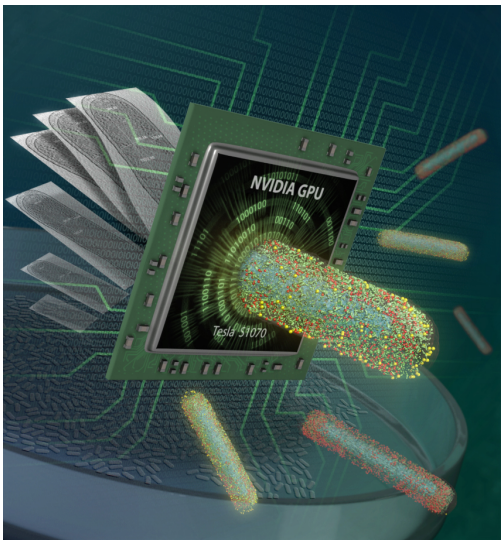
Lattice Microbes Problem Solving Environment User's Guide



LM Version 2.3, pyLM Version 1.1
June 1, 2016

Joseph R. Peterson, Mike J. Hallock, Elijah Roberts, John A. Cole, Piyush Labhsetwar, John E. Stone, and Zaida Luthey-Schulten

University of Illinois at Urbana-Champaign
http://www.scs.illinois.edu/schulten/lm

Description

The Lattice Microbes User's Guide describes the capabilities, license and installation of the software. Lattice Microbes development is supported in part by the DOE (Office of Science BER) under grant DE-FG02-10ER6510, the NIH (Center for Macromolecular Modeling and Bioinformatics) under grant NIH-RR005969, and the NSF under grant MCB-08226143.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This User's Guide contains a description of the software, its capabilities and instructions for installing Lattice Microbes, the software described in the following publications: [1–3]. This guide is very much a work in progress and will continue to be expanded. At present, it should contain enough information to get you started using the Lattice Microbes software.

1.1 Lattice Microbes

Studying cellular processes, which show inherently noisy non-deterministic behavior, with single molecule resolution on timescales of biological relevance such as the lifetime of a cell, requires considerable computational effort. Lattice Microbes [1,2] is software developed to sample realizations of the spatially homogenous and heterogeneous stochastic Master equations, with thousands of reactions among hundreds of molecular species. The software uses graphics processing units (GPUs) to exploit the natural parallelism afforded by the Master equations to access timescales orders-of-magnitude larger than other particle– and grid–based software for sampling stochastic cellular processes as seen in Figure 1.1. While Lattice Microbes was originally designed for simulating single *E. coli* cells on one GPU, the desire to simulate cellular consortia and larger species like yeast, drove the development of a new version of Lattice Microbes that utilizes multiple GPUs to share the work [3]. In addition to larger simulations, multiple GPUs allow small simulations to be completed more quickly.

1.2 pyLM

pyLM is a Problem Solving Environment (PSE) for biological simulations [5]. Written in Python, it wraps and extends Lattice Microbes. The PSE is comprised of a base set of functionality to set up, monitor and modify simulations, as well as a set of standard post–processing routines that interface to other Python packages, including NumPy, SciPy, H5py, and iGraph to name a few.

The PSE is shown schematically in Figure 1.2. It sits on top of a SWIG interface that allows the C++ code to be accessible from the Python terminal. Using pyLM allows the user to set up, run and post–process simulations all within a single script. A general workflow for using LM is shown in Figure 1.3. For tutorials on using pyLM please see the "Instruction Guide" and for in–depth

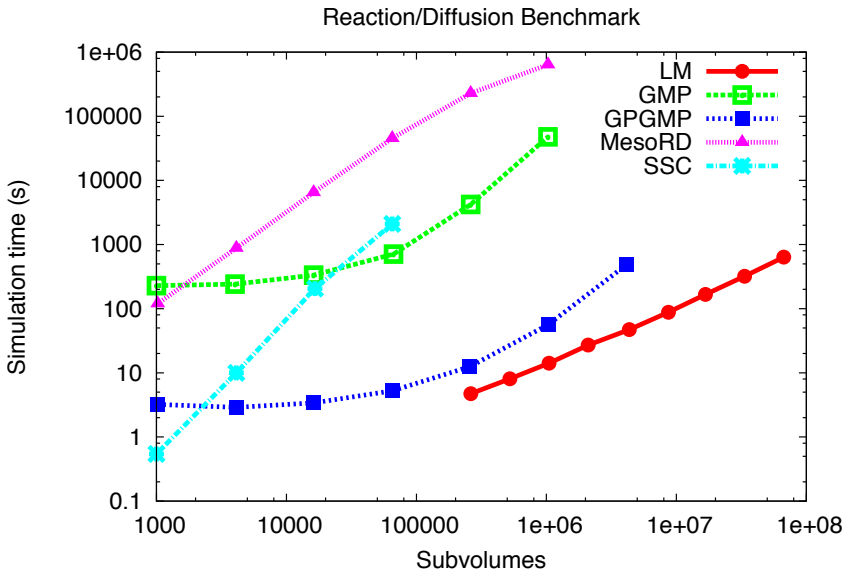


Figure 1.1: Comparison (lower is better) of simulation time to completion with Lattice Microbes to other grid-based stochastic software for a simulation of spatially resolved reversible bimolecular reaction with 100K particles at different system volumes. Data for other simulation codes from the paper: [4]. Key: LM - "Lattice Microbes Multi-particle diffusion RDME", GMP - "Gillespie Multi-Particle", GPGMP - "GPU Gillespie Multi-Particle", MesoRD - "MesoRD", SSC - "Stochastic Simulation Compiler".

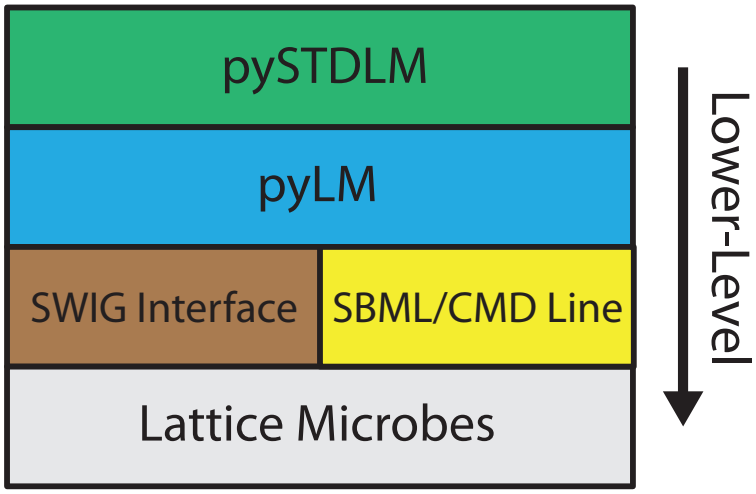


Figure 1.2: A schematic of the pyLM and the Lattice Microbes software.

description of all pyLM functionality please see the documentation "Reference Guide" available on the website.

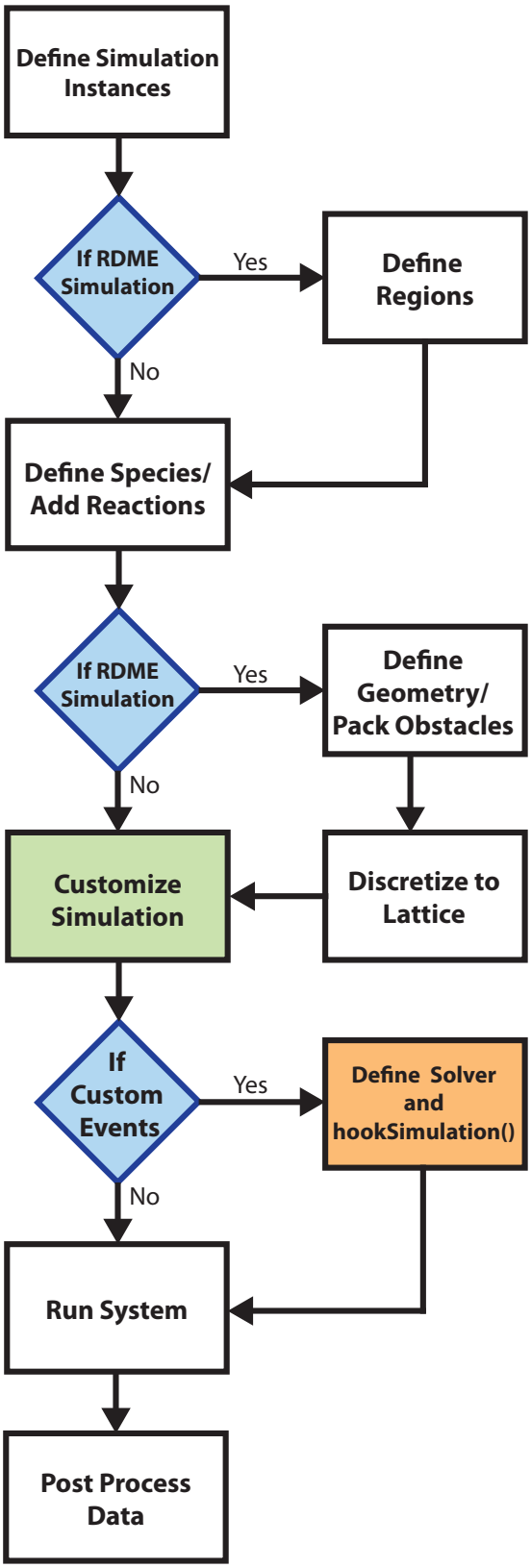


Figure 1.3: The workflow of the pyLM PSE.

Chapter 2

Capabilities

2.1 Stochastic Simulations

Lattice Microbes can be used to simulate chemical master equations (CME):

$$\frac{dP(\mathbf{x},t)}{dt} = \sum_{r}^{R}[-a_r(\mathbf{x})P(\mathbf{x},t) + a_r(\mathbf{x}_\nu - \mathbf{S_r})P(\mathbf{x} - \mathbf{S_r},t)]$$

and reaction-diffusion master equations (RDME):

$$\begin{aligned}\frac{dP(\mathbf{x},t)}{dt} = &\sum_{\nu}^{V}\sum_{r}^{R}[-a_r(\mathbf{x}_\nu)P(\mathbf{x}_\nu,t) + a_r(\mathbf{x}_\nu - \mathbf{S_r})P(\mathbf{x}_\nu - \mathbf{S_r},t)] \\ &+ \sum_{\nu}^{V}\sum_{\xi}^{\pm\hat{i},\hat{j},\hat{k}}\sum_{\alpha}^{N}[-d^\alpha x_\nu^\alpha P(\mathbf{x},t) + d^\alpha(x_{\nu+\xi}^\alpha + 1)P(\mathbf{x} + 1_{\nu+\xi}^\alpha - 1_\nu^\alpha,t)]\end{aligned}$$

using a variety of methods. Both CME and RDME support a number of different reaction types including zeroth, first, second, and second order self reaction. These reactions are of the form in Table 2.1. All rate constants input to Lattice Microbes should be the stochastic rate constant that has been scaled by the volume and Avogadro's number so that it is in units of **sec^{-1}**. Use the table for the correct conversion factor.
In addition, the pyLM problem solving environment provides tools to setup, run and post-process stochastic simulations as well as integrate the stochastic techniques with other simulation methodologies (for example, see: [6]). The capabilities are outlined here.

2.1.1 Well-Stirred Simulations

CME simulations require species, reactions along with their rate constants, and initial specie counts to be specified.

A number of algorithms for sampling the CME are available with CPU and GPU implementations. The Gillespie stochastic simulation algorithm (SSA) [7] is the slowest but most straightforward. The next-reaction (NR) and fluctuating next-reaction (FNR) [8] algorithms are also available and

Order	Form	Parameters	Macroscopic Units	Stochastic Rate Constant (s^{-1})
0th	$\emptyset \rightarrow \mathrm{A}$	k	Ms^{-1}	$k \cdot V \cdot N_A$
1st	$\mathrm{A} \rightarrow \mathrm{B}$	k	s^{-1}	k
2nd	$\mathrm{A} + \mathrm{B} \rightarrow \mathrm{C}$	k	$M^{-1}s^{-1}$	$\frac{k}{V \cdot N_A}$
2nd (Self)*	$2\mathrm{A} \rightarrow \mathrm{B}$	k	$M^{-1}s^{-1}$	$\frac{k}{V \cdot N_A}$
Michaelis-Menten†	$\mathrm{E} + \mathrm{S} \rightarrow \mathrm{E} + \mathrm{P}$	k_{cat}, K_M	s^{-1}, M	$\frac{k_{cat}}{V \cdot N_A}$, $K_M V \cdot N_A$
Competitive Michaelis-Menten†	$\mathrm{E} + \mathrm{I} + \mathrm{S} \rightarrow \mathrm{E} + \mathrm{I} + \mathrm{P}$	k_{cat}, K_M, K_I	s^{-1}, M, M	$\frac{k_{cat}}{V \cdot N_A}$, $K_M V \cdot N_A$, $K_I V \cdot N_A$
Uncompetitive Michaelis-Menten†	$\mathrm{E} + \mathrm{I} + \mathrm{S} \rightarrow \mathrm{E} + \mathrm{I} + \mathrm{P}$	k_{cat}, K_M, K_I	s^{-1}, M, M	$\frac{k_{cat}}{V \cdot N_A}$, $K_M V \cdot N_A$, $K_I V \cdot N_A$
Noncompetitive Michaelis-Menten†	$\mathrm{E} + \mathrm{I} + \mathrm{S} \rightarrow \mathrm{E} + \mathrm{I} + \mathrm{P}$	k_{cat}, K_M, K_I	s^{-1}, M, M	$\frac{k_{cat}}{V N_A}$, $K_M V \cdot N_A$, $K_I V \cdot N_A$

Table 2.1: Reactions available to both CME and RDME. Here, the stochastic rate constant should be computed from the macroscopic rate constant (perhaps from experiment) using the volume of the experiment, V, and Avogadro's number, N_A. *Note that for a 2nd order self reaction, the rate of A disappearing is $2k$. †Michaelis-Menten type reactions are currently only supported in CME simulations, and only compute the propensity of forming the product using the steady-state assumption.

(may, depending on the count of particles) considerably speed up the simulation using variable time-stepping. These techniques along with their capabilities can be found in Table 2.3.

Method	CPU	GPU	Solver Keyword
SSA	Y	Y	lm::cme::GillespieDSolver
NR	Y	Y	lm::cme::NextReactionSolver
FNR	Y	Y	lm::cme::FluctuatingNRSolver

Table 2.2: CME sampling algorithms available in Lattice Microbes. CPU and GPU columns indicate whether the algorithm is available (Y) or unavailable (N) for the particular compute device. The Solver Keyword column indicates the string to pass to the "lm" executable to perform that simulation.

2.1.2 Spatially-Resolved Simulations

In addition to species, reactions along with their rate constants, and initial specie counts, RDME also requires the lattice spacing, spatial organization of objects (membrane, cytoplasm, etc.) and diffusion rates in and between these spatial objects to be specified.

Two methods for solving the RDME are available in Lattice Microbes. The first is the next subvolume (NS) method [9], which is analogous to the next reaction method, of the CME. The other is a constant timestep method called Multiparticle diffusion (MPD) developed to take advantage of the fine-grained parallelism afforded by the GPU [2]. In the latter case, the timestep is specified by the diffusion time:

$$\tau = \frac{\lambda^2}{2 \cdot n \cdot D}$$

where λ is the lattice spacing, n is the dimensionality (conventionally 2 or 3) and D is the macroscopic diffusion constant. Both are GPU accelerated, while only one is available on the CPU as indicated in Table 2.3.
Additionally, a version of the MPD algorithm is available for computers with multiple-GPUs (called the MGPU-MPD solver) that can effectively and efficiently split the computation over the GPUs [10].

Method	CPU	GPU	Solver Keyword
NS	Y	Y	lm::rdme::NextSubvolumeSolver
MPD	N	Y	lm::rdme::MpdRdmeSolver
MGPU-MPD	N	Y	lm::rdme::MGPUMpdRdmeSolver

Table 2.3: RDME sampling algorithms available in Lattice Microbes. CPU and GPU columns indicate whether the algorithm is available (Y) or unavailable (N) for the particular compute device. The Solver Keyword column indicates the string to pass to the "lm" executable to perform that simulation.

2.2 Problem Solving Environment

pyLM and the included library of standard systems pySTDLM provide a problem solving environment for setting up, running and analyzing stochastic biological simulations [5]. It contains functionality for specifying simulation setup including:

- Named species
- Initial counts and distributions
- Reactions and rates
- Spatial localization and definition
- Diffusion properties
- Obstacles
- Define custom simulation flow

In addition, there are a number of pre- and post-processing functionalities including (but not limited to):

- Integration with Jupyter Notebook
- Handles to data in the popular Numpy array representation
- Plotting species averages/variances and individual time traces
- Plotting Kymographs of spatial species distributions
- Reaction network generation
- Dynamic reaction network representations

The standard library (pySTDLM) also includes a number of cell shapes, colony layout routines, and previously published reaction systems. Finally, pyLM provides functionality for accessing the underlying representation of the simulation data with time-based interrupts to allow users to modify or analyze the state of the simulation on-the-fly. This last functionality allows there merger of different methodologies together. For examples of how to use pyLM please refer to the "Instruction Guide" and for full enumeration of functionality please see the "Reference Manual" both of which are available online at `http://www.scs.illinois.edu/schulten/lm`.

Chapter 3

Installation

This chapter outlines the system and software requirements, and describes both methods of installing Lattice Microbes. Installation of precompiled binaries is discussed in Section 3.3. Installation from source code is discussed in Section 3.4. Regardless of which of the two of these options is taken, pyLM requires additional installation instructions that are described in Section 3.5.

3.1 System requirements

The Lattice Microbes software has been tested on Linux 2.6 and Mac OS X 10.8 and 10.9. Although the software can be run entirely on a system's CPU, Lattice Microbes was designed from the ground up to take advantage of NVIDIA Fermi (compute 2.0) and later GPUs which allow for orders-of-magnitude speedup over the CPU-only implementations.

3.2 Software Requirements

In order to take full advantage of the Lattice Microbes software, several external software packages should be installed on your system. While these lists appear daunting, the majority of the software have binary installers that simplify installation. The full enumeration of required packages:

- Lattice Microbes v2.2 — `http://www.scs.illinois.edu/schulten/lm/`
- CUDA v5.5+ — `http://developer.nvidia.com/cuda-downloads/`
- Python v2.7 — `http://www.python.org/download/`
- HDF v1.8.8+ — `http://www.hdfgroup.org/HDF5/release/obtain5.html`
- Protocol Buffers **v2.4.1** — `http://code.google.com/p/protobuf/downloads/`
- SBML v5.9+ — `http://sourceforge.net/projects/sbml/files/libsbml/`
- h5py v2.2.1+ — `http://code.google.com/p/h5py/downloads`
- NumPy and SciPy — `http://www.scipy.org/install.html`
- iGraph 0.6.5+ — `http://igraph.sourceforge.net/download.html`
- pygexf v0.2.2+ — `http://pythonhosted.org/pygexf/users.html`
- lxml 3.2.4+ — `https://pypi.python.org/pypi/lxml`
- matplotlib v1.3.0+ — `http://matplotlib.org/downloads.html`

Additional (highly recommended) optional packages:

- VMD v1.9+ — `http://www.ks.uiuc.edu/Research/vmd/`
- Gephi v0.8.2+ — `https://gephi.org/`
- Cytoscape v3.0+ — `http://www.cytoscape.org`
- An MPI Library:
  - OpenMPI — `http://www.open-mpi.org/software/ompi/v1.6/`, or
  - MPICH2 — `http://www.mpich.org/downloads/`, or
  - MVAPICH2 — `http://mvapich.cse.ohio-state.edu/download/`

Requisite for GPU acceleration, users of NVIDIA Fermi or later GPUs should ensure that the CUDA 5.5+ drivers and libraries are installed and up to date.

The popular molecular dynamics visualization and analysis software VMD can be used to view and animate output trajectories. We believe that these capabilities are vital in understanding the details of how microscopic phenomena give rise to cell–scale behavior. For viewing of networks of interaction both statically and dynamically we recommend installing Gephi. Static networks can also be viewed with the popular Cytoscape graph visualizer.

Python scripts are used to set up and analyze realistic models of crowded cellular environments. Users should ensure that Python is available on their system. In addition, pyLM requires NumPy, SciPy, matplotlib, iGraph, h5py, lxml and pygexf for proper functioning. Most, if not all of these, are available as binaries, via a package manager (such as macports, apt, etc.) or via "pip".

Some users may find it necessary to compile Lattice Microbes from source code if, for example, they wish to create custom reaction rate equations for their simulations, or will be installing Lattice Microbes on a compute cluster, or a pre–compiled binary is not available for their machine. The HDF5 and Protocol Buffers libraries and their development packages are required dependencies for source compilation. See section 3.4 for more details.

The Systems Biology Markup Language, or SBML, is used to efficiently import complex reaction networks into LM kinetic/spatial models. Users intending on compiling from source should download and install libSBML in order to ensure this functionality is available.

Finally, Lattice Microbes can be compiled with MPI in order to enable the distribution of replicates across many compute nodes on a cluster. We recommend OpenMPI for clusters without Infiniband interconnects and MVAPICH for clusters with Infiniband.

Lattice Microbes is not a GUI program, it must be used from the command line. This enables it to efficiently run on high performance computing (HPC) clusters with minimal overhead. Consequently, all user interaction with the software, including installation, must be performed through the command line interface. In this chapter, commands to be executed from the command line are written as: "`[user@host ~/usr]$ ls`" which means to run the command "`ls`" from the directory "`~/usr`".

3.3 Installing a precompiled binary

Available at the above url are several binaries precompiled for use either with or without GPU acceleration. Users should select the binary appropriate for their system.

For the purposes of these instructions, it is assumed that the Lattice Microbes software will be installed into the directory `/home/<user>/usr`, also referred to as `~/usr`. If you wish to install the software elsewhere, please adjust the instructions accordingly.

Download the appropriate binary distribution to a temporary directory `/tmp`. Open a terminal and then change to the directory, unpack and copy the binary and libraries:

```
[user@host ~]$ cd /tmp
[user@host /tmp]$ tar zxvf lm-2.2_<platform>.tgz
[user@host /tmp]$ cp lm-2.2/bin/* ~/usr/bin
[user@host /tmp]$ mkdir -p ~/usr/lib/lm
[user@host /tmp]$ cp -r lm-2.2/lib/lm ~/usr/lib/lm
[user@host /tmp]$ cp -r lm-2.2/lib/python ~/usr/lib/
```

Copy the VMD plugin to the plugins/molefile directory of your VMD installation:

```
[user@host /tmp]$ cp vmd/MACOSXX86/molfile/lmplugin.so /Applications/VMD
1.9.1.app/
Contents/vmd/plugins/MACOSXX86/molfile                    (OS X)
```

or:

```
[user@host /tmp]$ cp vmd/LINUXAMD64/molfile/lmplugin.so /usr/local/lib/vmd/plugins/
LINUXAMD64/molfile                                        (LINUX)
```

Note: if you have installed the software into a non-global location, such as installing to `~/usr`, you will need to add the installation directory to you path. For example, you might add the following line to your `~/.bashrc` or `~/.bash_profile` file:

```
export PATH=$PATH:$HOME/usr/bin
```

Furthermore, if you have downloaded the CUDA version of Lattice Microbes, you must set the environment variable for the loader to find the CUDA libraries. This can be done by adding the following line to your `~/.bashrc` or `~/.bash_profile` file:

(OS X)

```
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:/usr/local/cuda/lib
```

(LINUX)

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib
```

However, on Linux, this path may be slightly different, so if this command does not work, check your directory to make sure you have the right PATH for the CUDA library. If you are still unsure,

check with your system administrator.

Finally, test the software installation:
```
[user@host /tmp]$ lm --help
```
If a help prompt is printed (enumerating the command line options) installation is likely correct. Otherwise, check that libraries are in the correct environmental paths.

3.4 Installing from source code

As the computer architecture landscape becomes increasingly heterogeneous, many users may find that it is easiest to simply compile Lattice Microbes from source code on their own machine. In order to accomplish this as painlessly as possible, this section is designed with the average user—not a software engineer—in mind.

3.4.1 Satisfying external dependencies

Although Lattice Microbes can be compiled and run without taking advantage of available GPU hardware, we believe that the real strength of the software is the orders-of-magnitude speedup gained through GPU acceleration. In order to use Lattice Microbes to its fullest, you will first need to install the CUDA toolkit and drivers appropriate for your platform, if they are not installed already. They can be found here:

`http://developer.nvidia.com/cuda-downloads`

The MacOSX installations are trivial—simply click through the installation wizards. Linux installation can be somewhat more challenging, and will likely require you exit out of the GUI interface. Help can be found here:

`http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-linux/index.html`

Several other key features of the code also rely on external dependencies. Python is required for programmatic setup of realistically crowded models of cells and analysis of simulation data. The popular SBML file format can be used to import complex reaction networks. Both Python and libSBML should be installed prior to compiling the code, if they are not already. MacOSX systems generally have Python pre-installed, but Linux users can find it via a package manager or download it here:

`http://www.python.org/download/`

The requisite SBML libraries can be found here:

`http://sourceforge.net/projects/sbml/files/libsbml/`

Again, installation on a Mac should be a breeze; installing libSBML for Linux should also be straightforward. On Red Hat-based systems, the available .rpm package should be downloaded to a convienient location, such as `/home/<user>/usr`, and installed using rpm in a terminal window, for example:

```
[user@host /home/<user>/usr]$ rpm2cpio libSBML-5.6.0-Linux-x64.rpm | cpio
-idmv
```

For Debian-derived distributions, the .deb package should be downloaded and installed using dpkg, e.g.:

```
[user@host /home/<user>/usr]$ sudo dpkg -i libSBML-5.6.0-Linux-x64.deb
```

Although CUDA, Python, and libSBML are not *strictly* necessary, they impart functionalities crucial for studying large and complex biochemical systems under *in vivo* crowding conditions. The Protocol Buffers and HDF5 libraries, on the other hand, are absolutely essential to compiling the code. The protobuf library (version 2.4.1 only!!!) is used for serialization of messages across the transport layer. It is available from:

```
http://code.google.com/p/protobuf/downloads/list
```

This will need to be compiled from source. Download the latest version to a convenient directory install:

```
[user@host /home/<user>/usr]$ tar zxvf protobuf-2.4.1.tar.gz
[user@host /home/<user>/usr]$ cd protobuf-2.4.1
[user@host /home/<user>/usr/protobuf-2.4.1]$ configure
[user@host /home/<user>/usr/protobuf-2.4.1]$ make
[user@host /home/<user>/usr/protobuf-2.4.1]$ sudo make install
```

The HDF5 library is used for reading models from and writing simulation data to HDF5 formatted files. The HDF5 library is available from:

```
http://www.hdfgroup.org/HDF5/release/obtain5.html
```

Again, after downloading to a convenient directory extract the package, and move its contents to someplace that it is not likely to be moved, such as:

```
/home/<user>/usr/hdf5-1.8.8-mac-x86_64-static/
```

Note the word “static” in the directory above; static libraries, which end with extensions .a or .la for MacOS and LINUX, are compiled along with everything else into one monolithic executable binary. Also available are shared or dynamic libraries, which end in .dylib or .so; these are not compiled into the executable binary, but rather remain separate and are linked to. Whether you compile against static or dynamic libraries is up to you, but you will have to treat them differently

when setting up your `local.mk` file, which will be described in Section 3.4.3.

Finally, if you intend to compile with MPI for use on a compute cluster, you will need to acquire and install the requisite libraries. There are several implementations of MPI available, and care should be taken in order to choose the right one for your cluster.
If one of the above packages is needed and is not already installed on your system, download a binary or source installation package and follow the installation instructions that accompany it. If you have problems, please contact your system administrator for assistance.
Note: if you install any external shared libraries into a non-global location, such as installing to `~/usr`, you will need to set an environment variable for the loader to find these libraries. For example, you might add the following line to your `~/.bashrc` or `~/.bash_profile` file:

(OS X)

```
[user@host /home/<user>/usr]$ export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:/usr/
local/cuda/lib
```

(LINUX)

```
[user@host /home/<user>/usr]$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/
local/cuda/lib
```

where /usr/local/cuda/lib is the path to a dynamic library you will be compiling against, in this case a CUDA library. This will need to be done for all of the dynamic libraries you compile against.

3.4.2 Unpack the source distribution

For the purposes of these instructions, it is assumed that the Lattice Microbes software will be installed into the directory `/home/<user>/usr`, also referred to as `~/usr`. If you wish to install the software elsewhere, please adjust the instructions accordingly.

Download the source distribution from the URL above to the directory `~/usr/src`. Then unpack the software:

```
[user@host ~]$ cd ~/usr/src
[user@host ~/usr/src]$ tar zxvf lm-2.2.tgz
[user@host ~/usr/src]$ cd lm-2.2
```

3.4.3 Configuring the build for your local environment

The Lattice Microbes source distribution ships with two default configuration files, one for Linux and one for Mac OS X. These files are located at:

`docs/config/local.mk.linux`
and
`docs/config/local.mk.osx`.

To begin, copy the file corresponding to your system to `local.mk`:

```
[user@host ~/usr/src/lm-2.2]$ cp docs/config/local.mk.<platform> local.mk
```

Edit the `local.mk` file to contain the correct options and file locations for your local environment. For example, if you installed the HDF5 and protobuf libraries into the `/home/<user>/usr` directory, you should set the PROTOBUF and HDF5 options as follows:

```
HDF5_DIR := /home/<user>/usr

PROTOBUF_DIR := /home/<user>/usr
```

Note: the HDF5 library actually has two dependencies of its own—SZIP and ZLIB—and the corresponding libraries are likely bundled with the software you downloaded. This will work for linking dynamically against the libraries.
Each optional package has a section in the `local.mk` file that is initially disabled and begins with a line like:

```
USE_XXXX := 0
```

To enable a specific package, set the flag corresponding to the package to 1 and set the options and locations appropriately. For example, if you are using Open MPI you might set the MPI options as follows:

```
USE_MPI := 1
MPI_COMPILE_FLAGS = -DOMPI_SKIP_MPICXX=1 $(shell mpicc --showme:compile)
MPI_LINK_FLAGS = $(shell mpicc --showme:link)
```

For alternate MPI implementations you may need to experiment with the `mpicc` command to discover the correct settings or look at the example configuration files included with the Lattice Microbes source distribution.

If you are using Python 2.7 you might set the Python options as follows:

```
USE_PYTHON := 1
PYTHON_SWIG := /usr/bin/swig
PYTHON_INCLUDE_DIR := `python-config --includes`
PYTHON_LIB := `python-config --libs`
```

If you are using CUDA with a capable device you might set the CUDA options as follows:

```
USE_CUDA := 1
CUDA_DIR := /usr/local/cuda
```

The location of your CUDA_DIR will depend on your installation of Cuda. ou may also need to change the CUDA_ARCH variable to match your GPU version. You can determine your GPU's compute capability you need to enable by visiting `https://developer.nvidia.com/cuda-gpus`.
If you want to build Lattice Microbes with support for importing SBML files (and have libSBML installed to `/home/<user>/usr`) you might set the SBML options as follows:

```
USE_SBML := 1
SBML_DIR := /home/<user>/usr
```

(OS X) If you want to build the VMD plugin and have VMD installed to the Applications folder you might set the VMD options to:

```
USE_VMD := 1
VMD_DIR := "/Applications/VMD\ 1.9.1.app/Contents/vmd"
```

(LINUX) If you want to build the VMD plugin and have VMD installed to `/usr/local` you might set the VMD options to:

```
USE_VMD := 1
VMD_DIR := /usr/local/lib/vmd
```

Finally, you should set the installation location and build name for the Lattice Microbes software at the top of the `local.mk` file:

```
BUILD_DIR := Build-osx
INSTALL_PREFIX := /home/<user>/usr
```

Additional Compile Time Features

There are additional compile time flags that can be enabled to allow for nonstandard features in Lattice Microbes. If the option is specified as a FLAG it can be enabled by adding them to both CCFLAGS and CUDA_FLAGS defined in your `local.mk`. If the option is specified as a FEATURE it can be defined in `local.mk`. These are:

- -DGLOBAL_S_MATRIX – FLAG – Enable this flag if your stoichiometry matrix is large. By default the S matrix is stored in GPU constant memory, and therefore the number of species times the number of reactions must be less than 4096 (16KB). By enabling this feature, the maximum size of the S matrix is increased to 262144 (1MB) allowing for up to 1024 reactions with 256 species.
- -DLATTICE_MAX_OCCUPANCY=16 – FLAG – Enabling this flag doubles the maximum particles per site from 8 to 16, doubling the memory cost and potentially slowing the simulation speed. **WARNING: There is no support for 8/16 particle lattice interoperability. Files created by code with a specific occupancy should only be read and written by binaries built with the same value.**
- USE_VERBOSITY_LEVEL=# – FEATURE – Specify the verbosity of the Lattice Microbes output to the console/standard out. A larger number means more logging/debugging information is printed. A smaller number means less. The default level is 4 and a level of 10 will turn on debugging symbols during compile time.
- USE_PROF – FEATURE – Enabling this feature will cause Lattice Microbes to time the algorithm execution.

3.4.4 Build and install the software

Now that the build is configured, build and install the source code:

```
[user@host ~/usr/src/lm-2.2]$ make LOCALMK=local.mk
[user@host ~/usr/src/lm-2.2]$ sudo make install
```

Note: if you have installed the software into a non-global location, such as installing to ~/usr, you will need to add the installation directory to you path. For example, you might add the following line to your ~/.bashrc file:

```
export PATH=$PATH:$HOME/usr/bin
```

Finally, test the software installation:

```
[user@host ~/usr/src/lm-2.2]$ lm --help
```

If no errors occur, the software is installed correctly. If you get errors about a missing library, you will likely have to add their paths to the library path environment variable.

3.5 pyLM Installation

pyLM comes with both the source and the binary distribution of Lattice Microbes. Configuring pyLM to work requires installing external software and setting environment paths to the correct locations.

One the Lattice Microbes installation is complete, the paths that Python looks for code in must be updated. Add the paths to your ~/.bashrc or ~/.bash_profile:

```
export PYTHONPATH=/home/<user>/usr/lib/lm:$PYTHONPATH
export PYTHONPATH=/home/<user>/usr/lib/python:$PYTHONPATH
```

3.5.1 External Libraries

The instructions in this section may only be applicable for the particular version of the libraries that are listed. If the installation fails, please see the documentation on the website or in the tar file. Most of these can be installed via a package manager or via "pip".

Installing h5py

Once downloaded, h5py can be installed with the following set of commands:

```
[user@host /tmp]$ tar -xvf h5py-2.2.0.tar.gz
[user@host /tmp]$ cd h5py-2.2.0
[user@host /tmp/h5py-2.2.0]$ python setup.py build --hdf5=/home
/<user>/usr/hdf5-1.8.8-mac-x86_64-static/lib
```

```
[user@host /tmp/h5py-2.2.0]$ python setup.py test
[user@host /tmp/h5py-2.2.0]$ sudo python setup.py install
```

Alternatively, this can be installed via pip:

```
[user@host /tmp]$ pip install h5py
```

Installing NumPy

NumPy can be installed on almost any system from binary.

Installing SciPy

SciPy can be installed on almost any system from binary.

Installing iGraph

iGraph can be installed for Mac from binary.
For Linux machines, you must first install the C library:

```
[user@host /tmp]$ tar -xvf igraph-0.6.5.tar.gz
[user@host /tmp]$ cd igraph-0.6.5
[user@host /tmp/igraph-0.6.5]$ ./configure
[user@host /tmp/igraph-0.6.5]$ make
[user@host /tmp/igraph-0.6.5]$ sudo make install
```

Then the following commands may be used to build the python interface:

```
[user@host /tmp]$ tar -xvf python-igraph-0.6.5.tar.gz
[user@host /tmp]$ cd python-igraph-0.6.5
[user@host /tmp/python-igraph-0.6.5]$ python setup.py build
[user@host /tmp/python-igraph-0.6.5]$ sudo python setup.py install
```

Alternatively, this can be installed via pip:

```
[user@host /tmp]$ pip install python-igraph
```

Installing pygexf

First, you must install lxml:

```
[user@host /tmp]$ tar -xvf lxml-3.2.4.tar.gz
[user@host /tmp]$ cd lxml-3.2.4
[user@host /tmp/lxml-3.2.4]$ python setup.py build
```

```
[user@host /tmp/lxml-3.2.4]$ sudo python setup.py install
```

Alternatively, this can be installed via pip:

```
[user@host /tmp]$ pip install lxml
```

Finally, you can install pygexf:

```
[user@host /tmp]$ unzip pygexf-master.zip
[user@host /tmp]$ cd pygexf-master
[user@host /tmp/lxml-3.2.4]$ sudo easy_install pygexf
```

3.5.2 Testing

Several test input files are included in the pyLM distribution. To test that pyLM and Lattice Microbes are installed correctly, go to the "src/python/Examples" directory in the source directory and execute the command:

```
[user@host /tmp]$ python example-bimol.py -o cmebimol.lm
```

If no errors are reported, these programs are installed correctly. Next, a test of the correct installation of the other libraries should be performed. Execute the following command in the same directory:

```
[user@host /tmp/]$ python example-bimol-pp.py -o cmebimolpp.lm
```

If all goes well, two plots will be made in the directory named "BimolSpeciesTrace.png" and "BimolSpeciesFit.png". In addition, a graph file named "BimolGraph.gml" will be made in that directory which can be opened with popular network viewing software such as Gephi (`https://gephi.org`) or Cytoscape (`http://www.cytoscape.org`). Representative examples of each of these file can be seen in Figures 3.1, 3.2 and 3.3.

3.6 In case of difficulty

If you experience problems when building the software, email the Lattice Microbes User List:
`latticemicrobes-users@lists.illinois.edu`.
Also, consider joining the Lattice Microbes User List by visiting:
`https://lists.illinois.edu/lists/info/latticemicrobes-users`

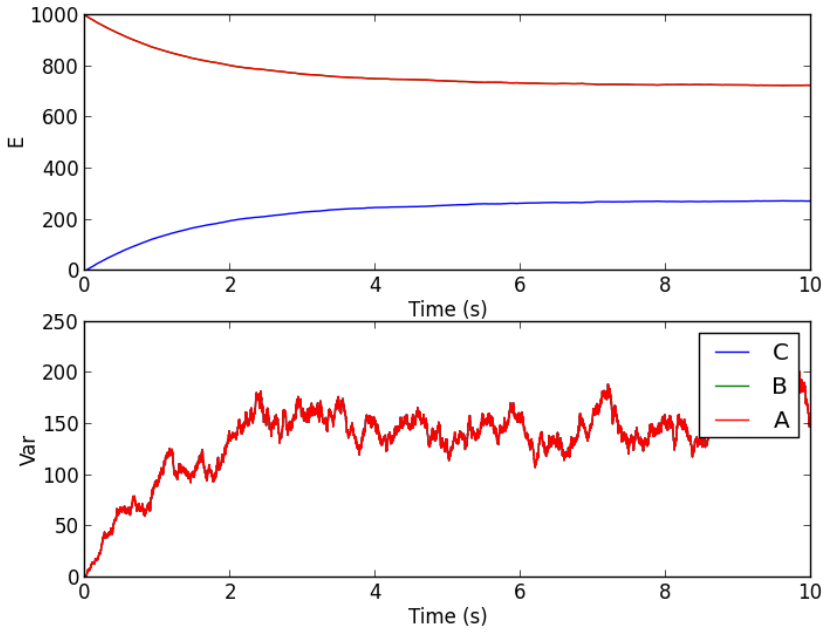


Figure 3.1: Average (top) and Variance (bottom) of the three species over 50 replicates for the reversible bimolecular reaction.

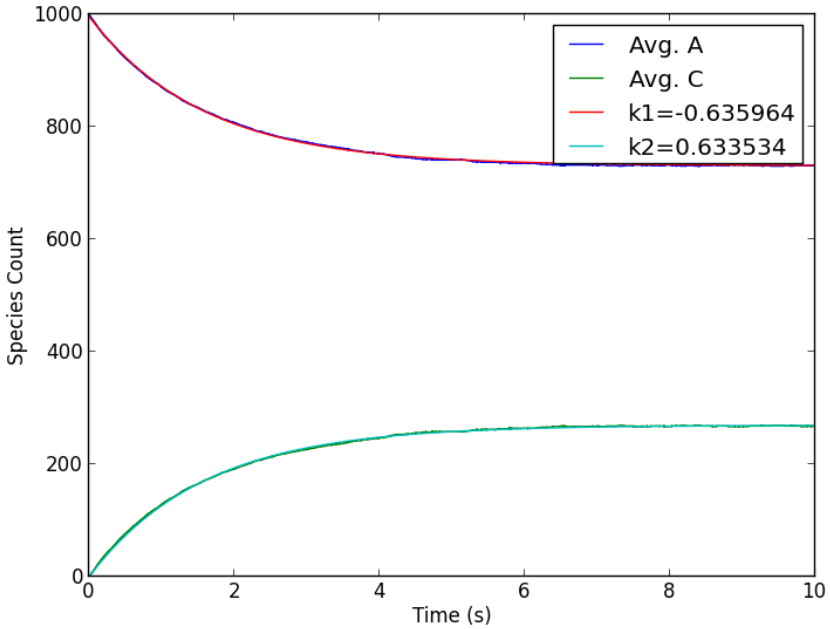


Figure 3.2: The same figure as before, except with fits to the rates performed in Python.

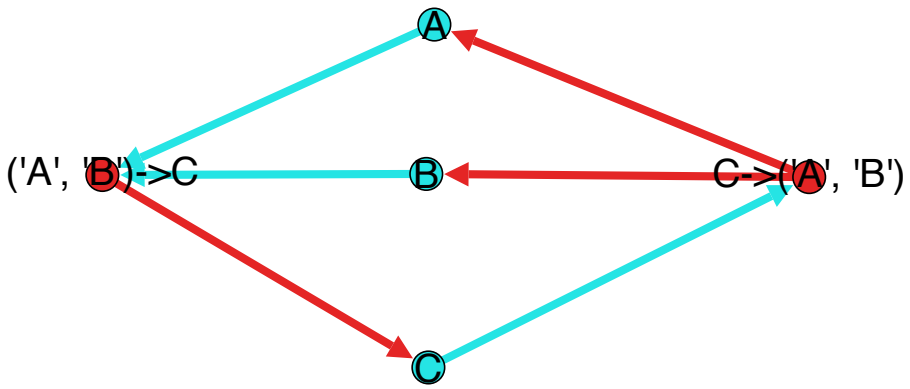


Figure 3.3: The graph of the simple bimolecular reaction. Nodes in cyan are reacting species and red are reactions. Direction of the arrows indicate flow of reactants.

Chapter 4

Examples Simulation Specifications

Many examples demonstrating features of pyLM exist both on the website (`http://www.scs.illinois.edu/schulten/lm/download/lm22/ExampleFiles.tgz`) and in the source code (`src/python/Examples`). Each demonstrates several different features of pyLM, and it is suggested that you read and work through all of the examples before starting to use pyLM for your project. It is recommended that you work through the problems in the order shown, as functionality documented in an earlier file is not described again in later files.
Various functionality is demonstrated in the examples, including:

- CME
    1. example-bimol.py – Demonstrates process of defining molecular species, reactions and initial conditions in a CME simulation
    2. example-bimolConc.py – Same as example-bimol.py using concentrations instead of particle numbers
    3. example-bimol-pp.py – Demonstrates the general trends for writing post-processing code for a CME simulation
    4. example-rnaprotein.py – An example of constitutive gene expression
    5. example-LotkaVolterra.py – An example of the well known Lotka-Volterra problem (predator-prey) simulated with stochasticity
    6. example-stochasticResonator.py – An example of a stochastic resonator problem with CME
    7. example-lac2state.py – This file demonstrates a more complex reaction scheme
- RDME
    1. example-MichaelisMenten.py – This demonstrates how to define a simulation domain in a 3D RDME simulation and defines an enzyme/substrate reaction system to be simulated in the domain
    2. example-minde.py – A demonstration of the popular Min system of *E. coli* showing how to construct default cell shapes, customize diffusion coefficients in regions and diffusion coefficients between regions

3. example-restart.py – This demonstrates how to restart an RDME simulation, however it is a hack as of now

- Advanced Setup

1. example-shapes.py – An example showing how to define complex objects such as boxes, spheres, tori, ellipses and intersections, unions and differences of the various objects
2. example-tightPackedCellArray.py – This example shows a pySTDLM feature that packs a cell shape into a tight regular grid spanning the whole RDME domain
3. example-extendrdme.py – This example shows how to extend the basic RDME solver with a hook that is run on every lattice write, allowing the user to modify the simulation based on the simulation state

Chapter 5

License and Copyright

Bibliography

[1] Roberts, E, Stone, JE, Sepulveda, L, Hwu, WMW, Luthey-Schulten, Z (2009) Long time-scale simulations of *in vivo* diffusion using GPU hardware. *The Eighth IEEE International Workshop on High-Performance Computational Biology*.

[2] Roberts, E, Stone, JE, Luthey-Schulten, Z (2013) Lattice microbes: high-performace stochastic simulation method for the reaction-diffusion master equation. *J. Comp. Chem.* 3:245–255.

[3] Hallock, M, Stone, JE, Roberts, E, Fry, C, Luthey-Schulten, Z (2014) Long time-scale simulations of in vivo reaction diffusion processes on multiple gpus. In Press.

[4] Vigelius, M, Lane, A, Meyer, B (2011) Accelerating Reaction-Diffusion Simulations with General-Purpose Graphics Processing Units. *Bioinform* 27:288–290.

[5] Peterson, J, Hallock, M, Cole, J, Luthey-Schulten, Z (2013) A Problem Solving Environment for Stochastic Biological Simulations. *Proceedings High Performance Computing Networking, Storage and Analysis Companion (SCC)*.

[6] Cole, J, Hallock, M, Labhsetwar, P, Peterson, J, Stone, J, Luthey-Schulten, Z (2013) *Computational Systems Biology*, eds Kriete, A, Eils, R (Academic Press).

[7] Gillespie, DT (1977) Exact stochastic simulation of coupled chemical reactions. *J Phys Chem* 81:2340–2361.

[8] Gibson, M, Bruck, J (2000) Efficient exact stochastic simulation of chemical systems with many species and many channels. *J Phys Chem* 104:1876–1889.

[9] Elf, J, Ehrenberg, M (2004) Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases. *IEE Syst Biol* 1:230–236.

[10] Hallock, MJ, Stone, JE, Roberts, E, Fry, C, Luthey-Schulten, Z (2014) Simulation of reaction diffusion processes over biologically relevant size and time scales using multi-GPU workstations. *Parallel Computing* 40:86–99.