

Amber22

Webpage

<http://ambermd.org/>

Version

Amber22, AmberTools 22

Build Environment

- Intel Parallel Studio 2018 Update4 (MPI only)
- GCC 9.3.1 (devtoolset-9)
- CUDA 11.1 Update 1

Files Required

- Amber22.tar.bz2
- AmberTools22.tar.bz2
- patch-cmake-python
 - use miniforge instead of miniconda (to avoid license issue of anaconda repository)

```
--- 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}-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/${MINICONDA_VERSION}/download/${MINICONDA_INSTALLER_FILENAME}")

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

- amber.csh.patch (this file was modified after the installation; not mentioned in the procedure below)
 - The trick to get script directory does not always work. Old-style way (hard-coded) is employed this time.

```
--- amber.csh.org 2022-06-06 16:53:47.000000000 +0900
+++ amber.csh 2022-06-06 16:57:01.000000000 +0900
@@ -16,11 +16,13 @@
     echo " Your shell does not appear to be a C shell: $0"
 endif

-## Get path used for this source file (credit scott brozell).
-set invocationpath = `echo $_ | cut -d' ' -f2- | sed "s@$myname.*@@"`
-if ( "$invocationpath" == "" ) then
- set invocationpath = '.'
-endif
+## RCCS: this does not always work. Use oldstyle setting...
+set invocationpath = "/local/apl/lx/amber22-up0"
+### Get path used for this source file (credit scott brozell).
+##set invocationpath = `echo $_ | cut -d' ' -f2- | sed "s@$myname.*@@"`
```

```

+#if ( "$invocationpath" == " " ) then
+#   set invocationpath = '.'
+#endif

setenv AMBERHOME `cd "$invocationpath" >&! /dev/null; pwd`
setenv PATH "$AMBERHOME/bin:$PATH"

```

Build Procedure

```

#!/bin/sh

VERSION=22
TOOLSVERSION=22

# amber22 + AmberTools22
INSTALL_DIR="/local/apl/lx/amber22-up0"
WORKDIR="/work/users/${USER}/work-amber"
TARBALL_DIR="/home/users/${USER}/Software/AMBER/22"

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

PARALLEL=12

#-----
module purge
module load mpi/intelmpi/2018.4.274
module load scl/devtoolset-9
module load cuda/11.1
module load cmake/3.16.3

export CUDA_HOME="/local/apl/lx/cuda-11.1"

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
```

```
# 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.cond
```

```
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
make test.cuda.parallel && make clean.test # DPFP
cd test; ./test_amber_cuda_parallel.sh SPFP; make clean; cd ../

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
make test.cuda.serial && make clean.test # DPFP
cd test; ./test_amber_cuda_serial.sh SPFP; make clean && cd ../

```

Tests

- Test results are available under `/local/apl/lx/amber22-up0/logs`.
- All the tests excluding `pbsa_cuda_cg` ones (see below) seem to be passed successfully.

メモ

- Performance on P100/V100 GPUs seem to be worse than amber20 by ~5%. (tested on JAC system)
 - amber18 is the fastest for P100, amber20 is the fastest for V100
 - In the current RCCS system, you might want to use amber18/20 if you don't need new functions of amber22. (There could be some updates later, though.)
- Gcc10 (devtoolset-10) failed to build `pmemd.cuda`. Gcc7 or 8 (devtoolset-7 or devtoolset-8) is OK.
- If `cuda-11.6.1` is used, there seems to be a slight performance loss on V100 (tested on JAC system). We thus employ `cuda 11.1`.
 - Official benchmark results for P100/V100 are not yet available now (June 6, 2022).
- `arpack` is installed in our frontend nodes but not in computation nodes (`ccca*`). That's why we added `-DFORCE_INTERNAL_LIBS="arpack"` flag.
 - Generally, this flag may not be necessary. This is very specific issue to the current RCCS system.
- `quick` is enabled for all the versions, `reaxff_puremd` is enabled only for OpenMP version.
- MKL is disabled for all the versions. This is because it is difficult to use MKL only for some of versions like [the amber20-up12 case](#).
 - (Mixing MKL and non-MKL versions may be the problem. There may be some trick to overcome this...)
- `pbsa_cuda_cg` tests failed just alike `amber20-cmake (ambertools20/21)` case.
 - (reported to official ML)
 - If `pbsa.cuda` is built using `configure` script (need some modifications, though), this bug does not occur.
- Environment setting script "`amber.csh`" failed to locate script directory in some case. The script is manually modified and the path is hard-coded. (see file diff above)