

**Figure 2. Data-oriented process**

fashion. The Grid Interface for Parameter Sweeps and Exploration (GIPSE) toolset offers an alternative interface to the grid that is specifically tailored to simulation-based research. In short, GIPSE sits on top of existing grid tools and employs XML metadata to link the service-centric nature of the grid with the data-oriented nature of the researcher. GIPSE does not change the execution of the grid but rather transforms the view of the grid through the use of the XML metadata, as shown in Figure 2. Thus, the researcher can still interact with the grid using traditional tools in a task-centric view with GIPSE-linked tasks simply appearing as normal tasks.

The remainder of our paper is organized as follows. We first give a brief overview of existing tools to meet modern grid problems in Section 2. Next, in Section 3, we motivate GIPSE through examples from the domains of network simulation and bio-complexity. The philosophy of GIPSE and its approach to these problems is described in Section 4 and, finally, Section 5 offers several concluding remarks.

## 2 Modern Parameter Sweep Tools

A very common use of the grid is to provide a solution to a so-called “embarrassingly parallel” problem. In this problem, one has a set of data points to compute, each of which is generated by the same basic simulation with a few different input parameters and none of which depends on the intermediate output of another task, implying no communication among tasks. There are a great many uses for this model by researchers in a variety of disciplines when conducting experiments

by simulation. Examples include simple cases where a range of random number seeds should be input to a simulation, or one of the more complex case studies discussed below. The size of the user pool and the inherent complexity of the grid as a computing resource have led to the design and construction of a variety of tools that implement a solution to the above parallel problem.

<b>Nimrod scripts:</b>
This file runs sim with inputs {0, 1, 2}.
parameter x from 0 to 2 step 1 task main node:execute sim \$x end task
Set up a compute resource and submit.
nimrod generate sim.pln nimrod resource computer.edu nimrod portalapi addrun sim G nimrod addserver sim computer.edu nimrod portalapi startexp sim

**Figure 3. Scripts for Nimrod (simplified)**

One such tool, named Nimrod/G [3], provides several basic services to grid programmers through a simple scripting tool and a set of shell programs. To prepare a set of tasks for execution in Nimrod/G, the researcher writes a script to specify the variable parameters and the list of commands to be executed, which may include node-to-node file copies, substitutions, and other programs. The provided tools must then be used to build up a database of computational resources, connecting these to the task. Nimrod/G builds up a task list by varying the parameters in the domain specified by the user, and the user then executes the task list, and may observe its progress by examining the database. An additional tool that builds upon Nimrod/G is Nimrod/O, which provides more advanced functionality for parameter optimization. An example use of Nimrod is shown in Figure 3.

A different approach to the same problem is taken by the AppLeS Parameter Sweep Template (APST) [6]. In this framework, the researcher must know all the tasks and parameters in advance, or produce this information by a separate script. This information is written into an XML file that provides the input for APST. The XML file also contains the requested computational and storage resources, input and output files, and other system information. The user then executes the APST client, which automatically executes the tasks on the various resources, copying files as necessary. An example use

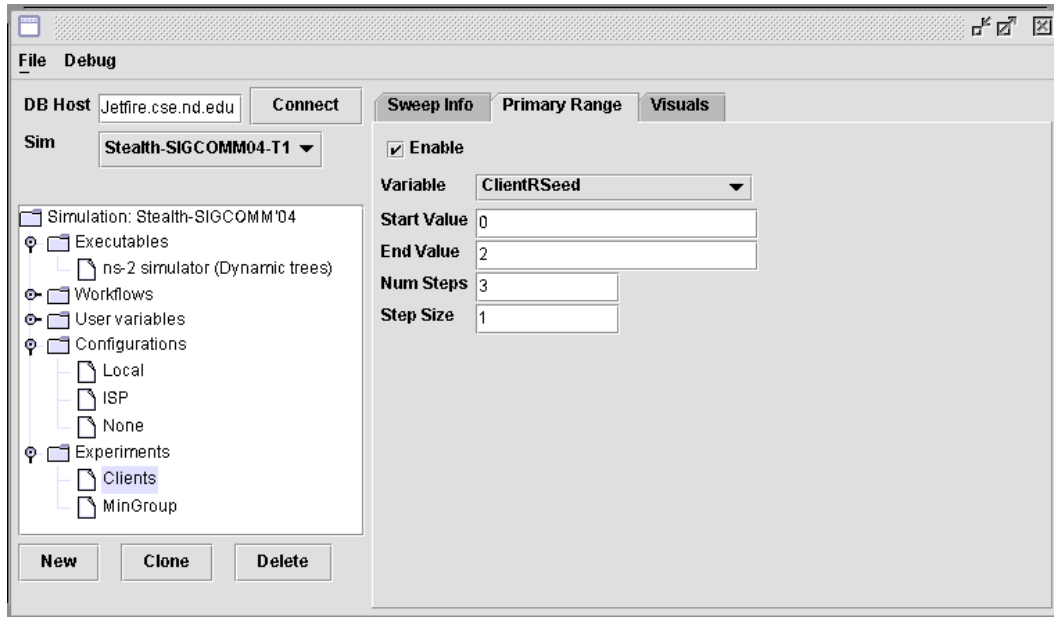


Figure 5. A GIPSE client performing a small parameter sweep

<b>APST scripts:</b>
This file sets up a compute resource and runs <code>sim</code> with inputs <code>{0, 1, 2}</code> .
<pre>&lt;apst&gt; &lt;compute&gt;&lt;host id='compute'&gt;   &lt;ssh server='compute.edu' /&gt; &lt;/host&gt;&lt;/compute&gt; &lt;tasks&gt;   &lt;task executable='sim'     arguments='0'&gt;   &lt;task executable='sim'     arguments='1'&gt;   &lt;task executable='sim'     arguments='2'&gt; &lt;/tasks&gt; &lt;/apst&gt;</pre>
Submit.
<code>apstd sim.xml</code>

Figure 4. Scripts for APST (simplified)

of APST is shown in Figure 4.

Perhaps the most important weakness in these software models is that they have no knowledge of the data produced by the individual runs. Thus, the data can not be used during run time to control the simulation environment. In contrast, our solution intends to provide a common data storage to allow for control of the overarching result. The user, with the resulting data set in mind, can set up the parameter sweep or search in a data-driven way, rather than a task-driven way. Figure 5 demonstrates a small parameter sweep in GIPSE. The left panel shows how a complete data run is specified with its configurations, and the simulation parameters are set in the right panel. A summary of differences between the existing tools and the GIPSE design is summarized in Table 1.

### 3 Case Studies

To better motivate the need for GIPSE, we describe two modern simulation applications that are currently managed by *ad hoc* grid tools. We then demonstrate how GIPSE would provide a useful, extensible and scalable solution for managing the simulation environment.

Feature	Nimrod	APST	GIPSE
Emphasis	Tasks	Tasks	Result
Scripting	Yes	Yes	None
Efficiency	None	None	Automatic
Deadline-Driven	Budget	None	Built-in
Data-Driven	No	No	Built-in
User Steerable	No	No	Automatic
Inter-operability	None	None	XML
Data Knowledge	None	None	General
Data Management	None	None	Built-in
Database queries	None	None	Built-in
Regression Testing	None	None	Built-in
Parameter Sweep	Built-in	None	Built-in
Parameter Search	Built-in	None	Built-in

**Table 1. Summary - GIPSE Toolset Improvements**

Characteristic	Networks	Biology
Application	ns-2 [10]	ProtoMol [11]
Run Time Avg.	30 min.	7,200 min.
Arguments	35+	15+
Parameters	7-12	10
Output Fields	100+	10+
Tasks / Data Run	1-1.5k	1-2k
Generated Data	200-500MB	10-100GB

**Table 2. Case Study Characteristics**

### 3.1 Network Simulation

Network simulation is currently a rich area of new research, algorithms, and software. This example is gleaned from ongoing work by one of the authors [9] that exemplifies the typical network simulation environment, in which a large amount of parameterized tasks are submitted to a grid engine by use-once scripts. However, as will be shown shortly, the gap between the existing set of tools and the actual required interfaces for utilizing the grid can be quite significant and time consuming to address.

Table 2 summarizes the characteristics associated with a typical data run. For each step along the process, scripts and user interactions were employed to create the simulation results. Currently, Perl is used to generate all the task command lines and input files by sweeping over the target parameters. The tasks are then submitted to

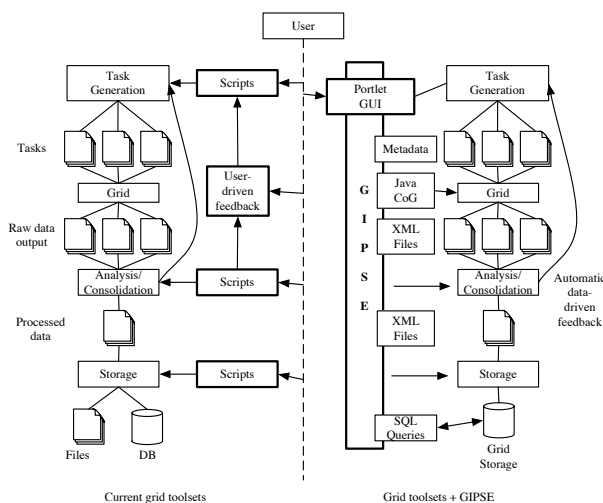
the grid by the script. The user may monitor the submission and completion of tasks by other scripts and shell tools. A final set of scripts collects the relevant data and builds the output plots, and a directory hierarchy is created to archive results. Often, this procedure must be repeated to correctly position the parameter sweep or fix scripting errors.

While the above scripts are sufficient to allow for the use of grid functionality, the scripts suffer from several key weaknesses:

- *Task estimation:* Although the use of task runtime extrapolation does allow for a rough estimated completion time, the results are often skewed to be lower than actually required due to the heterogeneity of task execution length.
- *User steerability:* Unless the tasks are appropriately organized in advance for individual tests, it is difficult to remove tasks to hasten the completion time. Since the grid engine only records tasks, one must rely on the current queue of the grid engine or *a priori* organization to determine which tasks to remove. This problem is only further compounded in the event of a busy grid in which a user's tasks are interspersed with other users' tasks.
- *Dataset management:* The use of scripts and wildcards does not lend itself to easily mitigating the effect of incomplete datasets. In the event that the user does steer the results, partial datasets must be purged manually or through the development of additional scripts.

In contrast, GIPSE offers a distinctly different perspective for the simulation. To start, all of the typically necessary scripts for submitting tasks to the grid or parsing the data are removed. The interface to the simulation is governed by a list of arguments that are based on a list of user-defined variables. In fact, GIPSE shares many of the properties of scripts. However, GIPSE removes the minutia of script programming in a simple, intuitive interface. Furthermore, unlike the previous case with multiple *disjoint* scripts, the definitions of variables and arguments are used throughout the entire simulation process.

Once the user has defined the basic interface points between the simulation and GIPSE, the user works from the perspective of the final visualization, not tasks or scripts. This process is further streamlined as the names of what is available to vary and plot are already available from the earlier setup. For example, the user selects what inputs to vary, the configurations needed, and



**Figure 6. GIPSE vs. existing tools**

what output values to plot. For the researcher, the setup process now deals with the desired end results. Figure 6 shows how GIPSE handles the total data run from start to finish.

Upon submission of a simulation suite, GIPSE automatically creates the underlying scripts and interacts with the chosen grid interface such as Globus via JavaCoG [7] or Condor-G [2]. Job control is unified as GIPSE presents an interface for monitoring and modifying the data run progress from a scientific perspective.

Furthermore, GIPSE can assess a much more accurate estimated completion time. Beyond using historical or benchmarked runs, GIPSE can infer changes in computational length due to changes in inputs since GIPSE knows under what parameters a task was generated. Thus, GIPSE can offer an effective deadline driven service driven by actual results, not by user guesswork. Finally, GIPSE assists with the simulation process by automatically parsing and analyzing the results and producing the desired graphs. The data storage is handled by GIPSE with the final statistical analysis or visualization presented to the user.

### 3.2 Bio-Complexity

Another research topic that is generating a great deal of new software is computational biology and bioinformatics. The advent of massive amounts of biological information brought by the human-genome and associated projects requires ever increasing amounts of computational power and storage to efficiently analyze the

data. For instance, molecular dynamics has been pursued with stochastic Monte Carlo steps (Hybrid Monte Carlo), cf. [12–15]. With its inherent complexity, many optimizations to the method are possible, and such optimizations typically involve the tuning of several parameters. The process of automatic optimization of algorithms and run-time software for the above cases requires extensive testing of the algorithm space and encapsulation of rules for guiding the selection of algorithms and parameters.

In general, the method developer needs to characterize the parameter space and the performance metrics that will be monitored. This type of problem usually involves a portfolio of scripts that manage a complicated set of input parameters that may be based on partial results of previous runs, and also must manage metadata and storage of the large output files. Statistics for typical computations are shown in Table 2. Current users of ProtoMol use a combination of awk, sed, and shell scripts to create a large directory hierarchy which combines input files and output files. Parameter sweeps are performed through shell loops, jobs are queued by the shell, and the data is stored in directories named according to the various metadata attributes of the data runs. Post-analysis is then performed by awk scripts.

Although the current methods are sufficient to develop the ProtoMol software and investigate new algorithms, users find that scripts are rarely reusable from job to job, so scripts and costly output data is not easily shared. Generally speaking, ProtoMol users face the same difficulties discussed in Section 3.1; task completion time cannot be estimated, there is no available information about running tasks. Data runs could be steered by complex scripts but this is not done in practice. Finally, the output data sets are not easily accessed.

GIPSE would provide users of ProtoMol a variety of techniques to manage data runs from a software perspective as well as a parameter search perspective. First, the configurations and variable lists that are specified for any ProtoMol/GIPSE run would be compatible with any other ProtoMol/GIPSE run. Second, complex parameter sweeps and searches and run time task selection and specification are not realistically possible with the current method. GIPSE would provide pluggable sweep algorithms and stop conditions to provide runtime aware controls of the tasks in the sweep.

As a practical example, a current ProtoMol user submits a parameter sweep over a variable that represents an error tolerance in an energy computation, and desires to observe the effect on total run time of the simulation. Each parameter value is determined in advance, though

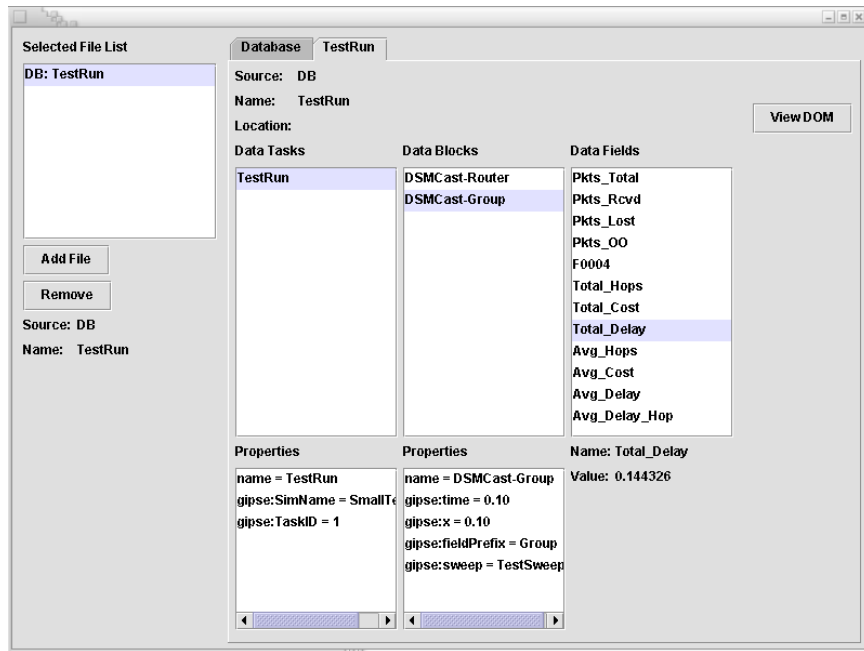


Figure 7. GIPSE data in a GUI client

all are between 0 and 0.2. As above, complex scripts and directory structures are used to run the sweep and extract the relevant output data, which is the runtime of the simulation. In GIPSE, however, once the input parameters and output data are known to GIPSE, sweeps could be phrased “Plot as many values as possible before 8 A.M. tomorrow morning, running up to 10 simulations in parallel.”

## 4 GIPSE Feature Overview

In this section, we describe the fundamental GIPSE philosophy for managing simulation environments and data runs.

### 4.1 Common Data Format

For most existing grid-friendly applications, solutions that are aware of the underlying data are application-specific. In the absence of a standardized base format, the tools can become time consuming to develop and maintain. In contrast, the foundation for the basic and advanced features of GIPSE begins with the use of a standardized XML format that is not application-specific, as illustrated in Figure 8. GIPSE

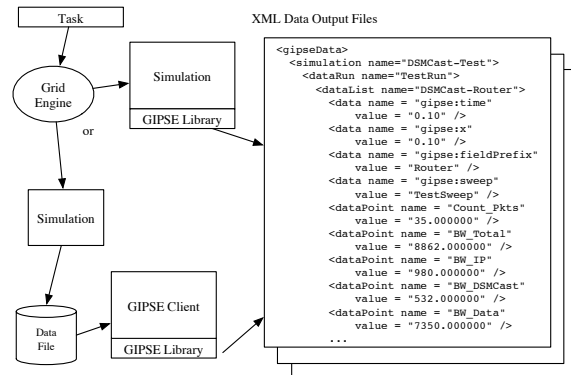


Figure 8. Example GIPSE XML output

offers a consistent and open interface for storing, analyzing, and gathering data.

The actual conversion process to the GIPSE XML format can be accomplished in one of two fashions. First, a GIPSE library can be directly included in the simulation and used for statistical data gathering. In this case, the inter-operability with GIPSE is seamless as the library contains appropriate output and submission functionality. In the second option, the user writes a client that incorporates the GIPSE library to convert the customized format to GIPSE.

## 4.2 Intelligent Task Submission

Since GIPSE can understand and parse the underlying data, GIPSE can offer a host of features not previously available on a generic basis. The ability to be aware of the data has implications for not only data mining but also for dynamically driven simulations, such as simulations driven to fill a search space or minimize a cost function. Furthermore, GIPSE can provide statistically-driven simulations or intelligently steered simulations, which is a significant improvement over the cycle of user guesswork from before.

Whereas the workload for task generation with current grid toolsets is left to the user and typically done in an application-specific manner, GIPSE handles all aspects of task generation in an application-agnostic manner. Due to the fact that GIPSE is targeted towards parameter-driven simulations, an interface is presented that lets the user select which variables to alter for the simulation. Based on the selected sweeps, the GIPSE server will generate the appropriate tasks and submit those tasks to the grid. GIPSE then adds the associated metadata to link the progress of individual tasks to the overall progress of the data run for the simulation.

Another benefit of GIPSE with regard to task submission is the approximation of completion time. As many simulation researchers know, better estimates of completion time allow the researcher to react quicker to the underlying impact of the computation. The fact that GIPSE is aware of the inputs that contributed to a specific task and its respective performance allows GIPSE to more quickly arrive at the correct estimation for data run completion. GIPSE can calculate the estimated performance of future tasks by including a weighting of the relative impact of parameters rather than a blind average.

A secondary application of the increased accuracy of task estimation would be better scheduling and in heterogeneous resource environments. Since GIPSE understands the relative impact of parameters on resource

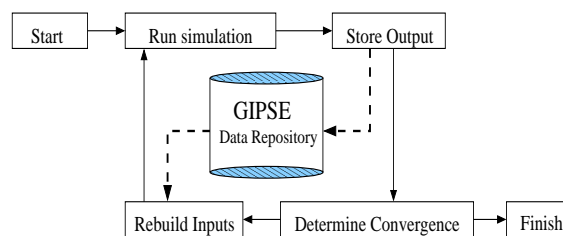


Figure 9. The GIPSE data repository

consumption, it can schedule tasks with more precise resource requirements. A more complete analysis of parameterized task time estimation in GIPSE will be discussed in a future paper.

## 4.3 GIPSE Data Repository

The final portion of GIPSE lies with the GIPSE data repository. The GIPSE data repository completes the entire abstraction of management for grid computing by virtualizing how the resulting data is stored on the system. GIPSE offers two important qualities, namely *organization* and *traceability*. GIPSE provides organization through its global namespace, and traceability by automatically recording useful information about how the data was produced. To the user, it simply appears that GIPSE is a giant database holding all of the various data iterations that have been produced by the simulations.

In Figure 9, a generic simulation is shown performing a parameter search until a certain statistical property is reached. The diagram is greatly simplified; it sequentially runs one simulation at a time. The solid lines represent event propagation from the completion of a task, and the dotted lines represent data flow filtered through the GIPSE Library or a GIPSE client. The output from each simulation is stored in the repository, where it may be recovered later to automatically generate input parameters for a new simulation. This powerful GIPSE feature will drive future work in the area of simulation parameter optimization by allowing for the use of hot-pluggable control algorithms.

## 5 Summary

In summary, GIPSE offers a new approach for the management of simulation-oriented grid computations. Rather than fundamentally changing the underlying grid engines, GIPSE abstracts the interactions with the grid

engine to present a research-centric view of computation rather than exposing the service-centric view. Table 1 presents a summary of the numerous improvements offered by GIPSE. Through its flexible XML-based interactions, the GIPSE package removes the complexity of managing grid software and offers the potential to dramatically lower the barriers to simulation-oriented research taking greater advantage of grid computing. Thus, we feel that GIPSE offers significant benefit and is a compelling platform for future development.

## Acknowledgements

The authors would like to thank Gautam Shewakramani, Brian Bien, Kevin McCusker, Christopher Picardo and James Gentile for assisting with the initial work on the GIPSE prototype toolset. We also acknowledge Matt Shorts and Erich Stuntebeck for contributing to the GIPSE project. Thanks also to Paul Brenner for input concerning the use of ProtoMol.

## References

- [1] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *Intl J. Supercomputer Applications*, vol. 11, 1997.
- [2] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-G: A computation management agent for multi-institutional grids," *Cluster Computing*, vol. 5, pp. 237–246, 2002.
- [3] D. Abramson, J. Giddy, I. Foster, and L. Kotler, "High performance parametric modeling with Nimrod/G," in *Proceedings of the International Parallel and Distributed Processing Symposium*, May 2000.
- [4] A. Natrajan, M. A. Humphrey, and A. S. Grimshaw, "Capacity and capability computing in Legion," in *International Conference on Computational Science*, May 2001.
- [5] S. Malaika, A. Eisenberg, and J. Melton, "Standards for databases on the grid," *SIGMOD Rec.*, vol. 32, no. 3, 2003.
- [6] W. Li, R. Byrnes, J. Hayes, V. Reyes, A. Birnbaum, A. Shabab, C. Mosley, D. Pekurovsky, G. Quinn, I. Shindyalov, H. Casanova, L. Ang, F. Berman, M. Miller, and P. Bourne, "The Encyclopedia of Life Project: Grid Software and Deployment," *Journal of New Generation Computing on Grid Systems for Life Sciences*, 2004.
- [7] Gregor von Laszewski, Jarek Gawor, Peter Lane, Nell Rehn, Mike Russell, and Keith Jackson, "Features of the Java Commodity Grid Kit," *Concurrency and Computation: Practice and Experience*, accepted.
- [8] D. Gannon, G. Fox, M. Pierce, B. Plale, G. von Laszewski, C. Severance, J. Hardin, J. Alameda and Ł. M. Thomas, and J. Boisseau, "Grid portals: A scientist's access point for grid services (draft 1)," *Global Grid Forum*, Sept. 2003, Work in Progress.
- [9] A. Striegel, "Stealth multicast: A catalyst for multicast deployment," in *Proc. of IFIP Networking*, Athens, Greece, May 2004.
- [10] "UCB/LBNL/VINT Network Simulator - ns (version 2)," Available at [www.mash.cs.berkeley.edu/ns/](http://www.mash.cs.berkeley.edu/ns/).
- [11] Thierry Matthey, Trevor Cickovski, Scott Hampton, Alice Ko, Qun Ma, Matthew Nyerges, Troy Raeder, Thomas Slabach, and Jesús A. Izaguirre, "Protomol, an object-oriented framework for prototyping novel algorithms for molecular dynamics," *ACM Transactions on Mathematical Software*, vol. 30, no. 3, Sept. 2004.
- [12] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, "Hybrid Monte Carlo," *Phys. Lett. B*, vol. 195, pp. 216–222, 1987.
- [13] Jesús A. Izaguirre and Scott S. Hampton, "Shadow hybrid Monte Carlo: An efficient propagator in phase space of macromolecules," *Journal of Computational Physics*, vol. 200, no. 2, pp. 581–604, 2004.
- [14] Q. Ma and J. A. Izaguirre, "Targeted mollified impulse — a multiscale stochastic integrator for long molecular dynamics simulations," *Multiscale Modeling and Simulation*, vol. 2, no. 1, pp. 1–21, 2003.
- [15] George S. Fishman, *Monte Carlo: Concepts, Algorithms, and Applications*, Springer Series in Operations Research. Springer-Verlag, New York, 2000.