Contents

- MCell - particle-based reaction dynamics simulator
- Motivation for Python API for MCell
- New MCell4 implementation
- Model structure
- MCell4 architecture
- BioNetGen library
- Validation and testing
- Demonstration
- Performance results
- Conclusion
Current MCell and new Python Interface

- MCell3 uses as input a domain specific language called MDL (Model Description Language)
  - The definition is mostly static and prescribed, still capable to describe a wide range of processes

- Python provides capabilities to do any manipulations once the simulation is running such as:
  - Change simulated state based on what’s going on in the simulation
  - Interact with external simulators
MCell4 - New MCell Implementation

- MCell3 is implemented in the C language
  - It has gotten complex over the >15 years of development
  - Practically impossible to parallelize, hard to do substantial changes

- New implementation in C++
  - Provides Python API
  - Prepared for parallelization
  - Easier extensibility
  - Native support for BioNetGen species and reactions
Base MCell4 Model Structure

- Having a defined structure helps with orientation in models
- Allows to create reusable models and libraries
Modularity

- Each subsystem (pathway) definition is independent and can be merged with others
- Requires uniform naming of substrates

parameters & subsystem 1:
definition of a pathway that affects the process I am studying

parameters & subsystem 2:
definition of another pathway

parameters & subsystem 3:
and one more pathway

geometry objects 1
instantiation: what molecules I will have initially in my system and what is the geometry

geometry objects 2

observables:
what do I need to know about my simulated system

model

not all dependencies are shown here
Overall Architecture of MCell4

- LibMCell
  - Python Interface
  - Model Representation
- MCell3 MDL Parser
- BNG Library
  - Species
  - Reactions
- MCell4 Engine
  - Scheduler
  - Events
  - Simulation State

Model Representation

LibMCell

BNG Library

MCell4 Engine

Python Interface

Species

Reactions

Scheduler

Events

Simulation State
Python API Definition and Generator

Definition of classes in YAML format

```yaml
Complex:
superclass: BaseDataClass
doc: |
  This class represents a complex molecule composed of molecule instances.
  It is either defined using a BNGL string or using a list of elementary molecule instances.
  On top of that, orientation may be defined.
  This class is used as argument in cases where either a fully qualified instance or a pattern can be provided such as in observable Count.

items:
  - name: name
type: str
default: empty
doc: |
  When set, this complex instance is initialized from a BNGL string passed as this argument, the string is parsed during model initialization so the molecule types it uses don't have to be defined before initialization.

  - name: elementary_molecule_instances
type: List[ElementaryMoleculeInstance*]
default: empty
doc: Individual molecule instances contained in the complex.

  - name: orientation
type: Orientation
default: Orientation.DEFAULT
doc: |
  Specifies orientation of a molecule.
  When Orientation.DEFAULT if kept then during model initialization is 'orientation' set to Orientation.NONE for volume complexes and to Orientation.UP for surface complexes.
  Ignored by derived class Species.

  - name: compartment_name
type: str
default: unset
```

Base C++ classes that hold the model representation

Python interface to C++ code

Constant names used in API (for Python generator)

API definition for syntax-directed editors

Documentation
BioNetGen Library

- The preferred way to define species and reactions in MCell4 models is in the BioNetGen language
- Implemented new BNG library
  - Existing NFSim is very useful but hard to maintain
  - Designed with independence on MCell4 in mind, hopefully useful in other tools
  - New implementation contains:
    - BNGL parser, classes to represent BNGL constructs, BNG reactions engine
- BNGL parser testsuite with 59 tests
- Created a proposal on improved surface reaction definition in BNGL
- Current status
  - Validated with complex models of SynGAP and CaMKII holoenzyme & other BNGL tests
  - Each complex may have just one compartment for now
  - No support for BNGL functions
MCell Usage Scenarios and Model File Formats

- Code history, comments, code reviews

Data model (JSON)

MDL (MCell3)

Python + BNGL (MCell4)
MCell4 Testing

Reference data (stored in mcell_tests repo, validated against MCell3, MCell3-R or BionetGen)

MDL model

mcell -mdl2datamodel4

mcell (mcell3 mode)

mcell -mcell4

Data model

data_model_to_pymcell

Python code (+BNGL)

Data model_to_mdl

BNGL export

Output data

data_output_diff

PASS/FAIL
Testing & Build Infrastructure

- New Python implementation of a test & build infrastructure
- New tests:
  - MDL: 214
  - Python/MCell4: 38
  - BNGL: 107
  - Data model: 27
- Total number of tested variants with conversions to various variants (MDL, Python/MCell4, BNGL, data model):
  - MCell4: 1184
  - MCell3: 433
- Single script to build CellBlender package and test it
- Public CellBlender releases 3.4.0, 3.5.0, and 3.5.1
- Virtual machines for build on MacOS, Linux Centos 6-7, Linux Debian 8-10, Windows 10
Example of Validation - CaMKII Holoenzyme Model

- 100 000 iterations (0.1 s), average from 512 runs with different random seeds, BNG values are obtained with NFSim, bands represent standard deviation
- Molecules in MCell3R and MCell4 use diffusion constant $1e^{-3}$ cm$^2$/s to emulate well-mixed solution (usual value is around $1e^{-6}$ cm$^2$/s)
- CaM1C - CaM(C~1,N~0,camkii), CaM1N - CaM(C~0,N~1,camkii), KCaM2N - CaMKII(Y286~P,cam!1).CaM(C~0,N~2,camkii!1)
Demonstration

- Model export from CellBlender
- MCell4 Python model example
- Debugging in Eclipse
Performance Results - MCell3 Reactions

- Single-threaded execution, Linux Debian 9, AMD Ryzen 9 3900X @3.8GHz
- MCell3 3.5.1, MCell4 4.0.internal.8
Performance Results - BioNetGen Reactions

Relative performance (higher is better)

Memory usage (in MB, lower is better)
Conclusion

● Python interface
  ○ Subsystems (sets of species and reactions) as independent modules
  ○ Provides a way to model features that are not directly supported
  ○ Integration with external simulators
  ○ Usage of Python debuggers & syntax-directed editors

● New MCell4 implementation
  ○ Extensible, prepared for parallelization

● New BioNetGen library
  ○ Used for all species and reactions in MCell4
  ○ Planning to release it as a standalone library

● Automatic build and testing system
Acknowledgements

Thomas M. Bartol
Robert Kuczewski
Ali Sinan Saglam
Leo McKee-Reid
Mariam Ordyan
Oliver Ernst
Guadalupe C. Garcia
Sara Sameni
Margot Wagner
Rachel Mendelsohn

James R. Faeder
Terrence J. Sejnowski
Backup slides
Features and Code Statistics

- Main features missing in MCell4 compared to MCell3:
  - Custom time step (needs to be validated)
  - Periodic boundary conditions
  - Checkpointing
  - Trimolecular reactions (not planned)

- Improved dynamic geometry
  - Changing geometry based on user’s Python code

- Lines of C & C++ code (without comments)

<table>
<thead>
<tr>
<th></th>
<th>MCell3</th>
<th>NFSim + nfsimCInterface</th>
<th>MCell4 + libMCell</th>
<th>libBNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code</td>
<td>50 516</td>
<td>26 851</td>
<td>22 236</td>
<td>8 324</td>
</tr>
</tbody>
</table>
Integration with Other Simulators

- Need to model
  - external environment
  - physics not covered by MCell
- Data exchange
- Python to define the interactions
- Allow parallel execution of included simulators
  - e.g. using task-based parallelism