Instruction Manual for Amoeba OpenMM Benchmark

Author: Lee-Ping Wang

Original version written April 10, 2013

Table of Contents:

I. How to use this document

This document will guide you through how to run the OpenMM AMOEBA benchmark that I recently set up.

```
Instructions for you to type into the command line are colored blue.
```

Occasionally I will show pieces of code, data, or input / output files. You will only need to modify something if it's shown in blue.

Special note: I ran my benchmarks using a development version of OpenMM 5.1, which provides some optimizations over OpenMM 5.0.1. I typically use development versions of OpenMM for my research, and instructions are provided for building from source. However, the current official release is OpenMM 5.0.1.

II. Installation

This chapter walks through the installation of the required software on a Linux system. I am unfamiliar with installation on other platforms.

A. Prerequisites and Dependencies

- GNU Compilers: gcc, gfortran, and g++ are required, but don't get a version that is too new or it won't work with CUDA. (Recommended version 4.4.x or 4.5.x).
- If you want to build OpenMM from source, GCCXML is needed to generate the OpenMM C and Fortran wrappers. Find the latest version at https://github.com/gccxml/gccxml (Older versions will not work.)
- Doxygen is required for building OpenMM – get the latest available version
- CMake is required for building OpenMM – get the latest available version
- Swig is required for building OpenMM – get the latest available version. This depends on PCRE but it's an easy dependency to satisfy.
- Python 2.7.3 is required for OpenMM
- NumPy is required for my OpenMM-MD script.

- Officially, CUDA 5.0 is required for the CUDA Platform (which in turn requires NVidia drivers version 304 or higher.) However, I have gotten it to work for CUDA 4.0.

B. Installing OpenMM

- OpenMM is hosted at https://simtk.org/home/openmm .
- Download OpenMM version 5.0.1 (under "Downloads") or check out the latest source code by running `svn checkout https://simtk.org/svn/openmm.`
- **Follow the instructions in the OpenMM User's Guide**, Chapter 3 (installing the binaries) or Chapter 9 (building from source code). The latest User's Guide is found at https://simtk.org/home/openmm , click on "Documents".
- Make sure the OpenMM libraries are in your `LD_LIBRARY_PATH` environment variable, using a command like this (installation location may vary):

```
export LD_LIBRARY_PATH=/usr/local/openmm/lib:$LD_LIBRARY_PATH
```

- Load the CUDA environment variables. It should look like the below five lines, although your `CUDA_HOME` environment variable might vary:

```
export CUDA_HOME=/usr/local/cuda
export PATH=$CUDA_HOME/bin:$PATH
export LD_LIBRARY_PATH=$CUDA_HOME/lib64:$CUDA_HOME/lib:$LD_LIBRARY_PATH
export INCLUDE=$CUDA_HOME/include:$INCLUDE
export OPENMM_CUDA_COMPILER=$CUDA_HOME/bin/nvcc
```

- In particular, the `OPENMM_CUDA_COMPILER` variable is needed for building CUDA kernels at runtime.

C. Installing TINKER

- TINKER is hosted at http://dasher.wustl.edu/ffe/ .
- Download the TINKER source distribution: http://dasher.wustl.edu/ffe/downloads/tinker-6.2.01.tar.gz
- TINKER is easy to install and instructions are provided in various places.
- My typical workflow for building TINKER goes something like this:

```
tar xvzf tinker-6.2.01.tar.gz
mv tinker tinker-6.2.01
cd tinker-6.2.01
mkdir bin build
cd build
ln -s ../source/* .
ln -s ../linux/gfortran/* .
./compile.make
./library.make
./link.make
./rename.make
```

- The TINKER executables should now be in the `tinker-6.2.01/bin` directory. In particular, the `OPENMM_CUDA_COMPILER` variable is needed for building CUDA kernels at runtime.

D. Installing TINKER-OpenMM Interface (Optional)

- The Tinker-OpenMM benchmark is used for testing the accuracy of OpenMM and setting up complex systems. It is not needed for the speed benchmark.
- Download the latest TINKER-OpenMM interface code from https://simtk.org/home/openmm under "Downloads", and look under "OpenMM Accelerated TINKER."
- Build the OpenMM-accelerated TINKER executable:

```
(Make sure you are still in the build directory from the previous step!)
mv dynamic.f dynamic.f.bak
tar xvzf dynamic_openmm.tar.gz
make dynamic_openmm.x
mv dynamic_openmm.x ./bin/dynamic_openmm
```

- Now you should be able to run OpenMM simulations using `dynamic_openmm` instead of `dynamic`. However, because the interface code is still in a development state, not all of the options provided in the TINKER `.key` file will be automatically passed to OpenMM.

III. Benchmark

A. About the Benchmark

Here we test the speed and accuracy of AMOEBA for various sized water boxes, ranging from 216 molecules, 648 atoms (small) to 288,000 molecules, 864,000 atoms (ocean). The DHFR system from the joint AMBER-CHARMM benchmark is also included; it contains DHFR as well as 7,023 water molecules for a total of 23,558 atoms.

All of the simulations are NPT, **0.5 fs time step** (take note), PME electrostatics with real space cutoff 0.7 Angstrom, vdW cutoff 0.9 Angstrom. Simulation speed is given in ns/day.

I picked the individual simulation lengths based on the system size; the number of steps taken for "small", "box", "big", "huge", "puddle", "lake", and "ocean" is 20000, 10000, 4000, 2000, 1000, 500, 200 respectively. Most of the simulations ran for at least 30 minutes.

With OpenMM, I ran single, mixed, and double precision simulations; single is sufficiently accurate for running our parameterization simulations for water. With TINKER I ran 1-core and 8-core simulations.

I ran benchmarks on two separate computers.

"not0rious" is a desktop with an Intel i7-3820 CPU and a NVidia GTX 580 GPU. Both OpenMM and TINKER are installed on this machine. The OpenMM simulations go up to "lake" (because the GPU runs out of memory), while the TINKER simulations only go up to "huge" (it segfaults for "puddle".)

"Stampede" is a compute node on the Stampede supercomputer with NVidia Kepler K20m GPUs. TINKER is not installed on this machine. The OpenMM simulations go up to "ocean" because the Kepler GPU has more memory.

B. Downloading Files

The benchmark is hosted at https://simtk.org/home/openmm-amoeba .

The main download (`OpenMM-AMOEBA-Benchmark.tar.gz`, under "Downloads") contains the input files and scripts needed for running the benchmark, along with two sets of outputs.

The benchmark results and the latest version of this document can be found under "Documents". If you want to upload anything, please create a Simtk.org account and let me know, so I can give you permissions to edit the page.

C. Running the speed benchmark

By now, OpenMM and TINKER should both be ready to go. The speed benchmark runs OpenMM and compares the speed with TINKER (i.e. it does not use the TINKER-OpenMM interface, but you should get similar results if you choose to do it this way).

Extract the archive:

```
tar xvzf OpenMM-AMOEBA-Benchmark.tar.gz
```

In the `scripts-not0rious` and `scripts-stampede` directory, the scripts `run-dhfr.sh` and `run-water.sh` are provided. You will need to edit the script so that it knows the location of the TINKER `dynamic` executable on your computer. Running the script will perform the benchmark calculations.

```
cp scripts-stampede/run-water.sh .
(edit the script to conform to your environment, and select which
systems / methods / platforms you want to test.)
./run-water.sh | tee run-water.out
(now wait for the benchmark calculations to complete.)
```

The simulation results will be written into the `Output` directory. When the calculations are done, the `extract-water.sh` and `extract-dhfr.sh` scripts will print out the data in a format that you can paste into Excel. See the `AMOEBA_Speed.xlsx` spreadsheet on the website for more details.

D. Running the accuracy benchmark

To run the accuracy benchmark, you need to recompile the dynamic_openmm executable. Specifically, the lines you need to change are:

```
(Line 351 of dynamic.f.  By enabling applyOpenMMTest, dynamic_openmm
will not run any molecular dynamics, but instead compare energies and
forces with native TINKER.)
      applyOpenMMTest = 1
c     applyOpenMMTest = 0
```

```
(Line 1272 of dynamic_openmm.cpp; Change “single” to “double” if
desired)
    OpenMM_Platform_setPropertyDefaultValue( platform,
"CudaPrecision", "single" );
```

Rebuild dynamic_openmm following the instructions in the section II.D.

I didn’t write a script for automatically doing the accuracy benchmark, but this calculation is easy to run if you’ve already done the speed benchmark, because the simulations are all set up.

Go into one of the output directories, and set up a TINKER .key file for the test calculation:

```
emacs water1.key
cp water1.key water-test.key
echo "vdw-correction" >> water-test.key
echo "integrator verlet" >> water-test.key
```

```
~/opt/tinker-6.1.01-openmm/build_openmm/dynamic_openmm.x water -k
water-test 1 1 1 1
```

The extra lines are necessary because OpenMM automatically applies the long range vdW correction, and the default integrator (Beeman) is not supported.

TINKER will start up and calculate the energy difference between TINKER and OpenMM, then generate output like the lines below. The highlighted lines show the difference between OpenMM and TINKER; this is explained in more detail in the `AMOEBA_Accuracy.xlsx` spreadsheet file.

```
     #############################################################################
   #################################################################################
  ###                                                                           ###
 ###            TINKER  ---  Software Tools for Molecular Design               ###
 ##                                                                             ##
 ##                           Version 6.1  June 2012                            ##
 ##                                                                             ##
 ##               Copyright (c)  Jay William Ponder  1990-2012                  ##
 ###                           All Rights Reserved                             ###
  ###                                                                           ###
   #################################################################################
     #############################################################################


 Molecular Dynamics Trajectory via Velocity Verlet Algorithm
Testing mode is active (=1).
Testing
Default OpenMM plugin directory: /home/leeping/opt/openmm/lib/plugins
Plugin lib: libOpenMMOpenCL.so
Plugin lib: libOpenMMRPMDCUDA.so
Plugin lib: libOpenMMAmoebaSerialization.so
Plugin lib: libOpenMMRPMDOpenCL.so
Plugin lib: libOpenMMCUDA.so
Plugin lib: libOpenMMAmoebaCUDA.so
System:                             number of particles: 648
AmoebaBondForce:                 number of bonds: 432
AmoebaAngleForce:                number of angles: 216
AmoebaInPlaneAngleForce:         number of angles: 0
PeriodicTorsionForce:                    number of torsions: 0
AmoebaPiTorsionForce:                    number of angles: 0
AmoebaStretchBendForce:           number of stretch bends: 0
AmoebaOutOfPlaneBend:             number of out-of-plane bends: 0
AmoebaTorsionTorsionForce: number of torsions-torsions: 0
AmoebaVdwForce
                                    number of particles: 648
                                  sigma combining rule: CUBIC-MEAN
                                epsilon combining rule: HHG
                                               cutoff:   9.0000000e-01 nm
                                                  PBC: 1

LPW: The polarization type has been set to 1
AmoebaMultipoleForce
                                  number of particles: 648
                                     nonbonded method: 1 (NoCutoff=0, PME=1)
                                   polarization type: 1 (Mutual=0,   Direct=1)
                                     cutoff distance:   7.0000000e-01 nm
                                              aEwald:   5.4459052e+00
                                Ewald error tolerance:   1.0000000e-04
                                     PME spline order: 5
```

```
                                    grid dimensions: [24 24 24]
                                Mutual induced epsilon:   1.0000000e-06
                            Mutual induced max iterations: 500

HarmonicBondForce (Urey-Bradley):   number of ixns: 216
Use Vdw neighbor list=0 Vdw cutoff=       9.000
Energies: check sum -1.9872001e+03
          esum= -1.9872001e+03   eb=  1.6959968e+02   ea=  6.1019392e+01
           eba=  0.0000000e+00  eub= -5.0689933e+00   eaa=  0.0000000e+00
          eopb=  0.0000000e+00 eopd=  0.0000000e+00   eid=  0.0000000e+00
           eit=  0.0000000e+00   et=  0.0000000e+00   ept=  0.0000000e+00
           ebt=  0.0000000e+00  ett=  0.0000000e+00   ev=  5.7376500e+02
            ec=  0.0000000e+00  ecd=  0.0000000e+00   ed=  0.0000000e+00
            em= -2.0943676e+03   ep= -6.9214756e+02   er=  0.0000000e+00
            es=  0.0000000e+00  elf=  0.0000000e+00   eg=  0.0000000e+00
            ex=  0.0000000e+00

use: bond=1 angle=1 tors= 0 pitors=1 strbnd=0 opbend=0 tortor=0 solv=0 mpole=1 vdw=1 urey=1 iSolv=0
AmoebaAllTest
Platfrom CUDA: using default value for CUDA device id.
Test AmoebaAllTest conversion=       -0.02390
Box(nm): [       1.864        0.000        0.000] [       0.000        1.864        0.000] [       0.000
0.000        1.864]
Potential energy: Absolute difference= -3.8004092e-04 Relative difference=  1.9124439e-07 Tinker= -
1.9872001e+03 OpenMM= -1.9872005e+03 AmoebaAllTest
 Index   TinkerNorm   OpenMMNorm     RelDiff     AbsDiff     Dot
Tinker Forces                                  OpenMM Forces
MaxDelta    = Maximum Force Difference in kcal/Angstrom
MaxRelDelta = Maximum Fractional Force Difference
MaxRelNorm  = Norm of Force for MaxRelDelta
AvgRelDelta = Average Fractional Force Difference
MaxNorm     = Maximum Norm of Force
MinDot      = Minimum Force Dot-Product
AvgDot      = Average Force Dot-Product
MaxDelta at       198   2.2224667e-03 AmoebaAllTest
MaxRelDelta at    334   3.0714689e-04 AmoebaAllTest
MaxRelNorm  at    334   3.1838745e+00 AmoebaAllTest
AvgRelDelta at          2.2569091e-05 AmoebaAllTest
MaxNorm     at    579   1.1433768e+02   1.1433827e+02 AmoebaAllTest
MinDot at         334   9.9999995e-01 AmoebaAllTest
AvgDot at               1.0000000e+00 AmoebaAllTest
Enclosing Box (in A): [ -1.0068535e+01   9.8891550e+00] [ -9.9745020e+00   9.8552780e+00] [ -9.9528360e+00
9.7721820e+00]   [  1.9957690e+01   1.9829780e+01   1.9725018e+01]
```