PRACE PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE

NB-LIB: A Performant API for Force Calculations

11 March 2021

NBLIB Team members

Joe Jordan
PDC

Sebastian Keller
CSCS

Prashanth
Kanduri
CSCS

Victor Holanda
CSCS

Artem Zhmurov
PDC

Outline

- **What is NB-LIB**
- Background
- Implementation
- Example workflows

Objectives

1. **Implement an API that permits existing performance-portable GROMACS non-bonded force calculation routines to be called as a library.**
2. Deploy bindings for the non-bonded API in C++17 and Python 3.
3. Make resulting code available under business-friendly free and open-source licenses.
4. Progressively integrate code with GROMACS development master branch and deploy the API to the library alongside the existing annual releases.

Current and upcoming machines

Machine	CPU	GPU
Fugaku	ARM	No
Summit	IBM Power9	NVIDIA
Piz Daint	Intel	NVIDIA
Lumi*	AMD	AMD

*Lumi not deployed yet

Outline

- What is NB-LIB
- **Background**
- Implementation
- Example workflows

Molecular Dynamics (MD)

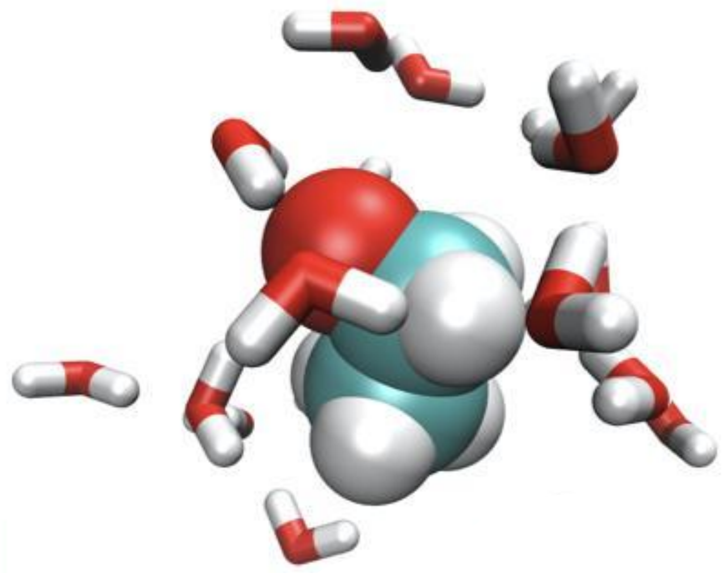
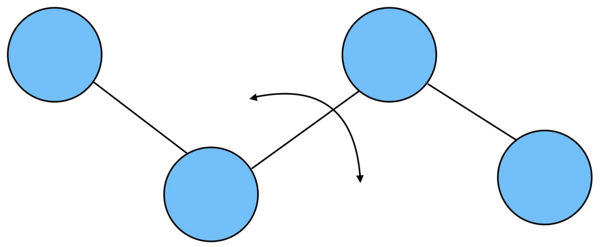
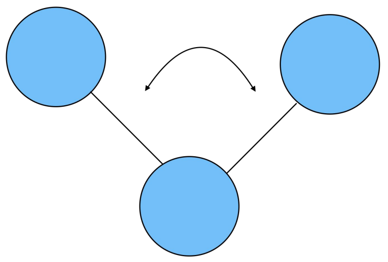


Image by Berk Hess & Magnus Lundborg

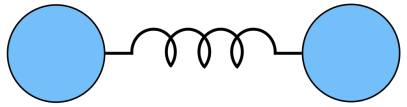
Types of forces in MD



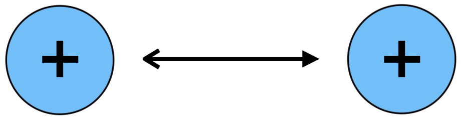
Torsion Rotation



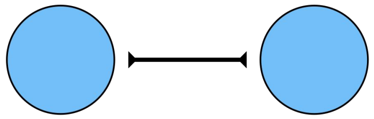
Angle Vibration



Bond Vibration



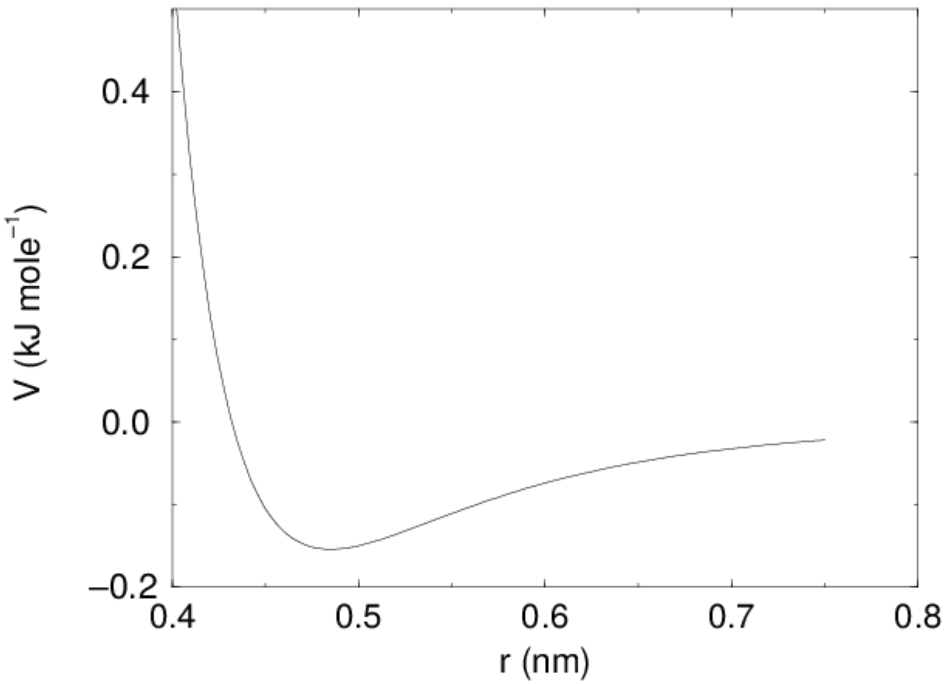
Electrostatic
Repulsion



VdW
Attraction

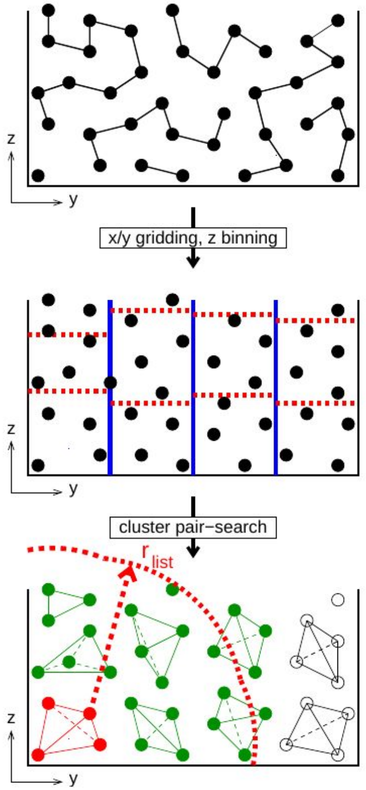
Lennard-Jones interactions

$$V_{LJ}(r_{ij}) = \frac{C_{ij}^{(12)}}{{r_{ij}}^{12}} - \frac{C_{ij}^{(6)}}{{r_{ij}}^{6}}$$



From the gromacs manual

Pair interaction calculation



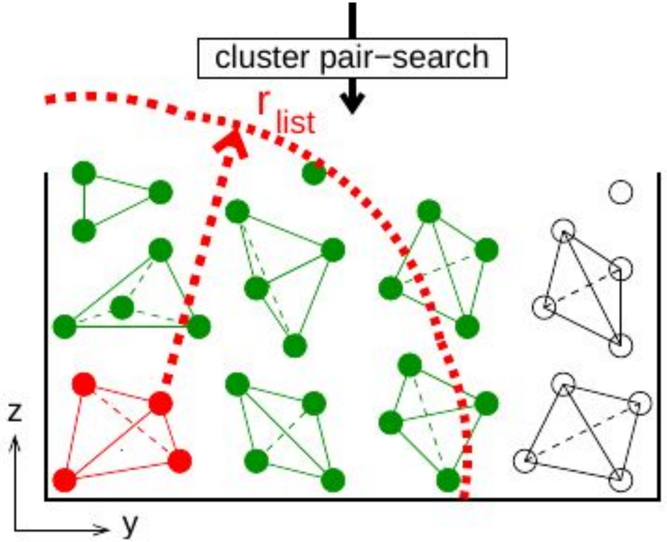


Figure 2: Illustration of the cluster pair-search algorithm for clusters of 4 particles. The bottom figure shows the j-cluster list in green for the red i-cluster.

Páll, Szilárd & Hess, Berk. (2013). Computer Physics Communications. 184.

Outline

- What is NB-LIB
- Background
- **Implementation**
- Example workflows

Particles are the smallest building blocks

```
// A Particle has a name and a mass
ParticleType Ow(ParticleName("Ow"), Mass(15.99940));
ParticleType H(ParticleName("H"), Mass(1.008));
```

Particles are the smallest building blocks

```
// A Particle has a name and a mass
ParticleType Ow(ParticleName("Ow"), Mass(15.99940));
ParticleType H(ParticleName("H"), Mass(1.008));

ParticleType OMet(ParticleName("OMet"), Mass(15.999));
ParticleType CMet(ParticleName("CMet"), Mass(15.035));
```

Non-bonded interactions are flexible

```
// A combination rule can be specified for non-self interactions
ParticleTypesInteractions interactions(CombinationRule::Geometric);
```

Lennard-Jones interactions: geometric combination rule

$$V_{LJ}(r_{ij}) = \frac{C_{ij}^{(12)}}{r_{ij}^{12}} - \frac{C_{ij}^{(6)}}{r_{ij}^{6}}$$

$$C_{ij}^{(6)} = \left(C_{ii}^{(6)}\,C_{jj}^{(6)}\right)^{1/2}$$
$$C_{ij}^{(12)} = \left(C_{ii}^{(12)}\,C_{jj}^{(12)}\right)^{1/2}$$

From the gromacs manual

Non-bonded interactions are flexible

```
// A combination rule can be specified for non-self interactions
ParticleTypesInteractions interactions(CombinationRule::Geometric);

// LJ parameters can be added per particle
interactions.add(ParticleName("OW"), C6(0.0026173456), C12(2.634129e-06));
interactions.add(ParticleName("H"), C6(0), C12(0));
```

Non-bonded interactions are flexible

```
// A combination rule can be specified for non-self interactions
ParticleTypesInteractions interactions(CombinationRule::Geometric);

// LJ parameters can be added per particle
interactions.add(ParticleName("OW"), C6(0.0026173456), C12(2.634129e-06));
interactions.add(ParticleName("H"), C6(0), C12(0));

interactions.add(ParticleName("OMet"), C6(0.0022619536), C12(1.505529e-06));
interactions.add(ParticleName("CMet"), C6(0.0088755241), C12(2.0852922e-05));
```

Non-bonded interactions are flexible

```
// A combination rule can be specified for non-self interactions
ParticleTypesInteractions interactions(CombinationRule::Geometric);

// LJ parameters can be added per particle
interactions.add(ParticleName("OW"), C6(0.0026173456), C12(2.634129e-06));
interactions.add(ParticleName("H"), C6(0), C12(0));

interactions.add(ParticleName("OMet"), C6(0.0022619536), C12(1.505529e-06));
interactions.add(ParticleName("CMet"), C6(0.0088755241), C12(2.0852922e-05));

// LJ interactions can be explicitly specified for non-self interactions
interactions.add(ParticleName("CMet"),ParticleName("OMet"),
                 C6(0.0044806276), C12(5.1373125e-06));
```

Molecules are built up from Particles

```
Molecule water;
// Particles in a Molecule must have a name
// They may have a charge and a residue name
water.addParticle(ParticleName("Oxygen"), Charge(-0.82), Ow);
water.addParticle(ParticleName("H1"), Charge(+0.41), H);
water.addParticle(ParticleName("H2"), Charge(+0.41), H);
```

Molecules are built up from Particles

```
Molecule water;
// Particles in a Molecule must have a name
// They may have a charge and a residue name
water.addParticle(ParticleName("Oxygen"), Charge(-0.82), Ow);
water.addParticle(ParticleName("H1"), Charge(+0.41), H);
water.addParticle(ParticleName("H2"), Charge(+0.41), H);

Molecule methanol;
methanol.addParticle(ParticleName("Me1"), Charge(-0.574), CMet);
methanol.addParticle(ParticleName("O2"), Charge(+0.176), OMet);
// Note that a Particle can be used in multiple Molecules
methanol.addParticle(ParticleName("H3"), Charge(+0.398), H);
```

Intramolecular exclusions can be added

```
// water exclusions
water.addExclusion("H1", "Oxygen");
water.addExclusion("H2", "Oxygen");
water.addExclusion("H1", "H2");
```

Intramolecular exclusions can be added

```
// water exclusions
water.addExclusion("H1", "Oxygen");
water.addExclusion("H2", "Oxygen");
water.addExclusion("H1", "H2");

// methanol exclusions
methanol.addExclusion("Me1", "O2");
methanol.addExclusion("Me1", "H3");
methanol.addExclusion("H3", "O2");
```

Molecules can have bonds

```
// A harmonic bond has an equilibrium distance and force constant
HarmonicBondType ohBond(distance, forceConstant);
water.addInteraction("Oxygen", "H1", ohBond);
water.addInteraction("Oxygen", "H2", ohBond);
```

Molecules can have bonds

```
// A harmonic bond has an equilibrium distance and force constant
HarmonicBondType ohBond(distance, forceConstant);
water.addInteraction("Oxygen", "H1", ohBond);
water.addInteraction("Oxygen", "H2", ohBond);

// Similar setup for the methanol bonds (united atoms on Me1)
HarmonicBondType oh3Bond(0.100, 31380);
methanol.addInteraction("O2", "H3", oh3Bond);
HarmonicBondType ometBond(0.136, 376560);
methanol.addInteraction("O2", "Me1", ometBond);
```

Addition of angles and dihedrals looks similar to bonds

Topologies are built in stages

```
// The various parts are combined by the topology builder
TopologyBuilder topologyBuilder;
```

Topologies are built in stages

```
// The various parts are combined by the topology builder
TopologyBuilder topologyBuilder;

// Molecules are added to the topology builder by type and number
topologyBuilder.addMolecule(water, 2);
topologyBuilder.addMolecule(methanol, 1);
```

Topologies are built in stages

```
// The various parts are combined by the topology builder
TopologyBuilder topologyBuilder;

// Molecules are added to the topology builder by type and number
topologyBuilder.addMolecule(water, 2);
topologyBuilder.addMolecule(methanol, 1);

// Interactions added to topology builder after all molecules added
topologyBuilder.addParticleTypesInteractions(interactions);
```

Topologies are built in stages

```
// The various parts are combined by the topology builder
TopologyBuilder topologyBuilder;

// Molecules are added to the topology builder by type and number
topologyBuilder.addMolecule(water, 2);
topologyBuilder.addMolecule(methanol, 1);

// Interactions added to topology builder after all molecules added
topologyBuilder.addParticleTypesInteractions(interactions);

// Once all the components have been added,  the Topology is built
Topology topology = topologyBuilder.buildTopology();
```

Non-topology simulation data

```
// Read in coordinates and velocities
std::vector<gmx::RVec> coordinates = readFromCSV("coords.csv");
std::vector<gmx::RVec> velocities  = readFromCSV("velocities.csv");
```

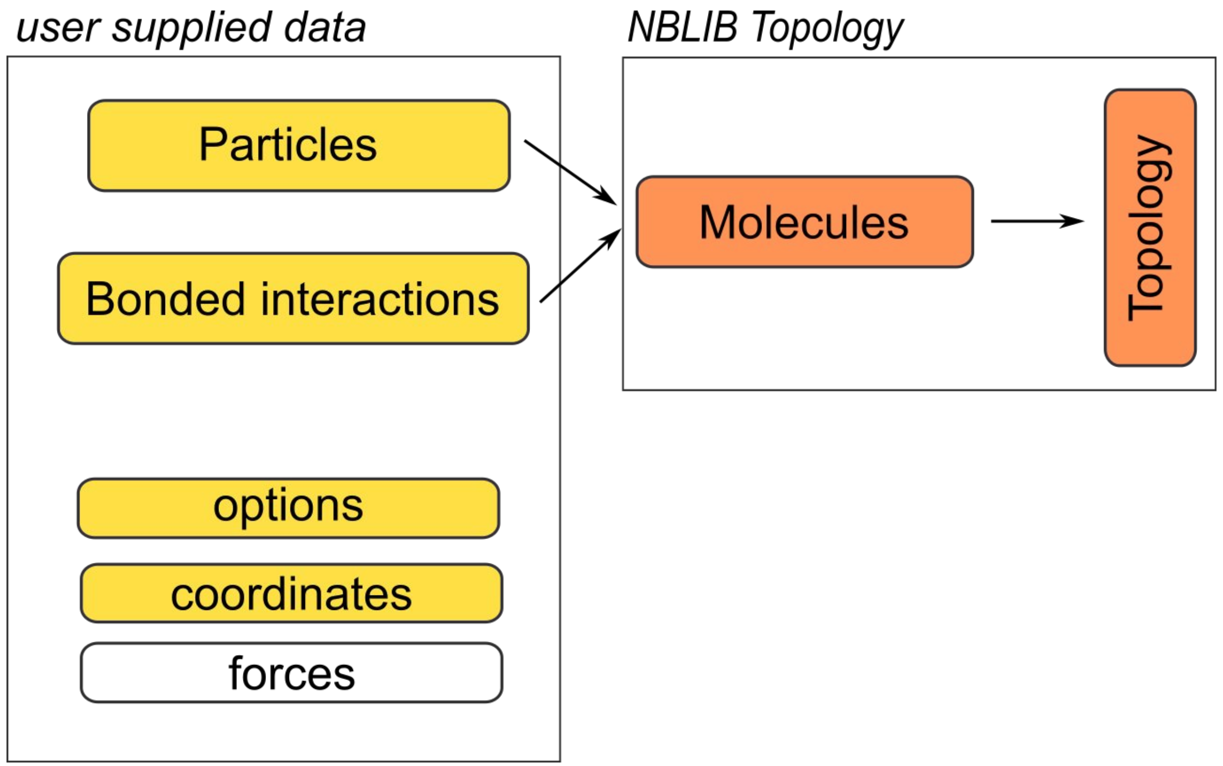
Non-topology simulation data

```
// Read in coordinates and velocities
std::vector<gmx::RVec> coordinates = readFromCSV("coords.csv");
std::vector<gmx::RVec> velocities  = readFromCSV("velocities.csv");

// Here we initialize forces to zero
std::vector<gmx::RVec> forces(coordinates.size(), gmx::RVec(0, 0, 0));
```

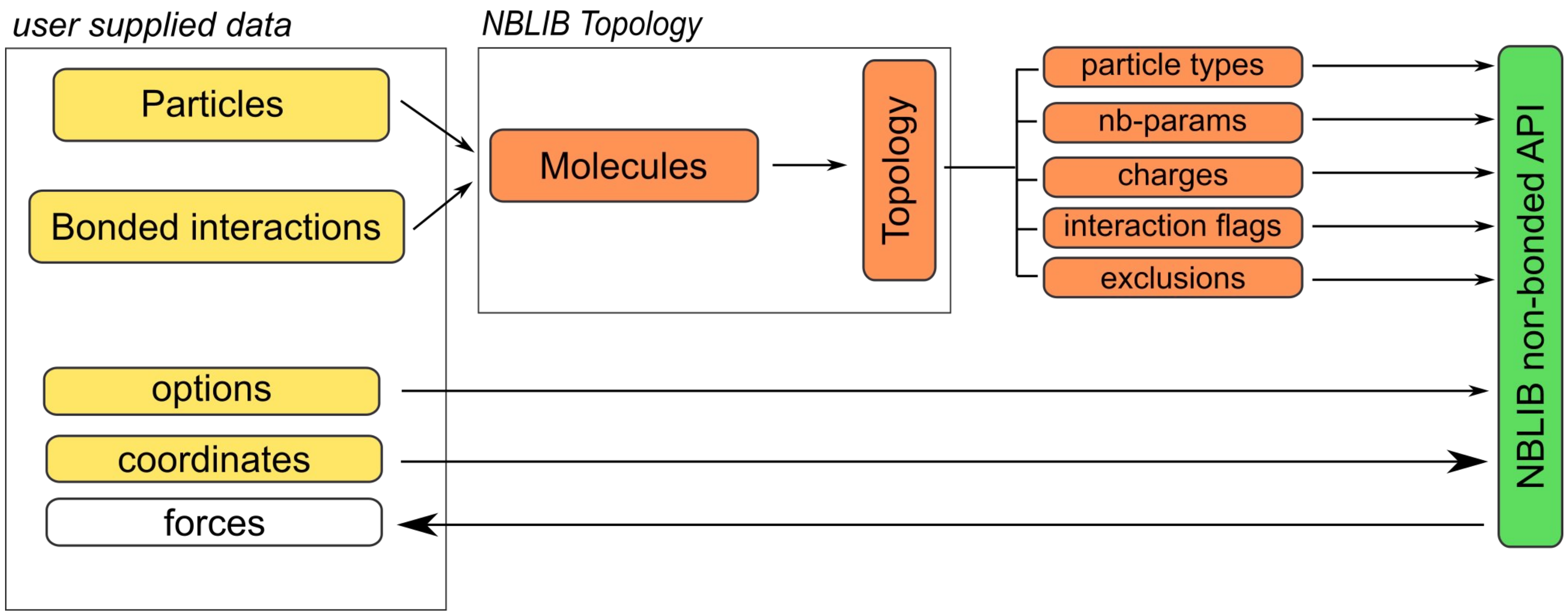
Non-topology simulation data

```
// Read in coordinates and velocities
std::vector<gmx::RVec> coordinates = readFromCSV("coords.csv");
std::vector<gmx::RVec> velocities  = readFromCSV("velocities.csv");

// Here we initialize forces to zero
std::vector<gmx::RVec> forces(coordinates.size(), gmx::RVec(0, 0, 0));

// A box dimension must be specified
Box box(6.05449);
```

Summary: Topology



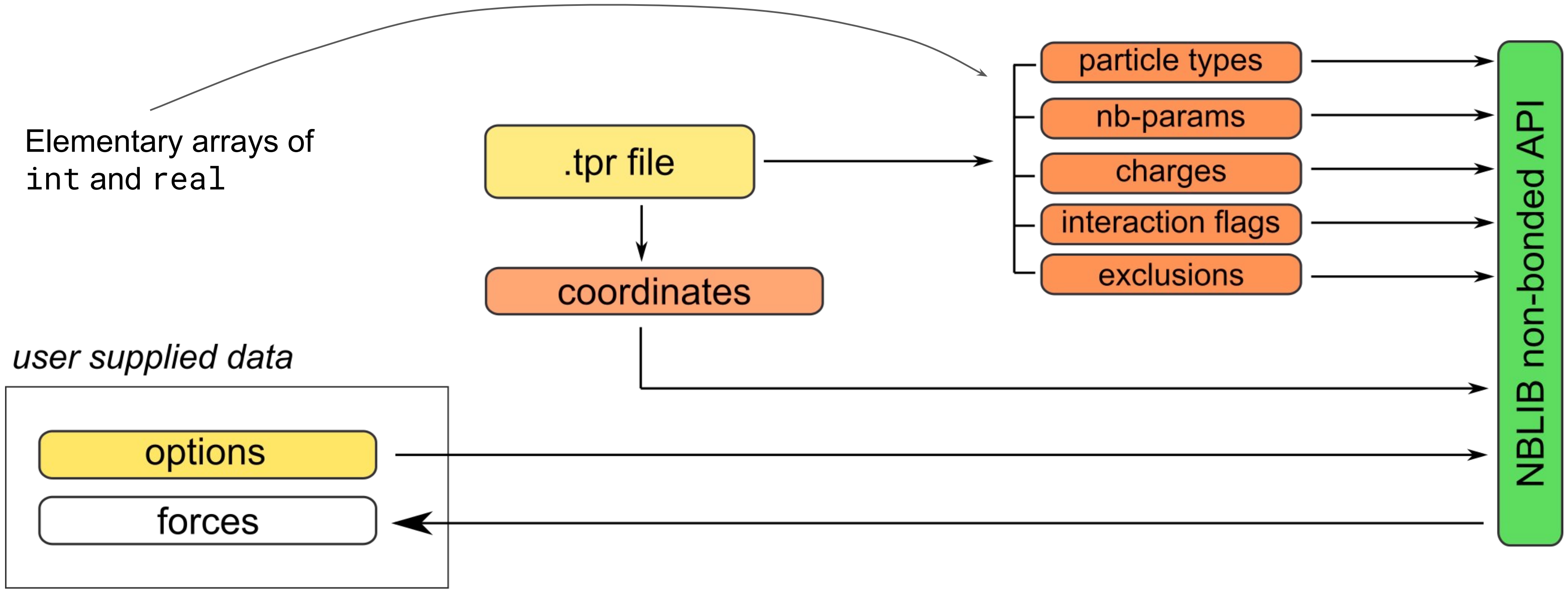
Connecting Topology to the force API



Building the non-bonded force calculator

```
NBKernelOptions   options; // has sensible defaults, sets PairList cutoff, etc.

NBForceCalculator forceCalculator(topology.particleTypesOfAllParticles(),
                                  topology.nonBondedParameters(),
                                  topology.charges(),
                                  topology.particleInteractionFlags(),
                                  topology.exclusionsLists().ranges(),
                                  topology.exclusionsLists().elements(),
                                  options);

forceCalculator.updatePairlist(box, coordinates);

forceCalculator.compute(box, coordinates, forces);
```

Alternative data sources for the force API



Alternate data source: TPR file input

```
TprReader tprReader(“example.tpr”);
ForceCalculator forceCalculator(tprReader.particleTypes(),
                                tprReader.nonBondedParams(),
                                tprReader.charges(),
                                tprReader.interactionFlags(),
                                tprReader.exclusionLists(),
                                tprReader.exclusionRanges(), options);

std::vector<gmx::RVec> coordinates = tprReader.coordinates();
std::vector<gmx::RVec> forces      = tprReader.forces();

Box box = tprReader.box();

forceCalculator.compute(box, coordinates, forces);
```

Building the listed force calculator

```
int numThreads = 4;

// getInteractionData() returns the interaction parameters for all
// bonded interactions in the topology

ListedForceCalculator listedCalculator(topology.getInteractionData(),
                                       topology.numParticles(),
                                       numThreads);

listedCalculator.compute(box, coordinates, forces);
```

A complete MD loop!

```
// Integrator is initialized with an array of inverse masses and the box
LeapFrog integrator(topology, box);
```

A complete MD loop!

```
// Integrator is initialized with an array of inverse masses and the box
LeapFrog integrator(topology, box);

real timestep = 1.0;
```

A complete MD loop!

```
// Integrator is initialized with an array of inverse masses and the box
LeapFrog integrator(topology, box);

real timestep = 1.0;

for (int step = 0; step < 100; step++)
{

}
```

A complete MD loop!

```
// Integrator is initialized with an array of inverse masses and the box
LeapFrog integrator(topology, box);

real timestep = 1.0;

for (int step = 0; step < 100; step++)
{
    forceCalculator.compute(box, coordinates, forces);
}
```

A complete MD loop!

```
// Integrator is initialized with an array of inverse masses and the box
LeapFrog integrator(topology, box);

real timestep = 1.0;

for (int step = 0; step < 100; step++)
{
    forceCalculator.compute(box, coordinates, forces);
    listedForceCalculator.compute(box, coordinates, forces);
}
```

A complete MD loop!

```
// Integrator is initialized with an array of inverse masses and the box
LeapFrog integrator(topology, box);

real timestep = 1.0;

for (int step = 0; step < 100; step++)
{
    forceCalculator.compute(box, coordinates, forces);
    listedForceCalculator.compute(box, coordinates, forces);
    integrator.integrate(timestep, coordinates, velocities, forces);
}
```

A complete MD loop!

```
// Integrator is initialized with an array of inverse masses and the box
LeapFrog integrator(topology, box);

real timestep = 1.0;

for (int step = 0; step < 100; step++)
{
    forceCalculator.compute(box, coordinates, forces);
    listedForceCalculator.compute(box, coordinates, forces);
    integrator.integrate(timestep, coordinates, velocities, forces);
    zeroCartesianArray(forces); // Zero the forces each step
}
```

MD mini-app

```
// Integrator is initialized with an array of inverse masses and the box
LeapFrog integrator(topology, box);
real timestep = 1.0;
for (int step = 0; step < 100; step++)
{
    forceCalculator.compute(coordinates,forces);
    listedForceCalculator.compute(coordinates, forces());
    integrator.integrate(timestep, coordinates,velocities,forces);
    … // other stuff
}
```

Outline

- What is NB-LIB
- Background
- Implementation
- **Example workflows**

Example workflows with NBLIB

- **Compute subsets of interactions**
    - drug docking
    - multiple time stepping
    - QM/MM
- Multiple states
    - swarms of trajectories
    - replica exchange
    - changing topology during the simulation

Workflow #1: Compute Subsets of Interactions

```
Molecule polymer1;
Molecule polymer2;
Molecule water;
// add particles, exclusions, interactions, as before
```

Workflow #1: Compute Subsets of Interactions

```
Molecule polymer1;
Molecule polymer2;
Molecule water;
// add particles, exclusions, interactions, as before

TopologyBuilder fullSystemBuilder;
fullSystemBuilder.addMolecule(polymer1, 1);
fullSystemBuilder.addMolecule(polymer2, 1);
fullSystemBuilder.addMolecule(water, 100);
```

Workflow #1: Compute Subsets of Interactions

```
Molecule polymer1;
Molecule polymer2;
Molecule water;
// add particles, exclusions, interactions, as before

TopologyBuilder fullSystemBuilder;
fullSystemBuilder.addMolecule(polymer1, 1);
fullSystemBuilder.addMolecule(polymer2, 1);
fullSystemBuilder.addMolecule(water, 100);

TopologyBuilder polymersBuilder;
polymersBuilder.addMolecule(polymer1, 1);
polymersBuilder.addMolecule(polymer2, 1);
```

Workflow #1: Compute Subsets of Interactions

```
Molecule polymer1;
Molecule polymer2;
Molecule water;
// add particles, exclusions, interactions, as before

TopologyBuilder fullSystemBuilder;
fullSystemBuilder.addMolecule(polymer1, 1);
fullSystemBuilder.addMolecule(polymer2, 1);
fullSystemBuilder.addMolecule(water, 100);

TopologyBuilder polymersBuilder;
polymersBuilder.addMolecule(polymer1, 1);
polymersBuilder.addMolecule(polymer2, 1);

Topology fullSystem = fullSystemBuilder.buildTopology();
Topology polymers   = polymersBuilder.buildTopology();
```

Workflow #1: Compute Subsets of Interactions

```
NBForceCalculator fullSystemNBForceCalculator(/* as before */);
ListedForceCalculator fullSystemListedCalculator(/* as before */);
LeapFrog integrator(topology, box);
```

Workflow #1: Compute Subsets of Interactions

```
NBForceCalculator fullSystemNBForceCalculator(/* as before */);
ListedForceCalculator fullSystemListedCalculator(/* as before */);
LeapFrog integrator(topology, box);

real timestep = 1.0;
for (int step = 0; step < 100; step++)
{
    fullSystemNBForceCalculator.compute(box, coordinates,forces);
    fullSystemListedCalculator.compute(box, coordinates, forces);
    integrator.integrate(timestep, coordinates,velocities,forces);
    zeroCartesianArray(forces);
}
```

Workflow #1: Compute Subsets of Interactions

```
NBForceCalculator fullSystemNBForceCalculator(/* as before */);
ListedForceCalculator fullSystemListedCalculator(/* as before */);
LeapFrog integrator(topology, box);

real timestep = 1.0;
for (int step = 0; step < 100; step++)
{
    fullSystemNBForceCalculator.compute(box, coordinates,forces);
    fullSystemListedCalculator.compute(box, coordinates, forces);
    integrator.integrate(timestep, coordinates,velocities,forces);
    zeroCartesianArray(forces);
}

// Now only compute forces for the polymers
NBForceCalculator PolymersNBForceCalculator(/* as before */);
PolymersNBForceCalculator.compute(box, polymerCoordinates,polymerForces);
```

Work is ongoing to expose the energy as part of the non-bonded API

Example workflows with NBLIB

- Compute subsets of interactions
  - drug docking
  - multiple time stepping
  - QM/MM
- **Multiple states**
  - swarms of trajectories
  - replica exchange
  - changing topology during the simulation

Workflow #2: Multiple states

```
// Same topology in both systems
Topology first = systemBuilder.buildTopology();
Topology second = systemBuilder.buildTopology();
```

Workflow #2: Multiple states

```
// Same topology in both systems
Topology first = systemBuilder.buildTopology();
Topology second = systemBuilder.buildTopology();

// first coordinates and velocities
std::vector<gmx::RVec> firstCoordinates = readFromCSV("coordsFirst.csv");
std::vector<gmx::RVec> firstVelocities  = readFromCSV("velsFirst.csv");
```

Workflow #2: Multiple states

```
// Same topology in both systems
Topology first = systemBuilder.buildTopology();
Topology second = systemBuilder.buildTopology();

// first coordinates and velocities
std::vector<gmx::RVec> firstCoordinates = readFromCSV("coordsFirst.csv");
std::vector<gmx::RVec> firstVelocities  = readFromCSV("velsFirst.csv");

// second coordinates and velocities
std::vector<gmx::RVec> secondCoordinates = readFromCSV("coordsSecond.csv");
std::vector<gmx::RVec> secondVelocities  = readFromCSV("velsSecond.csv");
```

Workflow #2: Multiple states

```
// first force calculators and integrator
NBForceCalculator firstNBForceCalculator(/* first data */);
LeapFrog firstIntegrator(/* first data */);
```

Workflow #2: Multiple states

```
// first force calculators and integrator
NBForceCalculator firstNBForceCalculator(/* first data */);
LeapFrog firstIntegrator(/* first data */);

// second force calculators and integrator
NBForceCalculator secondNBForceCalculator(/* second data */);
LeapFrog secondIntegrator(/* second data */);
```

Workflow #2: Multiple states

```
real timestep = 1.0;
for (int step = 0; step < 100; step++)
{
    firstNBForceCalculator.compute(box, firstCoordinates,firstForces);
    secondNBForceCalculator.compute(box, secondCoordinates,secondForces);

    firstIntegrator.integrate(/* first data */);
    secondIntegrator.integrate(/* second data */);
    // also need to zero the forces


}
```

Workflow #2: Multiple states

```
real timestep = 1.0;
for (int step = 0; step < 100; step++)
{
    firstNBForceCalculator.compute(box, firstCoordinates,firstForces);
    secondNBForceCalculator.compute(box, secondCoordinates,secondForces);

    firstIntegrator.integrate(/* first data */);
    secondIntegrator.integrate(/* second data */);
    // also need to zero the forces

    if (step % 10) { /* swap first and second velocities */ };
}
```

Future goals

- Find users!

Future goals

- Find users!
- Performance aspects
    - domain decomposition
    - data transfer
    - task scheduling

Directly involved at KTH:
Berk Hess
Design discussions + code review
Christian Blau
Eric Irrgang
Erik Lindahl
Mark Abraham
Paul Bauer
Szilard Páll

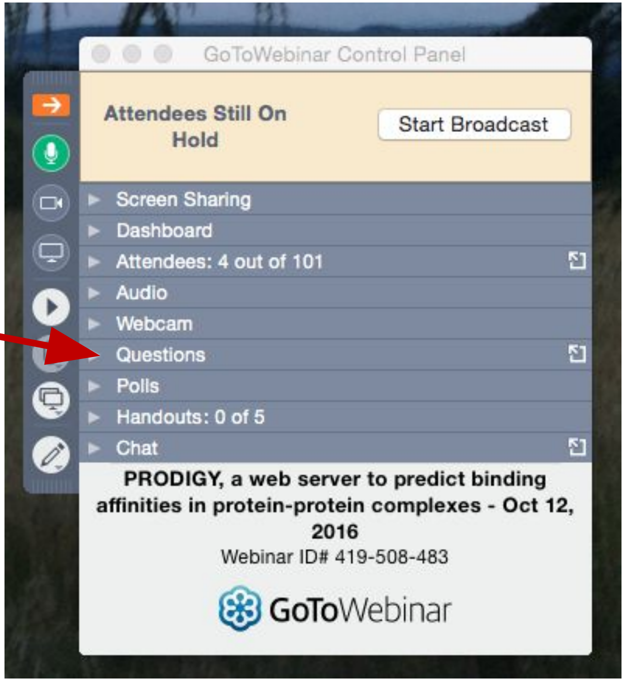
SNIC
Swedish National Infrastructure for Computing

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Audience Q&A session

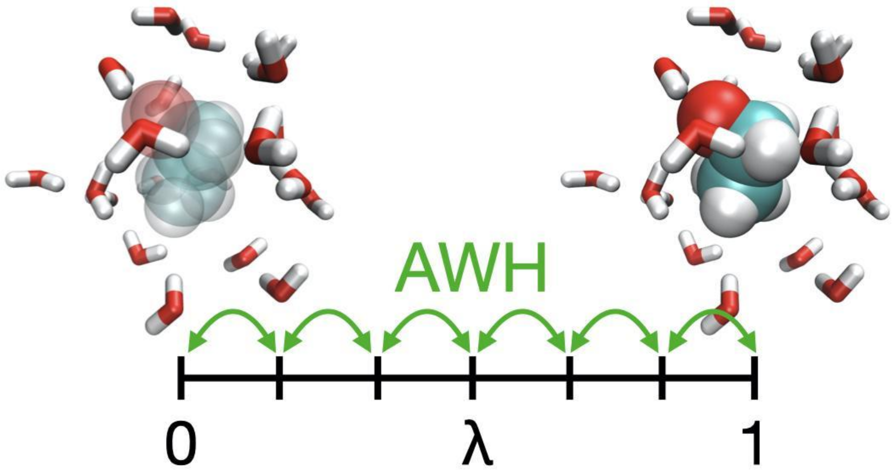
- Please use the Questions function in GoToWebinar application
    - If you *don’t have audio*, please mention that in the question.

- Any other questions or points to discuss after the live webinar? Join the discussions at http://ask.bioexcel.eu.



PRODIGY, a web server to predict binding affinities in protein-protein complexes - Oct 12, 2016
Webinar ID# 419-508-483

Next webinar occasion (30 March 2021)



Applying the Accelerated Weight histogram method to alchemical transformations.

by
Berk Hess and
Magnus Lundborg

See
https://bioexcel.eu/