OpenMM: GPU Accelerated Algorithm Development for Molecular Dynamics

Peter Eastman
GPU Technology Conference, March 26, 2014
OpenMM is...

A. An application for running molecular simulations
B. A library of simulation routines for use by applications
C. A domain specific language for molecular simulation
D. All of the above
Why?

• Writing MD code is hard!
• Writing GPU code is hard!
• Very active field of research
  – Lots of new algorithms being developed
  – Lots of people want to use them

• Make it easy for *scientists* to code new algorithms
The OpenMM Architecture

- **High Level Python Code**
  - Application Layer
- **Public Interface**
  - OpenMM Public API
- **Platform Independent Code**
  - Implementation Layer
- **Platform Abstraction Layer**
  - OpenMM Low Level API
- **Computational Kernels**
  - CUDA/OpenCL/MPI/etc.
Example Script

```python
from simtk.openmm.app import *
from simtk.openmm import *
from simtk.unit import *

pdb = PDBFile('input.pdb')
forcefield = ForceField('amber99sb.xml', 'tip3p.xml')
system = forcefield.createSystem(pdb.topology, nonbondedMethod=PME,
                                  nonbondedCutoff=1*nanometer, constraints=HBonds)
integrator = LangevinIntegrator(300*kelvin, 1/picosecond,
                                 0.002*picoseconds)
integration = Simulation(pdb.topology, system, integrator)
integration.context.setPositions(pdb.positions)
integration.minimizeEnergy()
integration.reporters.append(PDBReporter('output.pdb', 1000))
integration.step(100000)
```
You Can Do Anything

• Simulated annealing in three lines:

```python
for i in range(100):
    integrator.setTemperature(3*(100-i)*kelvin)
simulation.step(1000)
```
Creating Kernels

• Sometimes you need to create new kernels
  – New force types
  – New integration algorithms

• OpenMM can create them for you
  – The user specifies *equations*
  – We generate the kernels to implement them
  – No knowledge of GPU programming required!
Custom Forces

• Support arbitrary force kernels
  – The user specifies the energy with an algebraic expression
  – OpenMM does everything else!

```python
force = CustomBondForce("D*(1-exp(a*(r0-r)))^2")
force.addPerBondParameter("D")
force.addPerBondParameter("a")
force.addPerBondParameter("r0")
```
How It Works

1. Parse the expression:

\[ E(r) = D \left( 1 - e^{a(r_0 - r)} \right)^2 \]

2. Analytically differentiate it:

\[ F(r) = -2aD \left( 1 - e^{a(r_0 - r)} \right) e^{a(r_0 - r)} \]
3. Generate CUDA or OpenCL code to evaluate force and energy

```plaintext
real temp5 = 1.00000000e+00f;
real temp6 = bondParams1.z-r;
real temp7 = bondParams1.y*temp6;
real temp8 = EXP(temp7);
real temp9 = temp5-temp8;
real temp10 = temp9*temp9;
real temp11 = bondParams1.x*temp10;
energy += temp11;
real temp12 = -2.00000000e+00f*temp9;
real temp13 = -1.00000000e+00f*bondParams1.y;
real temp14 = temp8*temp13;
real temp15 = temp12*temp14;
real temp16 = bondParams1.x*temp15;
real dEdR = temp16;
```
How It Works (continued)

4. Insert it into a kernel (based on hardware, force type, etc.)

5. Compile it

- This happens at runtime, takes < 1 second
- Runs at full speed on the GPU
Custom Forces Classes

• CustomBondForce
  – Bonded force between two particles
  – Energy is a function of the distance

• CustomAngleForce
  – Bonded force between three particles
  – Energy is a function of the angle

• CustomTorsionForce
  – Bonded force between four particles
  – Energy is a function of the dihedral angle
Custom Forces Classes (continued)

- **CustomExternalForce**
  - Force applied to each particle independently
  - Energy is a function of the particle position

- **CustomNonbondedForce**
  - Nonbonded force between pairs of particles
  - Energy is a function of the distance
Custom Forces Classes (continued)

- CustomCompoundBondForce
  - Bonded force between an arbitrary number of particles
  - Energy can depend on positions, distances, angles, and dihedrals
Custom Forces Classes (continued)

• CustomGBForce
  – Supports various implicit solvent models

• CustomHbondForce
  – Supports various hydrogen bonding models
Example: OBC-II Implicit Solvent

```python
force = CustomGBForce()
force->addPerParticleParameter("q")
force->addPerParticleParameter("radius")
force->addPerParticleParameter("scale")
force->addGlobalParameter("solventDielectric", 78.3)
force->addGlobalParameter("soluteDielectric", 1.0)
force->addComputedValue("I", "
    step(r+sr2-or1)*0.5*(1/L-1/U+0.25*(1/U^2-1/L^2)*(r-sr2*sr2/r)+0.5*log(L/U)/r+C);
    U=r+sr2; C=2*(1/or1-1/L)*step(sr2-r-or1); L=max(or1, D); D=abs(r-sr2);
    sr2 = scale2*or2; or1 = radius1-0.009; or2 = radius2-0.009""
    CustomGBForce.ParticlePairNoExclusions)
force->addComputedValue("B", ""1/(1/or-tanh(1*psi-0.8*psi^2+4.85*psi^3)/radius);
    psi=I*or; or=radius-0.009""", CustomGBForce.SingleParticle)
force->addEnergyTerm(""28.3919551*(radius+0.14)^2*(radius/B)^6-0.5*138.935456*
    (1/soluteDielectric-1/solventDielectric)*q^2/B""", CustomGBForce.SingleParticle)
force->addEnergyTerm(""-138.935456*(1/soluteDielectric-1/solventDielectric)*q1*q2/f;
```

OpenMM implements 5 implicit solvent models with ~200 lines of Python code!
Custom Integrators

• Define integration algorithm as an arbitrary series of computations

• Supports many types of integrators
  – Deterministic
  – Stochastic
  – Metropolized
  – Generalized Langevin
  – Multiple time step
  – ...

Example: Velocity Verlet

```python
integrator = CustomIntegrator(0.001)
integrator.addComputePerDof("v", "v+0.5*dt*f/m")
integrator.addComputePerDof("x", "x+dt*v")
integrator.addComputePerDof("v", "v+0.5*dt*f/m")
```

- CustomIntegrator automatically recalculates forces when necessary
- Supports arbitrary global and per-DOF variables
Example: aMD

class DualAMDIIntegrator(CustomIntegrator):
    def __init__(self, dt, group, alphaTotal, ETotal, alphaGroup, EGroup):
        CustomIntegrator.__init__(self, dt)
        self.addGlobalVariable("alphaTotal", alphaTotal)
        self.addGlobalVariable("ETotal", ETotal)
        self.addGlobalVariable("alphaGroup", alphaGroup)
        self.addGlobalVariable("EGroup", EGroup)
        self.addGlobalVariable("groupEnergy", 0)
        self.addPerDofVariable("oldx", 0)
        self.addPerDofVariable("fg", 0)
        self.addUpdateContextState();
        self.addComputeGlobal("groupEnergy", "energy"+str(group))
        self.addComputePerDof("fg", "f"+str(group))
        self.addComputePerDof("v", "v+dt*fprime/m; fprime=fprime1 + fprime2;"
                          "fprime2=fg*(1-modifyGroup)+modifyGroup*(alphaGroup/(alphaGroup+EGroup-groupEnergy))^2);"
                          "fprime1=other*((1-modifyTotal)+modifyTotal*(alphaTotal/(alphaTotal+ETotal-energy))\^2);"
                          "other=f-fg; modifyTotal=step(ETotal-energy); modifyGroup=step(EGroup-groupEnergy)""
                          )
        self.addComputePerDof("oldx", "x")
        self.addComputePerDof("x", "x+dt*v")
        self.addConstrainPositions()
        self.addComputePerDof("v", "(x-oldx)/dt")
Summary

GPU Programming for Normal People!

• Use a *high level scripting language* to define *equations*
• OpenMM transforms them into optimized GPU code
• This happens *at runtime*
• Everything runs at full speed on the GPU
You Can Do This Too

- OpenMM is open source!
  - The relevant code is easy to extract/reuse
    - Lepton library: parse, differentiate, optimize mathematical expressions
    - CudaExpressionUtilities, OpenCLEExpressionUtilities: generate source code to evaluate expressions

- For more information: