The Molecular Modelling Toolkit

Konrad Hinsen

Centre de Biophysique Moléculaire (CNRS)

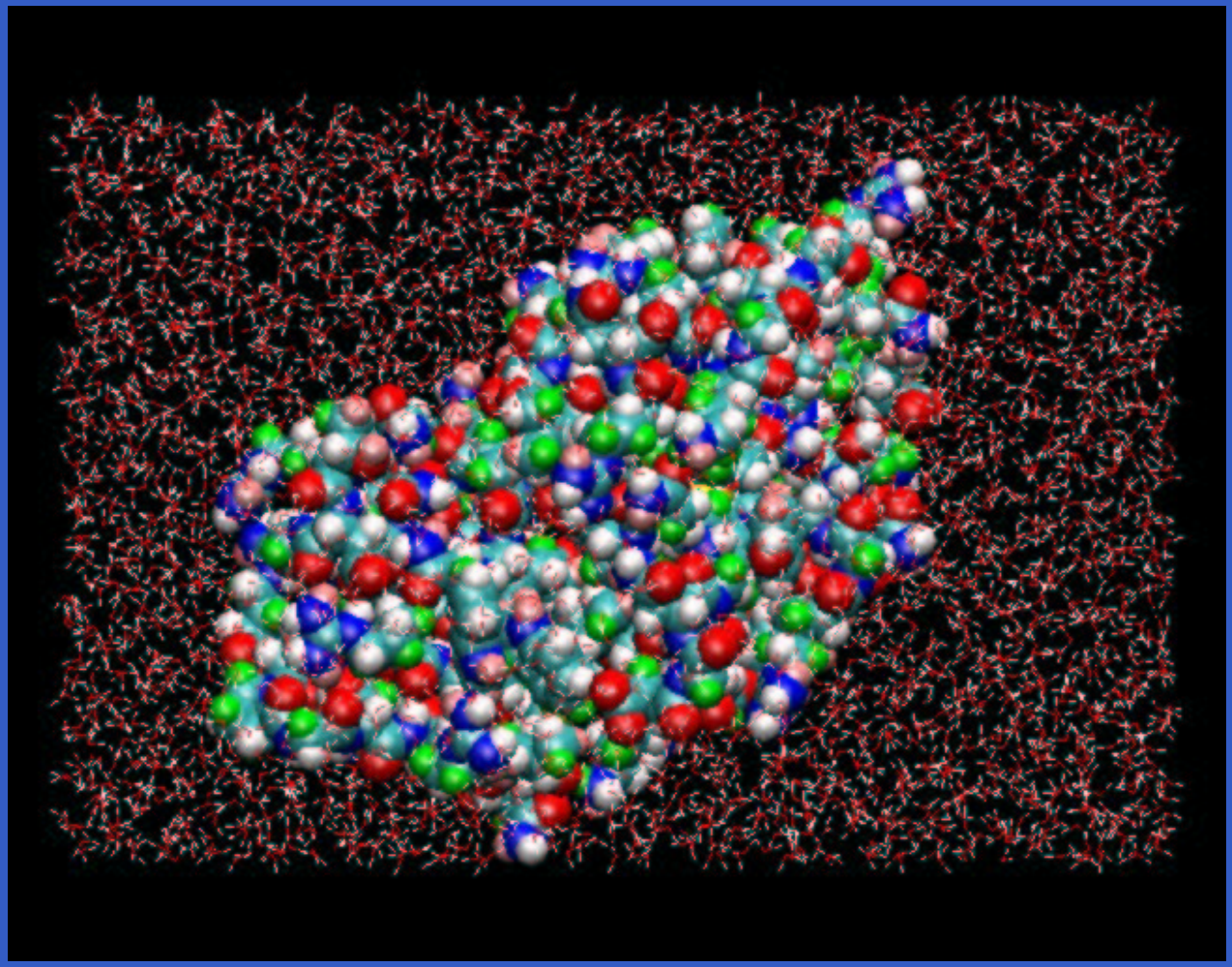
Orléans, France

Biomolecular Simulations

Features:

- Big and complex systems
  $\Rightarrow$ complex data structures
- CPU intensive: several weeks per job
  $\Rightarrow$ efficient code
- Models and algorithms in evolution
  $\Rightarrow$ modifiable code
- Analysis specific to each situation
  $\Rightarrow$ easy “programming”

Lysozyme



Description of biomolecular systems

- Multiple levels: atom, group, molecule, complex, . . .
- Specialisations:

  group $\rightarrow$ amino acid residue

  molecule $\rightarrow$ peptide chain

  complex $\rightarrow$ protein
- Interaction parameters
- Multiple conformations, trajectories, . . .

$\Rightarrow$ **object-oriented description**

Community

- Many users, few developers
- A few popular simulation packages (most not free)
- No standard file formats
- Users are “trapped” by a package:
  - Can’t switch between codes
  - Can’t modify code (too messy)
  - Can’t write their own (too much work)

MMTK Design Goals (1/3)

Development, implementation, and testing of computational techniques:

- code basis for all kinds of molecular simulations
- central part: object-oriented representation of molecular systems
- modular design
- carefully designed data structures
- interactive as well as batch use

MMTK Design Goals (2/3)

Foundation for end-user applications:

- library approach
- compatibility with other libraries (user interfaces etc.)

Reliability:

- modern software-engineering methods
- minimize the risk of user errors

MMTK Design Goals (3/3)

Portability:

- built on portable tools
- use portable data formats

Efficiency:

- suitable for long MD simulations
- parallelization

MMTK Design Choices

- High-level object-oriented language: Python (18000 lines)
- Time-critical parts implemented in C (10000 lines)
- Trajectory files in netCDF format, fully self-contained
- Provide defaults wherever possible, but stay on the safe side

MMTK Overview

A Python library for molecular simulations . . .

. . . with **C modules** for time-critical routines.

The **user** writes programs in **Python**,

simple scripts or graphical applications

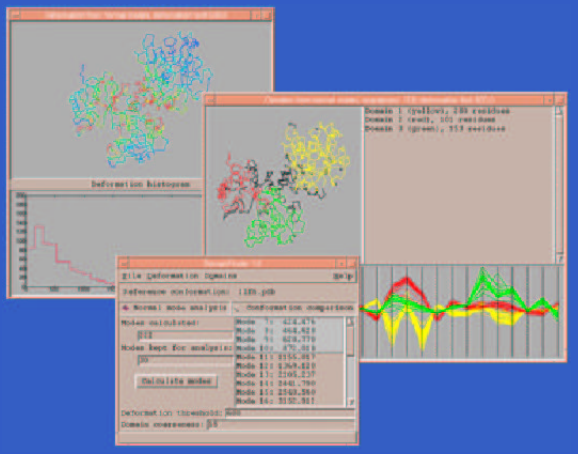
```
# A simple Molecular Dynamics simulation.
#

from MMTK import *
from MMTK.Proteins import Protein
from MMTK.ForceFields import Amber94ForceField
from MMTK.Dynamics import VelocityVerletIntegrator, Heater, \
                          TranslationRemover, RotationRemover
from MMTK.Visualization import view
from MMTK.Trajectory import Trajectory, TrajectoryOutput, \
                            RestartTrajectoryOutput, StandardLogOutput, \
                            trajectoryInfo

# Define system
universe = InfiniteUniverse(Amber94ForceField())
universe.protein = Protein('bala1')

# Initialize velocities
universe.initializeVelocitiesToTemperature(50.*Units.K)
print 'Temperature: ', universe.temperature()
print 'Momentum: ', universe.momentum()
print 'Angular momentum: ', universe.angularMomentum()

# Create integrator
integrator = VelocityVerletIntegrator(universe, delta_t=1.*Units.fs)
```



Mixed Python/C programming

Advantages:

- all Python advantages:
  - rapid development
  - access to many existing libraries
  - interactivity
  - easy GUI construction
- no loss of speed in the C parts
- access to libraries in Fortran and C

Partial Feature List

- construction and modification of molecular systems
- analysis of conformations
- visualization (via external programs)
- energy evaluation (AMBER force field)
- energy minimization
- molecular dynamics (NVE, NVT, NPT)
- normal modes
- simplified protein models

OpenSource building blocks

MMTK uses OpenSource projects:

- Python and its library
- Numerical Python (arrays, linear algebra)
- Scientific Python (geometry, IO, ...)
- netCDF (portable binary files)
- LAPACK (linear algebra)

MMTK is used by OpenSource projects:

- DomainFinder (protein domain analysis)
- nMOLDYN (MD trajectory analysis)

Scientific software

Specificities:

- High specialization $\rightarrow$ small user base
- Users are colleagues, need to know algorithms in detail
- Scientific publication requires disclosure of all procedures
- Publically funded research is for the public

OpenSource is the best solution!

MMTK development

- 1 developer, $\approx$100 users
- some “power users” who contribute code
- Few potential developers: must know simulation methods, object-oriented methods, and Python
- Significant reduction of development time due to Python

Interfaces to other code

- Reads and writes PDB files
- Reads and writes CHARMM trajectory files
- Converters for DLPOLY and AMBER trajectory files
- Reads AMBER forcefield parameter files

Wish list

- Common data formats for system definitions and trajectories
- More sharing of results, especially trajectories
- More OpenSource in neighbouring fields (e.q. quantum chemistry)
- Support for software development from scientific organizations

Conclusions (1/2)

After some years' experience:

- Python
  - made MMTK possible
  - scares away potential developers
  - causes some memory overhead
- Basic design works well
- netCDF was a good choice
- further optimization of the C parts desirable

Conclusions (2/2)

Recommendations for OpenSource in science:

- Write building blocks (libraries)
- Document interfaces (and don’t change them)
- Use high-level languages for easier interfacing
- Profit from other people’s (OpenSource) work