innovative science • intuitive software

RDKit OpenMM integration

Paolo Tosco

6th RDKit UGM, 20th September 2017

Outline

- Background

- Implementation

- Results

- Conclusions and outlook

Background

Background

> OpenMM

- > High performance toolkit for molecular simulation
- > Python, C, C++, Fortran (!) bindings
- > Open source
- > Actively maintained on GitHub
- > Licensed under MIT and LGPL

> http://openmm.org

OpenMM

About OpenMM

Backed by researchers and developers from Stanford University, MSKCC, and others around the world.

Custom Forces

Want a custom force between two atoms? No problem. Write your force expressions in string format, and OpenMM will generate blazing fast code to do just that. No more hand-writing GPU kernels.

Highly Optimized

OpenMM is optimized for the latest generation of compute hardware, including AMD (via OpenCL) and NVIDIA (via CUDA) GPUs. We also heavily optimize for CPUs using intrinsics.

Portable

We strive to make our binaries as portable as possible. We've tested OpenMM on many flavors of Linux, OS X, and even Windows.

What about…

> …spicing up the RDKit MM simulations with OpenMM?

- > Licenses are compatible
- > So are APIs
- > Initial proof-of-concept on MMFF94

Proof-of-concept MMFF94 implementation

Warning: content may offend

cresset software

Implementing MMFF94 terms into OpenMM (somehow)

$E_{MMFF} = \sum EB_{ij}$	bond stretching	$EB_{ij} = 143.9325 \frac{kb_{ij}}{2} \Delta r_{ij}^2 \left(1 + cs\Delta r_{ij} + \frac{7}{12} cs^2 \Delta r_{ij}^2\right)$	`OpenMM::CustomBondForce`
$+ \sum EA_{ijk}$	angle bending	$EA_{ijk} = 0.043844 \frac{ka_{ijk}}{2} \Delta\vartheta_{ijk}^2 \left(1 + cb\Delta\vartheta_{ijk}\right)$	`OpenMM::CustomAngleForce`
$+ \sum EBA_{ijk}$	stretch-bend	$EBA_{ijk} = 2.51210\left(kba_{ijk}\Delta r_{ij} + kba_{kji}\Delta r_{kj}\right)\Delta\vartheta_{ijk}$	`OpenMM::AmoebaStretchBendForce`
$+ \sum EOOP_{ijk:l}$	out-of-plane bending	$EOOP_{ijk:l} = 0.043844 \frac{koop_{ijk:l}}{2} \chi_{ijk:l}^2$	`OpenMM::AmoebaOutOfPlaneBendForce`
$+ \sum ET_{ijkl}$	torsion	$ET_{ijkl} = 0.5\left[V_1\left(1 + \cos\phi\right) + V_2\left(1 - \cos 2\phi\right) + V_3\left(1 + \cos 3\phi\right)\right]$	`OpenMM::CustomTorsionForce`
$+ \sum EvdW_{ij}$	van der Waals	$EvdW_{ij} = \varepsilon_{ij}\left(\frac{1.07R_{ij}^*}{R_{ij} + 0.07R_{ij}^*}\right)^7 \left(\frac{1.12R_{ij}^{*7}}{R_{ij}^7 + 0.12R_{ij}^{*7}} - 2\right)$	`modified` `OpenMM::AmoebaVdwForce`
$+ \sum EQ_{ij}$	electrostatic	$EQ_{ij} = 332.0716 \frac{q_i q_j}{D\left(R_{ij} + \delta\right)^n}$	`OpenMM::CustomNonbondedForce`

cresset software

(AMOEBA + MMFF94)^{hurry} = BIG HACK!

- I hacked the `OpenMM::AmoebaVdwForce` implementation adding a "MMFF" combination rule
- I abused the AMOEBA per-particle σ_I, ε_I and reduction (not used by MMFF94) parameters to pass R_I, $G_I a_I$ and a_I / N_I instead, using sign combinations to encode HBA/HBD features
- I split in two the electrostatic term making heavy use of exclusions to implement scaled 1,4 electrostatics
- As a result, there are three non-bonded terms where there should ideally be one

(AMOEBA + MMFF94)^{hurry} = BIG HACK!

> I hacked the `OpenMM::AmoebaVdwForce` implementation adding a "MMFF" combination rule

> I abused the AMOEBA per-particle σ_i scaling factor (not used by MMFF94) parameters to pass R_i^* via a_i and a_i/N_i instead, using sign combinations to encode HBA/HBD features

> I split in two the electrostatic term making heavy use of exclusions to implement scaled 1,4 electrostatics

> As a result, there are three non-bonded terms where there should ideally be one

DISGUSTING

But: 150-fold speedup compared to native RDKit MMFF94!

cresset software

Proper MMFF94 implementation

OpenMM side, C++

OpenMM implementation of MMFF94 Forces (really, I mean it)

$$E_{MMFF} = \sum EB_{ij}$$ bond stretching $$EB_{ij} = 143.9325 \frac{kb_{ij}}{2} \Delta r_{ij}^2 \left(1 + cs\Delta r_{ij} + \frac{7}{12} cs^2 \Delta r_{ij}^2\right)$$ `OpenMM::MMFFBondForce`

$$+ \sum EA_{ijk}$$ angle bending $$EA_{ijk} = 0.043844 \frac{ka_{ijk}}{2} \Delta\vartheta_{ijk}^2 \left(1 + cb\Delta\vartheta_{ijk}\right)$$ `OpenMM::MMFFAngleForce`

$$+ \sum EBA_{ijk}$$ stretch-bend $$EBA_{ijk} = 2.51210\left(kba_{ijk}\Delta r_{ij} + kba_{kji}\Delta r_{kj}\right)\Delta\vartheta_{ijk}$$ `OpenMM::MMFFStretchBendForce`

$$+ \sum EOOP_{ijk:l}$$ out-of-plane bending $$EOOP_{ijk:l} = 0.043844 \frac{koop_{ijk:l}}{2} \chi_{ijk:l}^2$$ `OpenMM::MMFFOutOfPlaneBendForce`

$$+ \sum ET_{ijkl}$$ torsion $$ET_{ijkl} = 0.5\left[V_1\left(1 + \cos\phi\right) + V_2\left(1 - \cos 2\phi\right) + V_3\left(1 + \cos 3\phi\right)\right]$$ `OpenMM::MMFFTorsionForce`

$$+ \sum EvdW_{ij}$$ van der Waals $$EvdW_{ij} = \varepsilon_{ij} \left(\frac{1.07R_{ij}^*}{R_{ij} + 0.07R_{ij}^*}\right)^7 \left(\frac{1.12R_{ij}^{*7}}{R_{ij}^7 + 0.12R_{ij}^{*7}} - 2\right)$$ `OpenMM::MMFFNonbondedForce`

$$+ \sum EQ_{ij}$$ electrostatic $$EQ_{ij} = 332.0716 \frac{q_i q_j}{D\left(R_{ij} + \delta\right)^n}$$

cresset software

That took a fair bit of work…

Directory Tree

```
openmm/plugins/mmff
|-- CMakeLists.txt
|-- openmmapi
| |-- include
| | |-- OpenMMMMFF.h
| | `-- openmm
| |     |-- MMFFAngleForce.h
| |     |-- MMFFBondForce.h
| |     |-- MMFFNonbondedForce.h
| |     |-- MMFFOutOfPlaneBendForce.h
| |     |-- MMFFStretchBendForce.h
| |     |-- MMFFTorsionForce.h
| |     |-- internal
| |     | |-- MMFFAngleForceImpl.h
| |     | |-- MMFFBondForceImpl.h
| |     | |-- MMFFNonbondedForceImpl.h
| |     | |-- MMFFOutOfPlaneBendForceImpl.h
| |     | |-- MMFFStretchBendForceImpl.h
| |     | |-- MMFFTorsionForceImpl.h
| |     | `-- windowsExportMMFF.h
| |     `-- mmffKernels.h
| `-- src
|     |-- MMFFAngleForce.cpp
|     |-- MMFFAngleForceImpl.cpp
|     |-- MMFFBondForce.cpp
|     |-- MMFFBondForceImpl.cpp
|     |-- MMFFNonbondedForce.cpp
|     |-- MMFFNonbondedForceImpl.cpp
|     |-- MMFFOutOfPlaneBendForce.cpp
|     |-- MMFFOutOfPlaneBendForceImpl.cpp
|     |-- MMFFStretchBendForce.cpp
|     |-- MMFFStretchBendForceImpl.cpp
|     |-- MMFFTorsionForce.cpp
|     `-- MMFFTorsionForceImpl.cpp
|-- platforms
| |-- cuda
| | |-- CMakeLists.txt
| | |-- include
| | | `-- MMFFCudaKernelFactory.h
| | |-- src
| | | |-- CudaMMFFKernelSources.cpp.in
| | | |-- CudaMMFFKernelSources.h.in
| | | |-- MMFFCudaKernelFactory.cpp
| | | |-- MMFFCudaKernels.cpp
| | | |-- MMFFCudaKernels.h
| | | `-- kernels
| | |     |-- mmffAngleForce.cu
| | |     |-- mmffBondForce.cu
| | |     |-- mmffNonbonded.cu
| | |     |-- mmffNonbondedExceptions.cu
| | |     |-- mmffOutOfPlaneBendForce.cu
| | |     |-- mmffStretchBendForce.cu
| | |     `-- mmffTorsionForce.cu
| | `-- tests
| |     |-- CMakeLists.txt
| |     |-- CudaTests.h
| |     |-- TestCudaMMFFAngleForce.cpp
| |     |-- TestCudaMMFFBondForce.cpp
| |     |-- TestCudaMMFFNonbondedForce.cpp
| |     |-- TestCudaMMFFOutOfPlaneBendForce.cpp
| |     |-- TestCudaMMFFStretchBendForce.cpp
| |     `-- TestCudaMMFFTorsionForce.cpp
| `-- reference
|     |-- CMakeLists.txt
|     |-- include
|     | |-- MMFFReferenceKernelFactory.h
|     | `-- windowsExportMMFFReference.h
|     |-- src
|     | |-- MMFFReferenceKernelFactory.cpp
|     | |-- MMFFReferenceKernels.cpp
|     | |-- MMFFReferenceKernels.h
|     | `-- SimTKReference
|     |     |-- MMFFReferenceAngleForce.cpp
|     |     |-- MMFFReferenceAngleForce.h
|     |     |-- MMFFReferenceBondForce.cpp
|     |     |-- MMFFReferenceBondForce.h
|     |     |-- MMFFReferenceForce.cpp
|     |     |-- MMFFReferenceForce.h
|     |     |-- MMFFReferenceNonbondedForce.cpp
|     |     |-- MMFFReferenceNonbondedForce.h
|     |     |-- MMFFReferenceNonbondedForce14.cpp
|     |     |-- MMFFReferenceNonbondedForce14.h
|     |     |-- MMFFReferenceOutOfPlaneBendForce.cpp
|     |     |-- MMFFReferenceOutOfPlaneBendForce.h
|     |     |-- MMFFReferenceStretchBendForce.cpp
|     |     |-- MMFFReferenceStretchBendForce.h
|     |     |-- MMFFReferenceTorsionForce.cpp
|     |     `-- MMFFReferenceTorsionForce.h
|     `-- tests
|         |-- CMakeLists.txt
|         |-- ReferenceTests.h
|         |-- TestReferenceMMFFAngleForce.cpp
|         |-- TestReferenceMMFFBondForce.cpp
|         |-- TestReferenceMMFFNonbondedForce.cpp
|         |-- TestReferenceMMFFOutOfPlaneBendForce.cpp
|         |-- TestReferenceMMFFStretchBendForce.cpp
|         `-- TestReferenceMMFFTorsionForce.cpp
|-- serialization
| |-- include
| | `-- openmm
| |     `-- serialization
| |         |-- MMFFAngleForceProxy.h
| |         |-- MMFFBondForceProxy.h
| |         |-- MMFFNonbondedForceProxy.h
| |         |-- MMFFOutOfPlaneBendForceProxy.h
| |         |-- MMFFStretchBendForceProxy.h
| |         `-- MMFFTorsionForceProxy.h
| |-- src
| | |-- MMFFAngleForceProxy.cpp
| | |-- MMFFBondForceProxy.cpp
| | |-- MMFFNonbondedForceProxy.cpp
| | |-- MMFFOutOfPlaneBendForceProxy.cpp
| | |-- MMFFSerializationProxyRegistration.cpp
| | |-- MMFFStretchBendForceProxy.cpp
| | `-- MMFFTorsionForceProxy.cpp
| `-- tests
|     |-- CMakeLists.txt
|     |-- TestSerializeMMFFAngleForce.cpp
|     |-- TestSerializeMMFFBondForce.cpp
|     |-- TestSerializeMMFFNonbondedForce.cpp
|     |-- TestSerializeMMFFOutOfPlaneBendForce.cpp
|     |-- TestSerializeMMFFStretchBendForce.cpp
|     `-- TestSerializeMMFFTorsionForce.cpp
`-- wrappers
    |-- CMakeLists.txt
    |-- Doxyfile.in
    `-- generateMMFFWrappers.py

23 directories, 103 files
```

cresset software

Implementation

RDKit side, C++

The `OpenMMForceField` class, or where the magic lies

The core of the new OpenMM-powered force field implementation is the `ForceFields::OpenMMForceField` class

```
class OpenMMForceField : public ForceField {
  public:
    [...]
    OpenMM::System *getSystem() const;
    OpenMM::Context *getContext() const;
    OpenMM::Integrator *getIntegrator() const
    void initializeContext();
    void initializeContext(const std::string& platformName, const std::map<std::string, std::string> &prop);
    void initializeContext(OpenMM::Platform& platform, const std::map<std::string, std::string> &prop);
    double calcEnergy(std::vector<double> *contribs = NULL) const;
    double calcEnergy(double *pos);
    void calcGrad(double *forces) const;
    void calcGrad(double *pos, double *forces);
    int minimize(unsigned int maxIts = 200, double forceTol = 1e-4, double energyTol = 1e-6);
    virtual void cloneSystemTo(OpenMM::System& other) const;
    void copyPositionsTo(OpenMM::Context& other) const;
    void copyPositionsFrom(const OpenMM::Context& other);
  protected:
    [...]
  private:
    [...]
}
```

Getters for `OpenMM::System`, `Platform` and `Integrator`

OpenMM-enabled re-implementations of the base class methods

Facilitate usage of the native OpenMM API for expert users (particularly from Python)

cresset software

The `MMFF::OpenMMForceField` class, where more magic lies

The MMFF94-specific machinery is hosted by the `MMFF::OpenMMForceField` class

```
class OpenMMForceField : public ForceField {
  public:
    [...]
    void addBondStretchContrib(...);
    void addAngleBendContrib(...);
    void addStretchBendContrib(...);
    void addTorsionAngleContrib(...);
    void addOopBendContrib(...);
    void addNonbondedContrib(...);
const std::vector<std::string>& loadedPlugins();
    const std::vector<std::string>& failedPlugins();
  protected:
    [...]
  private:
    [...]
}
```

Get me an OpenMM-enabled force field, now!

To construct an OpenMM-enabled force field, all you need is call the familiar

```
OpenMMForceField *constructOpenMMForceField(ROMol &mol,
  MMFFMolProperties *mmffMolProperties, double nonBondedThresh = 100.0,
  int confId = -1, bool ignoreInterfragInteractions = true);
```

The only difference from the well-known call are those two tiny `OpenMM` prefixes

- Once you have created it, you may do the usual things:
  - Calculate the potential energy: `ff->calcEnergy()`
  - Run a minimization: `ff->minimize()`
- Or more exotic ones, through the native OpenMM C++ API:
  - Run *n* steps of MD

Implementation

RDKit side, Python

I already know how to do this, don't I?

Also in Python, you shouldn't be too surprised by the new API:

```
MMFFGetMoleculeOpenMMForceField( (Mol)mol, (object)pyMMFFMolProperties [,
  (float)nonBondedThresh = 100.0 [, (int)confid = -1 [,
  (bool)ignoreInterfragInteractions = True]]])
```

Again, you'll only need to add a tiny `OpenMM` to your existing scripts

- Once you have created it, you may do the usual things:
  - Calculate the potential energy: `ff.CalcEnergy()`
  - Run a minimization: `ff.Minimize()`
- Or more exotic ones, through the native OpenMM Python API:
  - Run *n* steps of MD

Results

On to the Jupyter notebook

Conclusions and outlook

There’s always more work to do

Conclusions

- The OpenMM implementation of MMFF94 within the RDKit
  - delivers impressive performance even on consumer GPU hardware
  - enables fast molecular mechanics and molecular dynamics simulations on medium/large systems
  - Can be accessed with minimal modifications to old scripts

Outlook

- Before I can get all this to you I need:
  - to write code at least for the CPU platform in addition to CUDA
  - to write OpenMM unit tests for all MMFF94 force field terms for all supported platforms
  - to submit a pull request with the MMFF94 plugin to the OpenMM developers

cresset®
software

innovative science • intuitive software

Thank you for your attention

paolo@cresset-group.com

cressetgroup