpu_serial
cmake .. \
    -DCMAKE_INSTALL_PREFIX=${INSTALL_DIR} \
    -DCOMPILER=GNU \
    -DMPI=FALSE \
    -DCUDA=FALSE \
    -DINSTALL_TESTS=TRUE \
    -DDOWNLOAD_MINICONDA=TRUE \
    -DFORCE_INTERNAL_LIBS="arpack" \
    -DBUILD_QUICK=TRUE \
    -DCHECK_UPDATES=TRUE

make -j${PARALLEL} install && make clean
cd ../ && rm -rf build_cpu_serial

# mark its origin at installation directory
cd ${INSTALL_DIR}
```

```
ln -s ./miniconda ./miniforge

# ad hoc fix shebang of amber.conda
cd miniconda/bin
# ad hoc ad hoc ad hoc ad hoc
perm=$(stat -c "%a" conda)
head -n 1 ipython >> conda.new
sed -e "1d" conda >> conda.new
mv -f conda.new conda
chmod $perm conda
# ad hoc ad hoc ad hoc ad hoc

cd ${WORKDIR}/amber${VERSION}_src

# reuse installed python
AMBER_PYTHON=${INSTALL_DIR}/bin/amber.python

# CUDA, serial, gcc
echo "[GPU serial edition]"
mkdir build_gpu_serial && cd build_gpu_serial
cmake .. \
    -DCMAKE_INSTALL_PREFIX=${INSTALL_DIR} \
    -DCOMPILER=GNU \
    -DMPI=FALSE \
    -DCUDA=TRUE \
    -DINSTALL_TESTS=FALSE \
    -DDOWNLOAD_MINICONDA=FALSE \
    -DPYTHON_EXECUTABLE=${AMBER_PYTHON} \
    -DCUDA_TOOLKIT_ROOT_DIR=${CUDA_HOME} \
    -DFORCE_INTERNAL_LIBS="arpack" \
    -DBUILD_QUICK=TRUE \
    -DCHECK_UPDATES=FALSE

make -j${PARALLEL} install && make clean
cd ../ && rm -rf build_gpu_serial

# GPU parallel
echo "[GPU parallel edition]"
mkdir build_gpu_parallel && cd build_gpu_parallel
cmake .. \
    -DCMAKE_INSTALL_PREFIX=${INSTALL_DIR} \
    -DCOMPILER=GNU \
    -DMPI=TRUE \
    -DCUDA=TRUE \
    -DINSTALL_TESTS=FALSE \
    -DDOWNLOAD_MINICONDA=FALSE \
    -DPYTHON_EXECUTABLE=${AMBER_PYTHON} \
    -DCUDA_TOOLKIT_ROOT_DIR=${CUDA_HOME} \
    -DFORCE_INTERNAL_LIBS="arpack" \
    -DBUILD_QUICK=TRUE \
    -DCHECK_UPDATES=FALSE

make -j${PARALLEL} install && make clean
cd ../ && rm -rf build_gpu_parallel

# CPU openmp
echo "[CPU openmp edition]"
mkdir build_cpu_openmp && cd build_cpu_openmp
cmake .. \
    -DCMAKE_INSTALL_PREFIX=${INSTALL_DIR} \
    -DCOMPILER=GNU \
    -DMPI=FALSE \
    -DOPENMP=TRUE \
    -DCUDA=FALSE \
    -DINSTALL_TESTS=FALSE \
```

```
    -DDOWNLOAD_MINICONDA=FALSE \
    -DPYTHON_EXECUTABLE=${AMBER_PYTHON} \
    -DFORCE_INTERNAL_LIBS="arpack" \
    -DBUILD_REAXFF_PUREMD=TRUE \
    -DBUILD_QUICK=TRUE \
    -DCHECK_UPDATES=FALSE

make -j${PARALLEL} install && make clean
cd ../ && rm -rf build_cpu_openmp

# CPU mpi (don't build mpi+openmp version)
echo "[CPU parallel edition]"
mkdir build_cpu_parallel && cd build_cpu_parallel
cmake .. \
    -DCMAKE_INSTALL_PREFIX=${INSTALL_DIR} \
    -DCOMPILER=GNU \
    -DMPI=TRUE \
    -DOPENMP=FALSE \
    -DCUDA=FALSE \
    -DINSTALL_TESTS=FALSE \
    -DDOWNLOAD_MINICONDA=FALSE \
    -DPYTHON_EXECUTABLE=${AMBER_PYTHON} \
    -DFORCE_INTERNAL_LIBS="arpack" \
    -DBUILD_QUICK=TRUE \
    -DCHECK_UPDATES=FALSE

make -j${PARALLEL} install && make clean
cd ../ && rm -rf build_cpu_parallel

# ad hoc fix for shebang
cd ${INSTALL_DIR}/bin
for f in *; do
  grep -d skip "^#\!.*python$" $f > /dev/null
  if [ $? -eq 0 ]; then
    perm=$(stat -c "%a" $f)
    head -n 1 amber.conda >> ${f}.new
    sed -e "1d" ${f} >> ${f}.new
    mv -f ${f}.new ${f}
    chmod $perm $f
  fi
done

# run tests
cd ${INSTALL_DIR}
. ${INSTALL_DIR}/amber.sh
# now, $AMBERHOME should be $INSTALL_DIR

# parallel tests first
export DO_PARALLEL="mpirun -np 2"

make test.parallel && make clean.test

export DO_PARALLEL="mpirun -np 4"
cd test; make test.parallel.4proc; make clean; cd ../

unset DO_PARALLEL

# openmp tests
make test.openmp && make clean.test

# serial tests
make test.serial && make clean.test
```

テスト

- GPU 版のテストはフロントエンドノードでは実行できないため、以下のスクリプトで演算ノード上で実行

```
#!/bin/sh

# amber22 + AmberTools22
INSTALL_DIR="/apl/amber/22u1"

#------------------------------------------------------------------------
module -s purge
module -s load gcc-toolset/10
module -s load openmpi/4.1.4-hpcx/gcc10
module -s load cuda/11.6

export LANG=C
export LC_ALL=C
ulimit -s unlimited

# run tests
cd ${INSTALL_DIR}
. ${INSTALL_DIR}/amber.sh

make clean.test

## gpu tests
export DO_PARALLEL="mpirun -np 2"
make test.cuda.parallel && make clean.test # DPFP
cd test; ./test_amber_cuda_parallel.sh SPFP; make clean; cd ../
#
unset DO_PARALLEL
make test.cuda.serial && make clean.test # DPFP
cd test; ./test_amber_cuda_serial.sh SPFP; make clean && cd ../
```

- テスト結果は /apl/amber/22u1/logs/ に有り。基本的には軽微な数値エラー。pbsa については前回と同じ問題がある。

メモ

- JAC 系でのベンチマークは AMBER20 の方が少し性能が上。
  - 前システムでの P100/V100 時と同様の雰囲気。
- pbsa 系は色々と問題がある気配。
- 基本的には前回の内容を踏襲。