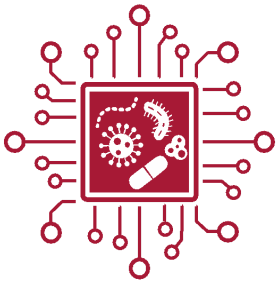
Simulation Library Optimization and Parallelization of Stochastic Simulation Code for Computational Epidemiology

Parallelizing the StochPy Library

Resistance Epidemiology Modeling Initiative
Dr. Eric Lofgren



WASHINGTON STATE UNIVERSITY
Resistance Epidemiology
Modeling Initiative

REMI-HPCStochPy

Michael J McNaughton
Ryan Charit
Gerald Hoff

September 28, 2022

I. Introduction

I.1 Project Introduction

The Resistance Epidemiology Modeling Initiative (REMI) conducts simulations modeling epidemics, particularly in smaller populations and focused on the beginning and end of epidemics. Such models can help with preventing and containing present and future epidemics and as such are of great utility. Most of the major tools for disease modeling in the field are deterministic, which function well for large population sizes. However, since the REMI lab focuses most of their research around smaller populations, a stochastic modeling tool is preferred, as stochastic models are superior for use in small populations [1]. Unfortunately, the tools for such modeling methods are not as highly developed as would be ideal for a research field as important as epidemiology.

As such, our project problem is to create an optimized and parallelized fork of the StochPy library. This library is used by REMI to conduct stochastic simulations in computational chemistry, physics, epidemiology, and ecology. The shortcoming of StochPy is that the stochastic simulation algorithm is slow, especially when trying to perform large computations with upwards of 10 million groups of simulations. However, this algorithm is also embarrassingly parallel, which means that if the codebase were refactored to run the simulations on many cores at once, the amount of time for a job to complete would be drastically reduced [2].

I.2 Background and Related Work

Stochastic simulations are generally performed using a process called the Gilespie direct method, making use of an algorithm popularized by Daniel Gilespie [3]. The algorithm converts the rates at which certain events take place to probabilities that that event will take place. The simulations use the probabilities generated by the algorithm to determine which of a set of potential events take place by drawing from a bag of weighted probabilities for each timestep. In the case of epidemiological simulations, events can include a new individual being infected or an infected individual recovering. An increase in the population size in such simulations also increases the frequency of events occurring and thus increases the computational intensity of the simulations which drastically reduces the speed at which simulations for higher population sizes can be completed.

The current version of StochPy is state-of-the-art at the time of the creation of this report [4]. That is, there currently are no available tools or libraries known to the REMI lab or its associates that outperform the StochPy library for their purposes. If our fork of the library were to demonstrate significant speedup, it would become an incredibly useful asset to the lab and any other labs that work with large-scale stochastic simulation for epidemiological research.

Knowledge and skills we will need to learn in order to complete this project include developing a basic understanding of the functionality of the StochPy library in its current state, as well as Python tools and libraries for increasing the speed of programs through the use of just-in-time (JIT) compilation and parallelization, such as by using the Numba and Dask libraries.

There are libraries that can do stochastic simulations similar to StochPy in parallel, Cayenne [5] being one of them. However Dr.Lofgren has specified that the improvement to StochPy is still warranted as there are some functionalities of StochPy which Cayenne does not include, on top

of it being much more efficient for the many individuals making use of StochPy if the library can be improved rather than needing to shift to another library entirely.

I.3 Project Overview

The Resistance Epidemiology Modeling Initiative is a product of collaboration between the WSU Paul G. Allen School for Global Health and the WSU School of Electrical Engineering and Computer Science. REMI aims to apply advanced computational methods to problems in public health, such as antimicrobial resistance, healthcare-associated infections, and COVID-19. In order to pursue its goals, one of REMI's most frequent activities is running simulations of epidemics centered around small, localized communities of people, such as schools and hospitals. This contrasts with other, more traditional epidemic simulations that focus on large, dense population centers like New York City.

This focus on smaller population centers necessitates the use of stochastic simulation models, as opposed to deterministic models for larger population centers. Deterministic models make use of fractional measurements, handle variance poorly, and are incapable of allowing a simulated disease to go extinct. This is problematic, as nonsensical measurements like one-quarters of an infected nurse in a hospital ward, inability to properly simulate the chances of a disease with a relatively low R_0 value spreading into a small scale outbreak, and inability for a simulated disease to die out are all detrimental when determining if the simulated disease grows into an epidemic or perishes before becoming an issue. However, as population sizes increase, stochastic simulations become more computationally intensive as the number of timesteps between events gets smaller, so running the many simulations necessary for the research with populations of moderate size sequentially even on HPC clusters can take several days.

Indeed, one of REMI's main limitations in conducting its stochastic epidemic simulations is the slow speed at which they are executed, which becomes a glaring issue when considering that upwards of tens of millions of simulations may need to be run for a single research experiment. This is a direct consequence of StochPy, the stochastic simulation Python library in use by REMI, being designed for single-threaded execution. Moreover, StochPy's authors and users have made no public attempt to optimize the library, and other stochastic simulation libraries may not meet REMI's needs or smoothly interface with existing lab codebases and digital infrastructure.

Slow simulation speeds pose a significant issue for REMI. Given that REMI is involved with research surrounding antimicrobial resistance, COVID-19, and other important public health subfields, improving the speed at which simulations are conducted is critical for accelerating potentially lifesaving research, aiding in rapid response to potential outbreaks, and assisting with formulating public policy decision recommendations. Furthermore, sufficiently improving StochPy performance would enable REMI researchers to feasibly run simulations on local devices like laptops and desktop computers, reducing dependency on the Kamiak cluster and allowing for research to be conducted more efficiently and conveniently, which would further increase research output.

The bare minimum intended outcome for the project is to have StochPy demonstrate a fair amount of speedup. This can be accomplished by optimizing the codebase. We can also create a large amount of speedup comparatively easily by running the disparate simulations on individual processors. StochPy, being a stochastic simulation model, calculates the interactions of individual elements in the simulation. This makes it slower than large scale simulation algorithms, but more accurate at smaller scales. These interactions between elements of the

simulation are oftentimes isolated from other interactions between other elements, meaning that these computations are done separately from other groups of computations. These groups are essentially sub-simulations of the larger simulation. These sub-simulations can be done on different cores concurrently, allowing for significant speedup. This is the embarrassingly parallel nature of the algorithm. This could be completed using threads, and it likely would not be that hard to implement. This is the minimum work for us to do.

The parallelization of sub-simulations onto standard non-GPU cores would be the easiest and likely the most effective way to create speedup. However, doing the sub-simulations on GPUs themselves could be promising for creating further speedup. Whether or not having the sub-simulations running on GPUs would be feasible or not is dependent on the architecture of the GPUs, the nature of the algorithm, and the limitations of Python.

Another area where we could see some amount of speedup is with JIT compilation. Using CPython could be beneficial. Likely JIT compilation or another form of compilation would be the only available way to work within the limitations of Python's speed. As rewriting the entire codebase in a language such as Julia or Chapel is unfeasible.

I.4 Client and Stakeholder Identification and Preferences

The primary stakeholder of this project is REMI, which is represented by Dr. Eric Lofgren. At a minimum, REMI's primary need is a speed-up in the stochastic simulation functionality of the StochPy library. This can be achieved through the optimization of the codebase to make use of JIT compilation as well as parallelization of the StochPy library, specifically in regards to its stochastic simulation feature. This parallelized fork should still be capable of operating on existing lab machines and the Kamiak cluster. Stretch goals include adding the ability to employ GPUs to improve simulation speeds, such as through NVIDIA's CUDA framework, as well as the removal of unwanted popup messages accompanying the startup of StochPy.

Student members of the Lofgren Lab, along with other students working under the auspices of REMI, serve as a secondary stakeholder. For them, improving the speed of stochastic simulations is a critical need. An additional preference is improving the speed of StochPy to the point that stochastic simulations run in reasonable timeframes on portable devices like laptops, as opposed to just the Kamiak cluster.

Additionally, government institutions funding REMI's research, such as the National Institutes of Health (NIH), also act as secondary stakeholders for this project. Given the significance of REMI's research, speeding up StochPy would help fulfill NIH mission objectives. Also, it is probable that improving REMI's research capabilities would make it more competitive for future government grant funding.

Other labs and researchers using StochPy serve as indirect stakeholders for our project. Though our team is not in contact with other researchers using StochPy, producing a parallelized fork of the library stands to benefit their research activities as well. Communicating with other research teams may give input as to possible functionalities that a parallel version of StochPy could have. Of course, REMI's goals and requirements are paramount, considering that they are the sponsors of this project.

Finally, those populations which benefit from the research conducted by REMI and other labs using StochPy are tertiary stakeholders, as improvements in the speed of research at such labs could result in lifesaving policy changes and developments. Such populations could include the global community at large in the case of containing and preventing epidemics in their early

stages. More specifically, the staff and patients of hospital wards, staff and students of schools, as well as staff and detainees of jails stand to benefit from REMI conducting faster simulations.

II. Team Members - Bios and Project Roles

Michael McNaughton is a computer science major with interests in cyber security, machine learning, and artificial intelligence. His technical skills include experience with SQL databases as well as basic knowledge of Python and the C language family. His responsibilities for the project include the recording of meeting minutes, keeping the github repository updated with current versions of documentation, and study of user side parallelization implementation tools.

Gerald Hoff is a computer science student interested in HPC, linear programming, and other various topics. His technical skills include Python, C/C++, Java, and knowledge of the CUDA library. For this project his responsibilities include communication with the client, assignment submissions, and leading efforts to refactor the StochPy library for implementation of Numba optimization methods.

Ryan Charit is a computer science major interested in application development, data science, and software design. His technical skills include C/C++, basic Python, and design and implementation experience with SQL. For this project he was tasked with organization and maintenance of the github repository, team coordination and management, as well as documentation formatting and presentation.

III. System Requirements Specification

III.1. Use Cases

It is difficult to provide new specific use cases, as the changes we are to implement are centered around improving existing functionalities in StochPy. Still, general current and anticipated use cases for StochPy are summarized as follows:

REMI researchers frequently need to run stochastic simulations as one of many computational activities needed for their experiments. The StochPy library allows for the sequential execution of these stochastic simulations, albeit inefficiently. As such, to begin with, a major need for REMI researchers is a general speedup in StochPy's baseline sequential execution speed. Even after acceleration with JIT compilation, removal of the welcome message, and other modern Python, REMI researchers should be able to continue using StochPy. This use case can be linked to functional requirements **III.2.1** and **III.2.3**.

Furthermore, building off the prior user story, REMI researchers also want to significantly reduce the time it takes to conduct stochastic simulations, which would allow for StochPy to be more practically used on consumer devices like laptops and desktop computers, along with generally improving research experiment speeds. To achieve this, significant simulation speed increases are beyond what StochPy currently offers. Dr. Lofgren has identified parallelization as an approach to pursue. As such, REMI researchers should be able to perform a parallel stochastic simulation function added to StochPy in the form of DoParallelStochSim. This use case can be linked to functional requirement **III.2.2**.

Dr. Lofgren has expressed an interest in running stochastic simulations on GPUs, taking advantage of their inherently parallel nature to accelerate StochPy simulation speeds even further and make full use of the Kamiak cluster. Consequently, REMI researchers should be able to use the StochPy library to run stochastic simulations on GPUs. This use case can be linked to functional requirement **III.2.4**.

III.2. Functional Requirements

III.2.1. General Speedup using Modern Python Libraries

General Speedup: Our first requirement is to attain a general speedup in the execution of the functions in the StochPy library, which we aim to achieve through the use of modern Python tools for JIT compilation such as Numba, as well as optimizing any code in the library which is inefficiently implemented.
Source: This requirement originated from Dr. Lofgren, who represents REMI, as the primary need of REMI is for the execution of simulations to be completed more quickly. One of the most effective means to facilitate this is to speedup the execution of individual instances in the simulation through the use of JIT compilation. This requirement is tied directly to REMI, as they are the stakeholder which has requested this improvement to the functionality of the library.

However, this requirement is also tied to other researchers who could or do make use of the library for their own simulations, as well as policymakers and policy beneficiaries who would benefit from improvements in the speed of simulations conducted by REMI and similar research institutes.
Priority: Level 0 - Essential and required functionality.

III.2.2. Implement Parallel Stochastic Simulation

Parallel Stochastic Simulation: Our second requirement is to parallelize the DoStochSim function of the StochPy library in order to improve the speed with which simulations can be completed. We aim to achieve this using tools such as Dask or Ray meant for easy parallelization of Python code. The function DoStochSim from the StochPy library must be refactored to run simulations in parallel, and this new implementation must become a usable option in the library.
Source: This requirement originated from Dr. Lofgren, as the primary need of REMI is for stochastic simulations to execute faster. The best way to facilitate this is to parallelize the separate instances in the simulations rather than running them sequentially. This requirement is also tied directly to REMI, as they are the stakeholder which requested such an improvement. Once again, stakeholders like other researchers, policymakers, and policy beneficiaries are also involved, due to parallelization imparting benefits similar to that of the first requirement.
Priority: Level 0 - Essential and required functionality.

III.2.3. Remove Welcome Banner from StochPy

Removal of Welcome Banner from StochPy: Our third requirement is the removal of an unwanted welcome message popup built into StochPy which triggers whenever the StochPy tool is used. Finding and removing the code for generating the welcome banner is straightforward, though adding a configuration file so that the appearance of the message can be left up to the discretion of the users is also an option.
Source: This request ties directly back to Dr. Lofgren of REMI specifically, as he personally uses the tool a great deal and wishes to not have to see the message every time he does so. This would also benefit any members of our listed stakeholder groups, such as REMI and other affiliated researchers, who would also like to not be bothered by such a message.
Priority: Level 1 - Desirable functionality.

III.2.4. Run Stochastic Simulations on GPUs

Ability to Run Simulations on GPUs: Our fourth requirement is the potential addition of the ability to run StochPy simulations on GPUs rather than CPUs to make use of their additional computing power. Such a goal will require the research of the architecture of the GPUs in use by REMI as well as the library and language themselves to check for limitations which may prevent such a task from being completed in the first place.
Source: This request originated from Dr. Lofgren, as the increase in computing power available using GPUs would be a significant boon in the speedup of simulations beyond that achieved by previous version of StochPy and other stochastic simulation libraries. This requirement also ties

directly back to REMI, as they possess the GPUs which would be utilized for the implementation and are once again the source of the request for this functionality. Other researchers are also involved with this requirement, as some may have access to GPUs and would be interested in a GPU-capable version of StochPy.
Priority: Level 2 - Stretch goal.

III.3. Non-Functional Requirements

Ensure Simulation Accuracy
The parallelization of StochPy should not compromise the accuracy of simulation results and should generally match that of the sequential version of StochPy.

Stable and Reliable
Considering that a large number of simulations may be run at a time or over the course of several days, along with the importance of the simulations being run, our StochPy rewrite should not be prone to crashing, freezing, stalling, or other types of instability.

Efficient
Based on the potential complexity and volume of stochastic simulations, our rewrite should be efficient, with CPU cores, main memory, and other computational assets being utilized to their fullest without slowdown originating from inefficient programming practices.

Maintainable
Our rewrite of StochPy should be maintainable, in that it should be well-commented, well-documented, adhere to object-oriented principles, and follow good programming practices. This will aid student programmers and REMI members that may need to modify parallelized StochPy in the future.

Fast
At a minimum, our parallelized fork of StochPy should run stochastic simulations faster than the current sequential version.

Continue to Accept .psc Files as Input
Currently, simulation setup and execution information is stored on preexisting .psc files. Our StochPy rewrite should continue to accept these files for simulation information.

Easy to Use
Ultimately, most users of the stochastic simulation features of StochPy do not have a technical background. There should be no added difficulty in using parallelized StochPy over sequential StochPy.

Compatible with the Kamiak Cluster and other WSU Servers
Since our project scope is largely limited to backend changes, our forked version of StochPy should continue to run on the Kamiak cluster and other capable WSU devices with minimal necessary disruptions.

Written in Python
Given that StochPy is written in Python, our rewrite should still be written in Python.

III.4. System Evolution

Hardware evolution
Changes to the hardware that will be used is unlikely. The REMI lab will be using the project primarily on the Kamiak cluster and personal machines, and Python is a language with which all the current users are already familiar with and are capable of implementing StochPy on said cluster and machines. We do not expect the Kamiak cluster to drastically alter its hardware, either. Since Python is a language capable of being run with little setup on all computer devices, changes in individual personal machine choices should not impact the ability to run StochPy and our changes to it. Improvements in processor and GPU performance would likely not change the way the project would be designed. The only concern would be if we created some sort of CuPy application and the GPUs were replaced with non-NVIDIA GPUs, which is unlikely.

Changing user needs
The current users of the software, the REMI lab, are experienced scientists who are adept at managing the workflow for their stochastic simulations. Their needs are simply to be able to create .psc files and run simulations based on them. As they are a lab focused on simulations for relatively small population sizes using a simulation method similarly specialized for such populations, their needs are not expected to change for the foreseeable future in regards to the necessary utility of this tool, unless their research focus shifts drastically.

Another user base that may emerge is doctors and healthcare professionals who are not expert researchers. Another capstone team is working on creating a frontend application so that these users can run their own simulations. They will likely not have access to the Kamiak cluster or other advanced computing resources, so any speedup they experience will likely not allow for the use of GPUs. This lends further emphasis to the importance of the general speedup and parallelization aspects of the project, as they will provide the most benefit to the greatest number of users.

Programming language limitations
Python is a commonly used language among scientists and researchers. It then makes sense that we are using Python. However, Python is limited in terms of high performance computing. We are in the process of evaluating whether these limitations can be overcome with different compilers and libraries as it relates to StochPy and stochastic simulations. Even if using Python makes the project run slower than it hypothetically could, we do not foresee switching to another language, as this would mean rewriting the entire codebase in addition to the researchers likely not knowing the language used instead of Python.

IV. Project Solution Approach

IV.1. System Overview

The purpose of this document is to create a more in-depth plan for our project. That is, we will go over the design of our system. We will look at a general overview of our proposed system, the architecture of our software, the way our data will be stored, and the user interface.

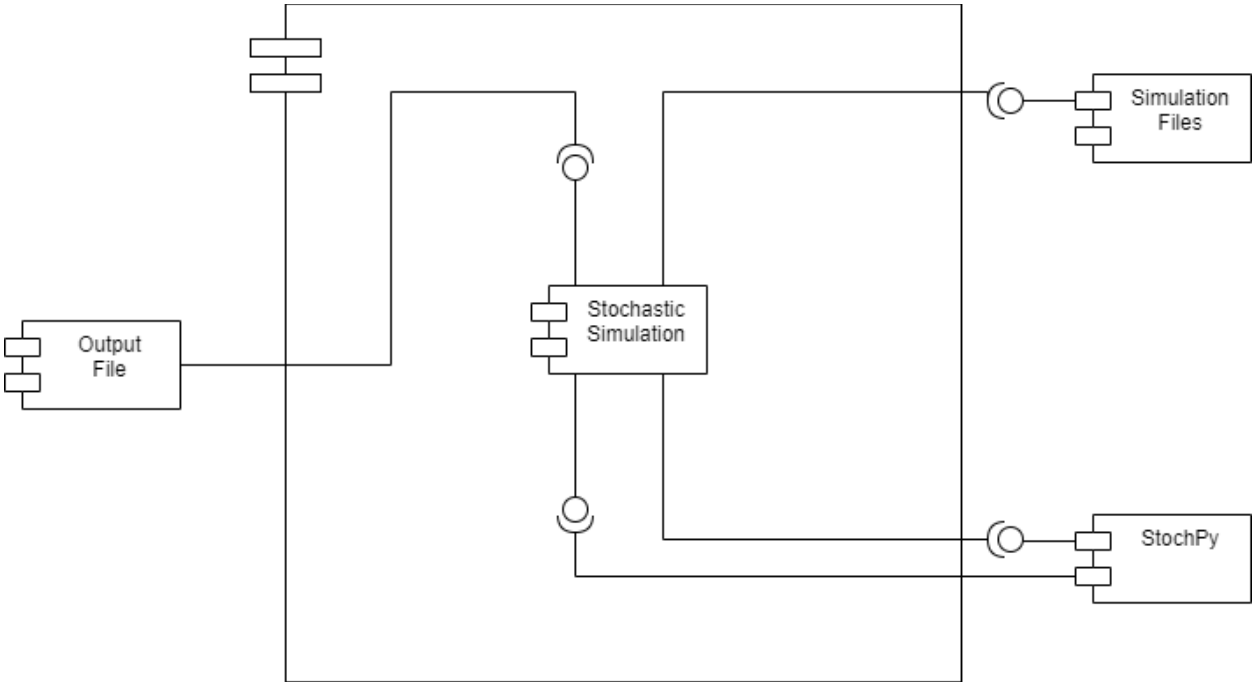
The primary functionality of the product is to upgrade the StochPy library to include a parallelized version of a function which it already has called DoStochSim, as well as implement a general speedup in the execution time of the StochPy library by refactoring code integral to stochastic simulation to make use of JIT compilation and other modern Python features. If time permits, we additionally aim to improve the StochPy library to be able to make use of the computational power of GPUs, as this functionality would further aid in achieving an overall increase in simulation speed which is sought by the client. A much more design focused additional goal of the product is to remove an unwanted pop up from the startup sequence of StochPy.

IV.2. Architecture Design

IV.2.1. Overview

Due to the nature of our project, it is difficult to generate an extensive architecture for the software we are creating, as we are adding new functionality to an existing library rather than building software from scratch. While a proper overview of the architecture for the library itself could be provided, to do so in the context of this report may be considered plagiarism and contrasts with the defined purpose for this section. As such, an extensive architectural overview has been omitted, as our work is much more limited in scope than what would necessitate an overview of the entire library, and to include one with that being the case may be viewed as dishonest and disingenuous. For this reason, we cannot do a decomposition of the systems we are working beyond saying that our parallel implementation of DoStochSim will have the same provided and required services as the original DoStochSim. However, expansions in the scope or changes in the nature of this project may alter our architectural design in the future. Regardless, we do not anticipate REMI or Dr. Lofgren modifying their requirements in the immediate future.

As it stands for our ad-hoc architectural design, the Stochastic Simulation subsystem is the primary one being developed at this point. It is anticipated to receive inputs from the Simulation Files subsystem, which consists of .psc files and any other other functionality that serve to provide parameters for stochastic simulation execution. The Output File subsystem represents the user-accessible “front-end” results generated by the Stochastic Simulation subsystem. Additionally, StochPy itself is represented as a greater subsystem, as to denote that our stochastic simulation improvement efforts are still but a subset of the greater library, and that our changes contribute to the greater StochPy codebase. Potential future changes in scope may alter this reality.



IV.2.2. Subsystem Decomposition

IV.2.2.1 Stochastic Simulation

a) Description

This subsystem encapsulates the implementation of the sped-up sequential, parallelized, and GPU-accelerated stochastic simulation functions.

b) Concepts and Algorithms Generated

Our current efforts are centered around understanding StochPy's stochastic simulation algorithm implementation and analyzing possible methods for speedup and eventual parallelization. Dr. Lofgren has offered potential approaches for parallelization, which we are currently investigating.

c) Interface Description

Services Provided

1. Service Name*:* Improved DoStochSim
   Service Provided To: StochPy
   Description: Improved DoStochSim will implement an improved version of the current DoStochSim function by leveraging optimizations offered by modern Python libraries to accelerate sequential execution. Input values will generally consist of simulation parameters provided by .psc simulation files and any necessary auxiliary StochPy library functions. Output values will be stored in output files, and the function itself contributes to the rest of the StochPy library.

2. Service Name*:* Parallelized DoStochSim
   Service Provided To: StochPy

Description: Parallelized DoStochSim will implement a version of DoStochSim that uses a parallel algorithm to perform stochastic simulations. Input values will generally consist of simulation parameters provided by .psc simulation files and any necessary auxiliary StochPy library functions. Output values will be stored in output files, and the function itself contributes to the rest of the StochPy library.

3. Service Name*:* GPU-Accelerated DoStochSim
   Service Provided To: StochPy
   Description: GPU-Accelerated DoStochSim will implement a version of DoStochSim that uses a parallel algorithm capable of running on GPUs. Input values will generally consist of simulation parameters provided by .psc simulation files and any necessary auxiliary StochPy library functions. Output values will be stored in output files, and the function itself contributes to the rest of the StochPy library.

Services Required

1. Service Name: Provide .psc Simulation Information
   Service Provided By: Simulation Files

2. Service Name: Provide Access to Rest of StochPy Codebase
   Service Provided By: StochPy

IV.4. Data Design

The initial specifications for each simulation are stored in .psc files (pronounced pisces). This data from these files is then used to do the simulation which stores output in .csv files. The StochPy library itself makes use of arrays, unassigned integers, and floats from the NumPy library for data manipulation during the simulations.

IV.5. User Interface Design

Installation

To install StochPy, the user will need to run this command in their Python environment:

```
pip install scipy matplotlib python-libsbml jedi==0.17.2 ipython stochpy
```

For Linux/MacOS instead you can simply run this command in the directory of the cloned repository:

```
sudo python setup.py install
```

Use

The user interface will simply be whatever terminal the program is being run from. To run the application, the user will create a .psc file, storing the specifics of the simulation to be run. The user will then use the following commands to do necessary operations.
Run ipython and import stochpy:

```
import stochpy
```

```
smod = stochpy.SSA()
```

Please see Appendix A.1 for more StochPy commands.

V. Test Plan

V.1. Introduction

V.1.2. Test Objectives and Schedule

Our approach will be to test JIT compilation implementation, parallelization, and GPU usability in the order they are implemented. Major work activities will include the implementation and testing for each improvement. Resources required for the testing will be those available from the client to be specified in later updates as we obtain more information regarding their specifics. Major work activities will be the implementation of comparative performance tests for each improvement to the library, major milestones will be the successful completion of such tests. The documentation and code deliverables related to each incremental improvement planned for the library will also be included with their respective milestones. At this time we do not have a concrete schedule for such milestones but will update this section to include one when we obtain one.

V.1.3. Scope

This document is meant to cover the plans for testing which we will be implementing to confirm that our optimization and parallelization of the StochPy library is successful in achieving a speedup in execution times for the stochastic simulations it is used to perform.

V.2. Testing Strategy

1. Identify the requirements to be tested. All test cases shall be derived using the current

Software Requirements Specification.

2. Identify which particular test(s) will be used to test each module.

3. Review the test data and test cases to ensure that the unit has been thoroughly verified

and that the test data and test cases are adequate to verify proper operation of the unit.

4. Identify the expected results for each test.

5. Document the test case configuration, test data, and expected results.

6. Perform the test(s).

7. Document the test data, test cases, and test configuration used during the testing

process. This information shall be submitted via the revised Test Plan document.

8. Successful unit testing is required before the unit is eligible for component

integration/system testing.

9. Unsuccessful testing requires a bug form to be generated. This document shall describe

the test case, the problem encountered, its possible cause, and the sequence of events

that led to the problem. It shall be used as a basis for later technical analysis.

10. Test documents and reports shall be submitted. Any specifications to be reviewed,

revised, or updated shall be handled immediately.

We will be using Continuous Integration (CI) in our testing.

V.3. Test Plans

V.3.1. Unit Testing

Unit tests will not be necessary for our case as much of our changes will be adding speed up functionality to an existing library, the only testing necessary will be to the execution time of the updated library.

V.3.2. Integration Testing

In a similar manner to unit testing, due to the nature of our project we will not make use of integration tests beyond whether or not the library continues to function correctly when updated.

V.3.3. System Testing

V.3.3.1. Functional testing:

As we do not plan to implement any major changes to the base functionality of the library we do not plan to make use of functional testing at this time.

V.3.3.2. Performance testing:

Performance testing will be the primary focus of our testing as the changes we will be making to the library are focused on improving execution time for simulations using the library. As such we will test each major component implemented to achieve speed-up by comparing the execution time of workloads using the updated library against that of the original serving as a baseline.

The tests will consist of running a simulation with a workload using both the unimproved library and our improved variant and then comparing the execution times to confirm that a speedup was achieved. Hardware specifications will be updated once we have access to specifics of the available resources of the client for testing.

V.3.3.3. User Acceptance Testing:

We will conduct user acceptance testing by demonstrating the functions of our parallelized version of StochPy to Dr. Lofgren as functional requirements are implemented in a manner which he finds satisfactory. Given the focused scope of this project, these demonstrations will generally consist of running StochPy on test workloads. The major factor for user acceptance will be whether our StochPy implementation demonstrates empirical speedup.

V.4. Environment Requirements

While we do not expect to need any special tools in order to conduct our testing. We will likely use the facilities available to the client for testing in general to keep the environment consistent. Updates will be made to further specify what hardware the client is using once we get access to them for testing.

VI. Alpha Prototype Description

VI.1 Stochastic Simulation

VI.1.1 Functions and Interfaces Implemented

We have implemented Dask functionality to parallelize the DoStochSim() method. However this functionality has not yet been integrated with the main branch. We have created a Jupyter Notebook that demonstrates basic usage of the StochPy library and demonstrates its time complexity. We have made progress in refactoring the library in order to prepare it for just-in-time compilation. JIT compilation requires somewhat specific formatting for the code it is optimizing. Remaining work is to finish formatting the library to integrate Numba JIT functionality, which should also allow relatively easy implementation of Numba parallelization functionality. After which we will commence work on understanding what changes may be needed to implement the stretch goal of GPU integration for the library and the variouses options for tools which can be used to implement such functionality.

VI.1.2 Preliminary Tests

As the manner of this project is centered around the improvement of performance of an existing library through implementing functionality of optimization libraries, there are no unit or integration tests necessary aside from whether or not the library remains to correctly run simulations with the enhancements made. As such we have no unit or integration tests as at the level we are working the functionality of the library as a whole is the only element of concern as it is all that should be impacted by our modification. Formal performance tests have not been performed at this time as we have not yet implemented all of the planned improvements to the library. Planned tests for a variety of potential implementations to be implemented are provided in appendix A.2.

VI.2 Messageless StochPy

VI.2.1 Functions and Interfaces Implemented

No new functions or interfaces were implemented as the StochPy library does not make use of one nor suffer from the lack thereof. However there is a message which is displayed with every use of the library that the client found bothersome, said message no longer displays with the enhanced library. This requirement is considered to have met the clients needs and so has no further work planned at this time.

VI.2.2 Preliminary Tests

No formal tests for this feature were implemented as it is evident through the use of the library itself whether or not the message was removed successfully.

VII. Alpha Prototype Demonstration

To start off our presentation Ryan gave a modified version of our class presentation. He covered our progress thus far in studying the StochPy library itself using the materials provided by the client as well as documentation available online and compiling our own study materials, our progress in identifying and modifying the sections of the StochPy library necessary to allow for Numba optimization, and our progresses in developing an understanding of and implementing some of the libraries we have investigated as a means to achieve the intended speedup such as Numba and Dask. He also covered what work is still underway and planned for the future. This includes finishing the changes to StochPy in order to demonstrate speedup through Numba, our plans to test our improvements to demonstrate they achieve significant speedup in a formal environment, and investigating the possibility of implementing GPU parallelization to achieve further speedup for the library. He also presented our planned agenda for the rest of the demo, including a demonstration of the speedup achieved using a user side implementation of Dask and the presentation of our analysis of StochPy.

Michael then presented a demonstration of the significant speedup achieved using the user side implementation of Dask that had been investigated as an alternative means of speedup. He also covered several of the concerns regarding the elements of the implementation that had not yet been tested due to its recent development as a course of investigation, and the likelihood of such untested elements proving a problem being low given the method of implementation employed. He also gave a short tutorial and explanation of the changes which were implemented to the template of the client's simulation in order to implement the speedup using this method. Ryan also mentioned that we are trying to implement Dask parallelization into the StochPy library itself though this is more difficult due to the design of StochPy being somewhat troublesome to work with. Gerald then presented our notes on the difficulties in implementing JIT functionality with StochPy as it is now and what elements of StochPy it is believed need to be changed in order to properly implement the JIT compilation functionality of Numba to achieve speedup as well as what progress has been made in making those changes. Additionally, Gerald presented his findings from investigating the execution time of the StochPy library and its correlation with longer simulation times to wrap up our presentation.

During our demonstration of the implementation of user side Dask parallelization the client had several questions about Dask and how it was going about parallelizing the code. Specifically he inquired as to whether Dask was performing parallelization on the for loops calling the simulation functions used in the template or on the functions themselves declared within the template, and if Dask was able to intelligently determine how many cores are available on a given machine to parallelize across. While we were able to give simplistic and generalized explanation to both of these questions due to our limited knowledge of the library, we were unable to provide more detailed and comprehensive answers as we have not thoroughly researched the library at his point since it was an experimental investigation which begun late in the semester when Numba optimization was proving problematically difficult to implement.

After our demonstration the client made comments and offered some information which Gerald asked for in regards to the difficulty in scaling stochastic simulations. He explained that stochastic simulations are excellent not just for small population models, but for larger ones as well. However, as population sizes grow the simulations become prohibitively slow in their execution times due to the increase in the number of simulated elements and for this reason are not widely used for larger populations. He also expressed that he was pleased with the speedup achieved by the user side Dask implementation and was interested in further investigation of such methods for attaining further speedup. He also explained that he is unsurprised by the

slow progress in refactoring StochPy in order to optimize it as the codebase is known to be difficult to work with. It was specified that while the goal mentioned for the project was to try and improve the library itself, the true goal is simply a speedup in the execution of simulations and any means of achieving speedup is acceptable. So while implementing changes to the StochPy library itself in order to achieve speedup is still a major goal, it has been made clear that further investigation of user side means of achieving speedup is permitted and even encouraged given the success of our limited experimentation with it.

Given this new information and clarification from the client. While the initial design is proving difficult to implement, we will continue for the time being as it is not believed impossible. However, we will also continue our investigation and implementation of user side means of obtaining speedup. Specifically we will formally test and confirm that the integration of Dask parallelization into the template used to run simulations does not result in a loss of functionality and quantify how much speed up it can achieve on the clients machines, as well as further develop our understanding of the Dask library on the whole

VII. Future Work

Our most pressing task next semester will be to finish implementing Numba Jit compilation in the StochPy library itself. Our plan for this is to finish modifying the StochPy library in order to make it compatible with Numba as quickly as possible, and then integrate Numba into the elements of the library we have identified as necessary to optimize the DoStochSim function of the StochPy library.

Once we have completed the implementation of changes to the StochPy library in order to facilitate the use of Numba, we will also implement the parallelization function of Numba to further optimize the library. All that should be required to accomplish this after the implementation of the JIT compilation functionality of Numba is to integrate the parallelization functionality of Numba alongside it as all the other steps to facilitate its use should have been completed during preparation to integrate the Jit functionality.

We also plan to develop a version of the StochPy library which integrates Dask optimization. To accomplish this we will first investigate Dask further to confirm that it is as suited to our task as it appears to be, develop a better understanding of how it works and how best to implement it, and finally integrating it into the elements of the StochPy library needed to parallelize DoStochSim as appropriate.

After we have completed the above tasks we will begin determining what optimization and combinations of optimizations are most effective at achieving speedup. To do so we will compare the speedup obtained by implementation of Numba and Dask optimizations, and determine if Dask parallelization and Numba JIT compilation can be implemented together or if Numba must be implemented exclusively.

Having completed all of the main tasks for the project as laid out above, if there is time remaining in the semester we plan to determine if GPU parallelization is necessary and possible and implement it. To do so we will be studying the available options for integrating GPU parallelization, both the functionality included in Numba as well as other potential options like the CUDA library, we will then go about implementing this element of optimization in the manner we determine most beneficial.

Time permitting, we will also be conducting more thorough testing to determine if there is any difference in the efficacy of the optimization on simulations of differing population sizes and complexity, in particular those which are arbitrarily large by the standards of stochastic simulation. We will do this by compiling various simulations in a jupyter notebook and tracking if the size of simulations changes the proportional speedup in execution using the optimized versions of StochPy with the original serving as a baseline for comparison.

Glossary

CUDA - Proprietary parallel programming platform created by NVIDIA.

CuPy - Open-source Python library for GPU-accelerated code. Interfaces well with NumPy and SciPy.

Graphics processing unit (GPU) - Specialized processing units that can do many small numerical operations at once. They are primarily useful for graphics processing, but can be used in some parallel programming applications.

High-performance computing (HPC) - Encompasses the subfield of computing concerned with performing advanced calculations at extremely fast speeds, especially on computer clusters.

Kamiak cluster - A high-performance computing cluster owned and operated by WSU. Used by REMI to conduct stochastic simulations and other resource-intensive computational activities.

Lofgren Lab - A WSU lab under the leadership of Dr. Eric Lofgren focused on epidemiology and public health.

Parallelization - A programming paradigm where a task is subdivided across processing units so that its subtasks can be run concurrently.

R_0 - For a communicable disease, represents how many people an infected person is expected to infect.

Resistance Epidemiology Modeling Initiative (REMI) - A research initiative established at WSU. Seeks to apply advanced computational methods to problems in public health, such as antimicrobial resistance, healthcare-associated infections, and COVID-19.

Stochastic simulation - A simulation of a system where individual variables change randomly with individual properties.

Gillespie Algorithm - A stochastic simulation algorithm. Used in the StochPy library.

Just-in-time compilation (JIT) - A compilation method that compiles the code exactly at runtime. A combination of ahead-of-time compilation and interpretation.

Appendix

A.1. Sample StochPy Commands

Basic Simulation with the Direct method

smod.DoStochSim(IsTrackPropensities=True)

smod.data_stochsim.simulation_endtime

smod.data_stochsim.simulation_timesteps

smod.GetWaitingtimes()

smod.PrintWaitingtimesMeans()

Do some Plotting

smod.PlotSpeciesTimeSeries()

smod.PlotWaitingtimesDistributions()

smod.PlotPropensitiesTimeSeries()

Write data to a text file

smod.Export2File()

smod.Export2File(analysis='distribution')

smod.Export2File(analysis='distribution',datatype='species')

smod.Export2File(analysis='mean',datatype='species')

smod.Export2File(analysis='std',datatype='species')

smod.Export2File(analysis='autocorrelation',datatype='species')

Show the means from the data of 3-th trajectory

smod.DoStochSim(trajectories=3) # multiple trajectories

smod.data_stochsim.simulation_trajectory

smod.PrintSpeciesMeans()

smod.PrintSpeciesStandardDeviations()

Switch to data from trajectory 1 and show the means of each species

smod.GetTrajectoryData(1)

```
smod.PrintSpeciesMeans()
smod.PrintSpeciesStandardDeviations()
```

Do one long simulation

```
smod.DoStochSim(trajectories=1,end=1000000,mode='steps')
smod.PrintSpeciesMeans()
smod.PrintSpeciesStandardDeviations()
```

Plot the PDF for different bin sizes

```
smod.PlotSpeciesDistributions()
smod.PlotSpeciesDistributions(bin_size=5)  # larger bin size
smod.PlotSpeciesDistributions(bin_size=10) # again a larger bin size
smod.Export2File(analysis='distribution',datatype='species')
```

Usage of the Reload Function: Ksyn = 20, kdeg = 0.2

```
smod.ChangeParameter('Ksyn',20.0)
smod.ChangeParameter('Kdeg',0.2)
smod.DoStochSim()
smod.PrintSpeciesMeans()   # should be ~Ksyn/Kdeg
```

Use another model to show the Interpolation features

```
smod.Model('dsmts-001-01.xml.psc')
smod.DoStochSim(trajectories=1000,end=50,mode='time')
smod.GetRegularGrid(npoints=51)
smod.PlotAverageSpeciesTimeSeries()
smod.PrintAverageSpeciesTimeSeries()
smod.Export2File(datatype='species',analysis='timeseries',IsAverage=True)
```

Test each method for different models:

```
smod.Model('Autoreg.psc')
smod.DoStochSim(trajectories=1,end=1000,mode='steps')
smod.Method('NextReactionMethod')
```

```
smod.DoStochSim(trajectories=1,end=1000,mode='steps')

smod.data_stochsim.species

smod.PlotWaitingtimesDistributions()

smod.Method('FirstReactionMethod')

smod.DoStochSim(trajectories=1,end=1000,mode='steps')

smod.Method('TauLeaping')

smod.DoStochSim(trajectories=1,end=1000,mode='steps')

smod.Model('DecayingDimerizing.psc')

smod.DoStochSim(method = 'Direct',trajectories=1,end=50,mode='time')

smod.DoStochSim(method = 'NextReactionMethod',trajectories=1,end=50,mode='time')

smod.DoStochSim(method = 'FirstReactionMethod',trajectories=1,end=50,mode='time')

smod.PlotWaitingtimesDistributions()

smod.DoStochSim(method = 'TauLeaping',trajectories=1,end=50,mode='time',epsilon=0.03)
# Should outperform all other implementations

smod.PlotSpeciesTimeSeries()

#smod.PlotWaitingtimesDistributions()   # Should give an error

smod.Model('chain500.psc')

smod.DoStochSim(method = 'Direct',trajectories=1,end=10000,mode='steps')

smod.DoStochSim(method =
'NextReactionMethod',trajectories=1,end=10000,mode='steps') # should outperform the
direct method and all other implementations
```

Use the Next Reaction Method to test a model with a time event

```
smod.Model('dsmts-003-03.xml.psc')

smod.DoStochSim(method = 'NextReactionMethod')

smod.DoTestsuite()
```

Use the First Reaction method to test a model with a concentration event

```
smod.Model('dsmts-003-04.xml.psc')

smod.DoStochSim(method = 'FirstReactionMethod')

smod.DoTestsuite()
```

Volume Models

```
smod.Model('dsmts-001-11.xml.psc')
smod.DoStochSim(method = 'Direct',trajectories=1000,end=50,mode ='time')
smod.PrintAverageSpeciesTimeSeries()
```

A.2 Test Cases

OriginalVsDaskTemplate:
Aspect Tested: Comparing the speed of simulation execution when using the Dask implementation compared to the template the client currently uses for running simulations.
Expected Results: The simulations run using the Dask template are completed significantly faster than those run using the original.
Observed Results: Not available as the test has not yet been performed.
State of test: Untested as the development of this line of optimization occurred late into the semester and is not ready for formal testing at the time of this report.

OriginalVsNumbaJIT:
Aspect Tested: Comparing the speed of simulation execution when using the version of StochPy with Numba JIT compilation integrated into the DoStochSim function and associated helper functions to that of the original.
Expected Results: Simulations run using the improved StochPy library are completed significantly faster than those run using the original. Speedup here would be the result of decreased compilation time and more efficient numpy data types.
Observed Results: Not available as the test has not yet been performed.
State of test: Untested as implementation of this improvement to the library is not completed at the time of this report.

OriginalVsDaskParallel:
Aspect Tested: Comparing the speed of simulation execution when using the version of StochPy with Dask parallelization integrated to that of the original.
Expected Results: Simulations run using the improved StochPy library are completed significantly faster than those run using the original. With speedup scaling with the parallelism in the specific architecture.
Observed Results: Not available as the test has not yet been performed.
State of test: Untested as implementation of this improvement to the library is not completed at the time of this report.

OriginalVsNumbaParallel:
Aspect Tested: Comparing the speed of simulation execution when using the version of StochPy with Numba parallelization integrated to that of the original.
Expected Results: Simulations run using the improved StochPy library are completed significantly faster than those run using the original. Speedup would be due to more efficient numpy data structures, lower compilation time, and would scale with the parallelism of the machine's architecture.
Observed Results: Not available as the test has not yet been performed.
State of test: Untested as implementation of this improvement to the library is not completed at the time of this report.

DaskParallelVsNumbaParallel:
Aspect Tested: Comparing the speed of simulation execution when using the version of StochPy with Numba parallelization integrated to that of the version with Dask parallelization integrated..
Expected Results: Not certain. This test will be used to help determine which of the two implementations to use in the final release of the updated library. However, it is likely that

Observed Results: Not available as the test has not yet been performed.
State of test: Untested as implementation of these improvements to the library are not completed at the time of this report.

OriginalVsNumbaGPU:

Aspect Tested: Comparing the speed of simulation execution when using the version of StochPy with Numba GPU parallelization integrated into the DoStochSim function and associated helper functions to that of the original.
Expected Results: Simulations run using the improved StochPy library are completed significantly faster than those run using the original.
Observed Results: Not available as the test has not yet been performed.
State of test: Untested as implementation of this improvement to the library is not completed at the time of this report.

DaskParallelVsNumbaTotal:

Aspect Tested: Comparing the speed of simulation execution when using the version of StochPy with all baseline Numba optimizations integrated to that of the version with Dask parallelization integrated.
Expected Results: The Numba based implementation is expected to be faster as it will be benefiting from both JIT compilation and parallelization, while the Dask implementation will only be making use of parallelization. This test will be used to help determine which of the two implementations to use in the final release of the updated library in the event that Numba JIT compilation cannot be used alongside Dask parallelization.
Observed Results: Not available as the test has not yet been performed.
State of test: Untested as implementation of these improvements to the library are not completed at the time of this report.

OptimizedOriginalVsOptimizedDaskTemplate:

Aspect Tested: Comparing the speed of simulation execution when using the version of StochPy with the optimization integrations deemed best due to previous tests using the Dask template to that of the optimized version of StochPy using the template the client currently uses.
Expected Results: The simulations run using the Dask template are completed significantly faster than those run using the original.
Observed Results: Not available as the test has not yet been performed.
State of test: Untested as implementation of these improvements to the library are not completed at the time of this report.

References

[1] E. T. Lofgren, “Pools versus Queues: The Variable Dynamics of Stochastic ‘Steady States,’” *PLOS ONE*, vol. 10, no. 6, p. e0130574, Jun. 2015, doi: 10.1371/journal.pone.0130574.

[2] A. M. Ridwan, A. Krishnan, and P. Dhar, “A Parallel Implementation of Gillespie’s Direct Method,” in *Computational Science - ICCS 2004*, Berlin, Heidelberg, 2004, pp. 284–291. doi: 10.1007/978-3-540-24687-9_36.

[3] D. T. Gillespie, “Exact stochastic simulation of coupled chemical reactions,” *J. Phys. Chem.*, vol. 81, no. 25, pp. 2340–2361, Dec. 1977, doi: 10.1021/j100540a008.

[4] T. R. Maarleveld, B. G. Olivier, and F. J. Bruggeman, “StochPy: A Comprehensive, User-Friendly Tool for Simulating Stochastic Biological Processes,” *PLOS ONE*, vol. 8, no. 11, p. e79345, Nov. 2013, doi: 10.1371/journal.pone.0079345.

[5] D. Kishore and S. Chandrasekaran, “cayenne : Python package for stochastic simulations,” *Read the Docs*, 2020. [Online]. Available: https://cayenne.readthedocs.io/en/latest/#cayenne-python-package-for-stochastic-simulations. [Accessed: Sep. 12, 2022].