December 2020

Accelerated Molecular Dynamics for the Exascale

Andrew Garmon
Clemson University, agarmon@g.clemson.edu

Accelerated Molecular Dynamics for the Exascale

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Physics

---

by
Andrew Garmon
December 2020

Abstract

A range of specialized Molecular Dynamics (MD) methods have been developed in order to overcome the challenge of reaching longer timescales in systems that evolve through sequences of rare events. In this talk, we consider Parallel Trajectory Splicing (ParSplice) which works by generating large number of MD trajectory segments in parallel in such a way that they can later be assembled into a single statistically correct state-to-state trajectory, enabling parallel speedups up to N, the number of parallel workers. The prospect of strong-scaling MD is extremely enticing given the continuously increasing scale of available computational resources: on current peta-scale platforms N can be in the hundreds of thousands, which opens the door to MD-accurate millisecond-long atomistic simulations; extending such a capability into the exascale era could be transformative. In practice, however, the ability for ParSplice to scale increasingly relies on predicting where the trajectory will be found in the future. With this insight in mind, we develop a maximum likelihood transition model that is updated on the fly and make use of an uncertainty-driven estimator to approximate the optimal distribution of trajectory segments to be generated next. In addition, we investigate resource optimization schemes designed to fully utilize computational resources in order to generate the maximum expected throughput.

Table of Contents

List of Figures

Glossary

boost multiplicative factor of increased performance or ratio of AMD-time:MD-time. xii

decorrelation time **(τ_c)** the required time spent within a state before accumulating MD-time; the time required for the trajectory to relax toward the state's QSD and become memory-less. xiii

optimal policy the policy corresponding to the maximum cumulative expected utility. lviii

policy a rule by which actions are taken. lviii

regret the difference between the expected cumulative utility of the optimal policy and that of a given policy. lix

residence time the expected time to spend within an absorbing Markov Model before absorbing/escaping from the model. lx

scheduling the process of assigning workers to the states where they will begin their MD trajectory; submitting a request for segments to be generated. xvi

segment an initial state and a final state, separated by some amount of MD-time. xiv

state a predefined region of configuration space, typically denoted by a single basin of attraction. xi

super-basin a group of states that are highly connected within the group, but weakly connected to states outside of the group. xiv

utility measure of the desirability or value pending the consequence of an action. lviii

worker component of ParSplice that runs MD and is responsible for generating segments of trajectory. xiv

Acronyms

AKMC	Adaptive Kinetic Monte Carlo. xi
AMD	Accelerated Molecular Dynamics. xii
EVSI	Expected Value of Sample Information. lx
hTST	harmonic Transition State Theory. xii
KMC	Kinetic Monte Carlo. xi
MD	Molecular Dynamics. ix
MDP	Markov Decision Process. lxii
ParSplice	Parallel Trajectory Splicing. xiv
PRD	Parallel Replica Dynamics. xiii
QSD	Quasi-Stationary Distribution. xiii
RL	Reinforcement Learning. lxii
TAD	Temperature Accelerated Dynamics. xii
TST	Transition State Theory. xi
VE	Virtual End. xvii
WCT	Wall Clock Time. x

Introduction

Over the past 50 years modern science has become increasingly reliant on computer simulation as the premier tool for understanding nature; particularly for studying phenomena that evolves over length and/or time scales which are not directly accessible to us through our senses. It has become essential to our ability as scientists to motivate theories and make predictions. In fact, a modern theorist who does not frequently make use of computer simulation is about as common as hens teeth.

Since the advent of computers, scientists have been inputting what they believe to be the meaningful physics into simulations and computing the output, which they can compare to experiments to gain an understanding. As time has passed, models have been refined and developed to better represent reality, resulting in simulations which are incredibly accurate. Many scientists will think of their simulations as "numerical experiments" and trust them as though they carry the same validity as experiment. This is, however, a great confusion. Just as with any powerful tool one must fully understand the tool if they are to truly understand the result.

Knowing the extent to which our simulations are valid is of upmost importance when interpreting the results. There is an aphorism which is commonly used within the modeling and simulation community: *All models are wrong, but some are useful.* It is our role as scientists to endow our simulations with as much of the meaningful physics as possible such that we can produce results that are in accordance with reality. Oftentimes, however, sacrifices in accuracy are made in order to reach length and/or time domains which are otherwise not directly accessible. This is often the case in atomistic simulations of thermally activated processes. To better understand why we must first understand the simulation tool itself.

The state of the art in atomistic simulation, Molecular Dynamics (MD), provides a means of peering deep down to the level of molecules to gain an understanding of the atomic behavior. It works by numerically integrating the classical equations of motion for a system of particles. The forces at each timestep are derived from interatomic potentials which are provided by the user. The resulting simulation produces a collection of "snapshots" characterizing the dynamics as the positions and velocities of each atom are evolved forward in time. Crucial to the validity of the simulation is a numerical timestep small enough to resolve the thermal vibrations. Unfortunately, this constrained timestep in turn restricts the times which we are able to reach through direct simulation. This *MD timescale problem* defines a barrier in time (sub-microseconds for general purpose hardware) past which MD simulations cannot access, thus leaving many rich physical processes significantly out of reach. Take for example film or crystal growth. Each deposition event takes on the order of picoseconds, which is no problem for MD. The time between deposition events however is on the order of seconds, meaning the diffusion events that occur on the surface (which affect the film morphology and can be extremely complex) are significantly far out of reach of MD. There is an entire world of example problems like these: grain boundary or dislocation dynamics, radiation damage annealing, contact damage during nanoindentation, slow chemical reactions, etcetera, etcetera. Slow processes like these could be better understood by scientists if only MD simulation could be used to inform our understanding.

One may assume that the ever increasing availability of massively-parallel computers could

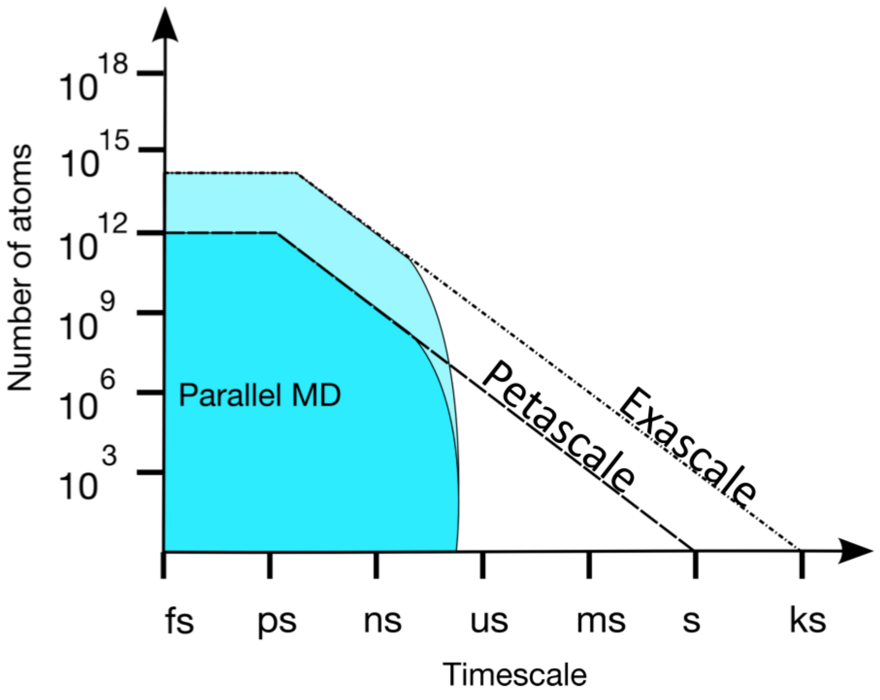


Figure 1: Illustration of achievable MD length and time scales provided one day of compute time. Dashed lines represent theoretical performance achievable if MD were to strong-scale on those platforms. Image courtesy of Danny Perez

be leveraged in order to overcome this timescale barrier, however this is not the case. Parallel MD typically involves some form of spatial decomposition, where the system domain is divided into subdomains that are then run on separate processors, communicating between processors when necessary. Unfortunately, MD only weak-scales with spatial decomposition. As seen in Figure 1, parallel MD is able to utilize more cores in running larger systems. It does not however enable us to reach longer timescales. This is because as the system is divided into smaller and smaller subdomains the simulation finds itself communication-bound; where it is primarily communicating between processors and running very little MD.

This leads to the question: *How we are to extract accurate long-time atomistic dynamics in the presence of this MD timescale problem?*

Seeing that the fixed timestep requires a certain number of numerical integration steps in order to reach a certain simulation time, the naive solution would be to somehow accelerate the process of the numerical integration thereby decreasing the Wall Clock Time (WCT) required to reach that simulation time. As the most computationally expensive part of MD is the force calls required at each timestep this would involve developing faster potentials and/or designing specialized hardware for running MD. While this solution is well founded in principle it's scope is generally limited as the desired simulation timescales are often many orders of magnitude past the accessible timescale barrier and would thus require speedups of the same proportion. Therefore, accelerating the numerical integration of MD alone will not be enough to reach the desired long-time results.

A more fruitful strategy is to exploit the separation of timescales that is often present within the dynamics of a slow, thermally-activated process. Typically, the dynamics of these systems can be characterized in a coarser sense by a series of quick transitions between long-lived, meta-stable regions of configuration space or what we refer to as *states*. The exact intrastate dynamics are usually of little interest as they to not qualitatively contribute to the overall long-time dynamics of the system. It is the interstate dynamics that are of interest as the meaningful long-time behavior of a system can be completely described in the form of a state-to-state trajectory. Note, if the intrastate dynamics for a particular state were of interest they could be obtained through direct MD simulation as the lifetime of the meta-stable state is likely well within the accessible timescales of MD.

Provided that the state-to-state trajectory encapsulates all of the meaningful information on long timescales, a common approach is to utilize Transition State Theory (TST) to compute the rate constants for transitions between states via the energy barrier at the dividing surface [4]. In practice, there are methods for obtaining the saddle point energy [5, 6] which can be employed en route of the rate calculation. If needed, dynamical corrections to the TST rate can even be computed for an increased level of accuracy. Once obtained, a list of states and transition rates can be used to generate a long-time state-to-state trajectory using a stochastic method like Kinetic Monte Carlo (KMC)[7]. KMC samples transitions and times according to the TST rate constants in order to dynamically evolve the system from state to state. In contrast to true-MD which numerically integrates the equations of motion and thus is limited the timescale problem, these stochastic dynamics can be sampled rapidly, generating long-time dynamics without any hesitation. When done correctly, and with a complete list of states and transition rates, one will obtain statistically-correct state-to-state trajectories.

However, acquiring a complete list of *all* of the states and transition rates is oftentimes very difficult, if not impossible. The complexity of many systems can result in an unfathomable number of reaction pathways. In such systems, having a complete list of all of the states and all of the transition rates a priori is simply not possible. Instead, scientists can try to account for all of the pathways which they believe to be relevant, constructing the list on-the-fly through a very clever Adaptive Kinetic Monte Carlo (AKMC) method [8]. This method constructs a list of escapes and TST rates from the current state through a series of saddle point searches until a confidence level criteria for the completeness of the list is met, at which point it samples a transition and time in the KMC-fashion, and proceeds to the next state where the process repeats [9].

A matter of particular interest, as it motivates the work of this thesis, is the confidence level criteria use for determining the rate list completeness. This is a very important concept as the dynamics produced by an AKMC simulation are purely stochastic, and thus their ability to describe reality are directly dependent on the completeness of the TST rate list. As the original AKMC procedure searches for transitions, each successive saddle point search that returns a transition already accounted for on the list is denoted as "relevant but redundant" and is accounted for in a parameter N_r. The confidence parameter, C, indicating that the rate list is complete and a relevant saddle will not be missed can be expressed as $C = (1 - \frac{1}{N_r})$ [9]. Inherent in this confidence parameter is the understanding that the AKMC rate list will never *truly* be complete, and if it were there would be no way of knowing it. This is an important point to make as any omitted pathway from the rate list could potentially have severe consequences on the long-time dynamics.

While more sophisticated uncertainty quantification methods have been developed for estimating the unknown-rates and enhancing the completeness criteria [10], the fundamental issue still remains. A purely stochastic method for generating dynamics will be susceptible to the incompleteness of the TST rate list and will only contain those transitions which are known to exist. This is dangerous as a very rare, perhaps coordinated atomistic event which would naturally occur in a long-time trajectory might not be present in the AKMC result. The dangerous part here is that the scientist would have no idea about the lapse present in their simulated trajectory, they would only know that their trajectory was generated using a confidence level of 99.99%. The intrinsic lack of *knowledge* present in this type of simulation is very non-scientific. The pursuit of truth and understanding becomes murky when the tool used produces a result that may or may not conform to reality. Therefore, while stochastic methods like AKMC are a very powerful tool for developing long-time trajectories of atomistic systems, their Achilles-heel of uncertain-accuracy leaves room for the development of better techniques.

Over the past 30 years, the scientists at Los Alamos National Lab (LANL) have been developing specialized MD-techniques that are designed for reaching long timescales while maintaining the accuracy of MD. This field of work, pioneered by Art Voter in the late 1990's, has come to be known as Accelerated Molecular Dynamics (AMD). Similar to KMC, AMD methods utilize the separation of timescales to produce long-time state-to-state trajectories rather than trajectories that are continuous in phase space. Again, this comes at little to no cost as the long-time behavior of a system is completely described by state-to-state trajectory.

The philosophy behind AMD is to leverage the most powerful component of MD, that is, the ability to find an appropriate escape in statistically correct time without any knowledge of the system a priori. Although direct MD is not capable of generating long-time dynamics past the timescale barrier, the mechanism of MD can be exploited to generate statistically correct escapes. The general idea is to coax the system into finding the appropriate escape faster by running MD in a setting where transitions happen more quickly, and then carefully accounting for the correction in time. Each AMD method is specially designed to accelerate the generation of a state-to-state trajectory **without biasing the result** in any way. The most impressive feature of AMD is the incredibly general statements that can be made regarding the accuracy of it's trajectories. When executed correctly, each AMD method is capable of producing arbitrarily-accurate, statistically correct state-to-state dynamics that push timescales far beyond the MD timescale barrier.

The original work done by Voter in the late 90's, which spawned the entire field of AMD, was devising three distinct "flavors" of acceleration techniques. Each flavor works by utilizing it's own unique mechanism to accelerate the finding of the next escape. These mechanisms are completely independent, meaning they can potentially be layered on one another or combined to further accelerate the dynamics - a concept which has been explored as each method continues to develop. Although the work in this thesis emerges from one of the three AMD techniques devised by Voter, I will briefly summarize each method as they have each played a foundational role in the field of AMD.

The first method designed by Voter is called hyperdynamics [11]. This method works by adding a bias potential to the underlying system in order to accelerate the escape to the next state. The bias potential must be carefully designed to be zero at the TST dividing surfaces in order to not bias the selection of the next state. The time it takes to find the escape with this added bias potential is referred to as the hyper-time, which can then be related back to the unbiased time of the target system through harmonic Transition State Theory (hTST). The *boost* in performance, i.e. the increased rate at which a state-to-state trajectory can be generated relative to direct MD, increases exponentially with the bias potential. This implies that systems with very deep states (for which a large bias potential can be applied) are able to obtain dramatic boosts in performance, pushing the accessible timescales by orders of magnitude. The drawback with hypedynamics is the requirement that the bias potential must be zero on all dividing surfaces. This constraint becomes limiting when very large systems are considered; as systems scale to large sizes the number of transition pathways increase, meaning it becomes more and more likely that a local distortion brings the system close to a dividing surface. Therefore, as systems scale to larger and larger sizes the bias potential must be smaller and smaller, resulting in a boost which decays to unity. Recent work developing hyperdynamics has led to a version where the bias potential is only applied locally, called local-hyperdynamics, which allows the method to scale to larger systems [12]. This advancement has enabled MD-accurate simulations of one million atoms to timescales on the order of a hundred microseconds [13], far beyond anything achievable with direct MD.

Another one of Voter's AMD methods is Temperature Accelerated Dynamics (TAD)[14]. This method considers the naive idea that increasing the temperature (of a low-temperature simulation) will increase the rate of reaction and thus produce transitions quicker. While this is true, the high-temperature transitions will be biased toward escapes with higher barriers, producing faster (but incorrect) transitions. In order to account for this effect TAD runs basin constrained dynamics (where MD is run within a single state and escapes from the state are noted but blocked) at a higher temperature, recording all of the attempted transitions and their corresponding high-temperature time. Then, using TST, TAD is able to correlate each high-temperature time back down to the low-temperature time, i.e the time it would have taken for the transition to occur at low-temperature. TAD can then accept the first transition that would have occurred at low-temperature, extending the state-to-state trajectory by the low-temperature time to the accepted transition state. From the new state the procedure repeats, running basin constrained dynamics with high-temperature MD. Using TAD, scientists are able to obtain incredible boosts, particularly when gap between T_{high} to T_{low} is large. The original paper showcased results of film ripening on a CU(111) surface [14]. The high energy barriers allowed the 150K simulation to use a T_{high} of 1000K, generating a state-to-state trajectory reaching timescales of hours. For comparison, direct MD would achieve roughly 0.4 microseconds for the same WCT, thus TAD provided a staggering boost of roughly 10^9. Note, however, that the boost obtained using TAD is highly dependent on temperature. TAD accelerating the dynamics of an identical simulation run at 300K (rather than 150K) only achieved a boost of roughly 250. Recent developments of TAD include a speculatively parallel version which is able to utilize multiple cores and start running basin constrained dynamics on transitions as they are discovered (i.e, before they are accepted) in order to further accelerate the state-to-state dynamics [15]. Research into TAD also led to the use an AMD simulator designed to explore potential algorithmic developments without the unnecessary expense required to generate true atomistic dynamics [16]; a research approach which was used in the development of this thesis work.

The last of Voter's original three AMD methods is Parallel Replica Dynamics (PRD) [17]. This method leverages the use of parallel computing to run unbiased dynamics on several independent replicas in order to find the next escape from a state sooner. It begins by broadcasting a single state to each replica where it will begin running MD. Since each replica is running MD at the target dynamics (i.e, unbiased MD) the escape which is found by a replica will be an appropriate transition. Moreover, the appropriate MD-time required for finding said transition is simply the "parallel-time" or the sum of MD-time accrued by each replica. As a result, the WCT time invested in finding the escape from a state is reduced by a factor of N, the number of parallel replicas. Once a transition is found, each replica process is terminated and the newly transitioned state is broadcast to each replica, where the process repeats again.

In practice, each replica must first undergo some *decorrelation time* (τ_c) in a state before that replica begins to accrue MD-time. This decorrelation time is required to initialize a replica in a particular state and ensure that it's trajectory is statistically independent from the other replicas. Formally, it is the time required for the process to relax toward the state's so-called Quasi-Stationary Distribution (QSD). This is the self-similar distribution that results in the limit of dynamics confined to a single state with absorbing boundaries, and pertains to exponentially distributed first escape statistics. Once an appropriate decorrelation time has elapsed the replica can be viewed as a random sample from the state's QSD, meaning first-escapes have become completely Markovian, i.e the trajectory has become memory-less. It is only once this decorrelation cost is paid that a replica will start producing meaningful work towards the state-to-state trajectory. Therefore, while PRD is capable of accelerating the discovery of transitions by a factor of N, it comes at a cost of $N\tau_c$. This implies PRD is best suited for processes where the average time needed to escape a state, $\langle\tau_{esc}\rangle$, is such that $\langle\tau_{esc}\rangle/N \gg N\tau_c$; outside of this domain the overhead cost begins to compete with the performance. Provided the $\langle\tau_{esc}\rangle$ of a given process, this sets an upper-bound on the number of replicas which can be efficiently utilized, thus placing a scaling limit on the achievable boost.

The Mathematical underpinnings of PRD, as well as hyperdynamics and TAD, have been rigorously proven to produce arbitrarily-accurate, statistically-correct state-to-state trajectories [18]. The accuracy of PRD is dependent on the chosen decorrelation time τ_c spent relaxing toward the QSD. Formally, the dynamics produced are only exact in the limit $\tau_c \to \infty$, but the error produced decreases exponentially with increasing τ_c. This remarkable fact distinguishes PRD from the other

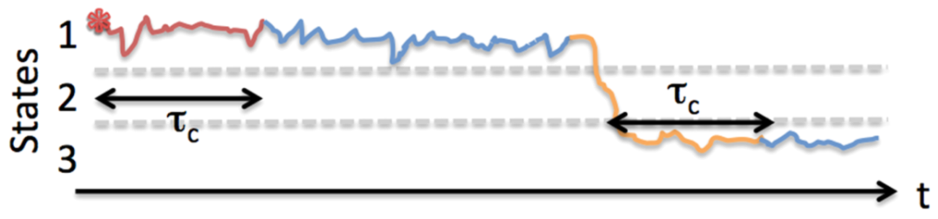


Figure 2: Illustration of the concept of spliceable segment taken from original manuscript [1]. The red star marks the beginning of an MD run in state 1. The dashed lines represent state boundaries. Any section of this trajectory that begins and ends in blue regions is a spliceable segment. Such segments however cannot contain any portion of the red section or begin or end in yellow sections.

two methods in that a linear cost in τ_c will produce an exponential payoff in accuracy. For most practical applications, a τ_c of a few picoseconds is sufficient for establishing a replica in a given state. Another factor that sets PRD apart is the incredibly general nature of the method. Since each replica is running unbiased MD there is no reliance on TST or hTST to convert the accelerated dynamics back to target dynamics of interest. Moreover, the concept of a QSD allows for more creative state definitions that can be employed; i.e not just a single potential basin.

Enabled by the notion of a QSD, a development was made to PRD that really transformed it into something new. The big idea was to assign replicas to generate trajectory in different states, and store the work done until it was needed by the state-to-state trajectory. Each replica or *worker* as they are referred to in this new method (since they are no longer all copies running MD in the same state) is assigned to a particular state and (after running MD for τ_c without an escape) generates a trajectory for some short predefined period of time, after which it attempts to end the trajectory at the first possible instance - conditional on having spent τ_c in the present state (see Fig 2). The result is a *segment* of trajectory; an initial state and a final state, separated by some MD-time. (Note: the initial and final state of a segment need not be different - They will only differ if the segment contains a transition). It is a requirement in the prescription of generating a segment that the trajectory spend at least τ_c in the initial state before starting and the final state before ending. Therefore, the endpoints of each segment can be considered random samples from their respective QSDs. Exploiting the fact that any two random samples from the same QSD are statistically equivalent (in regards to first escapes) then we can append or "*splice*" one segment onto the end of another if the one segment starts where the other segment ended. Utilizing this concept, segments of trajectory can be generated in parallel and then spliced together to create a single long-time trajectory (see Fig 3). Due to this distinctive feature the new method, born out of PRD, came to be called Parallel Trajectory Splicing (ParSplice) [1].

The developments which led to ParSplice enabled tremendous improvements in performance, primarily of which was the ability to handle *super-basins*: groups of states that are highly connected within the group, but weakly connected to states outside of the group. In the presence of a super-basin a trajectory will likely visit and revisit each state in the super-basin many times before eventually escaping the super-basin. A classic example is a trimer diffusing on a surface. The lowest energy barrier is the "spin" transition where one atom from the trimer hops to a neighboring lattice site (see Fig 4). As this is the lowest barrier event it occurs many times before the trimer diffuses elsewhere, thus creating a four state super-basin.

In practice, it is quite common for systems to contain super-basins of states (or even super-basins of super-basins, e.g glasses). This posed problematic for PRD, especially when the intra-super-basin transition rates were fast, i.e states within a super-basin were separated by low barriers. PRD is only able to accelerate the discovery of an escape from the current state. As previously mentioned, when a state is shallow (meaning escapes from that state are relatively quick) the boost provided by PRD will be minimal. Thus, PRD is unable to accelerate the dynamics of a process

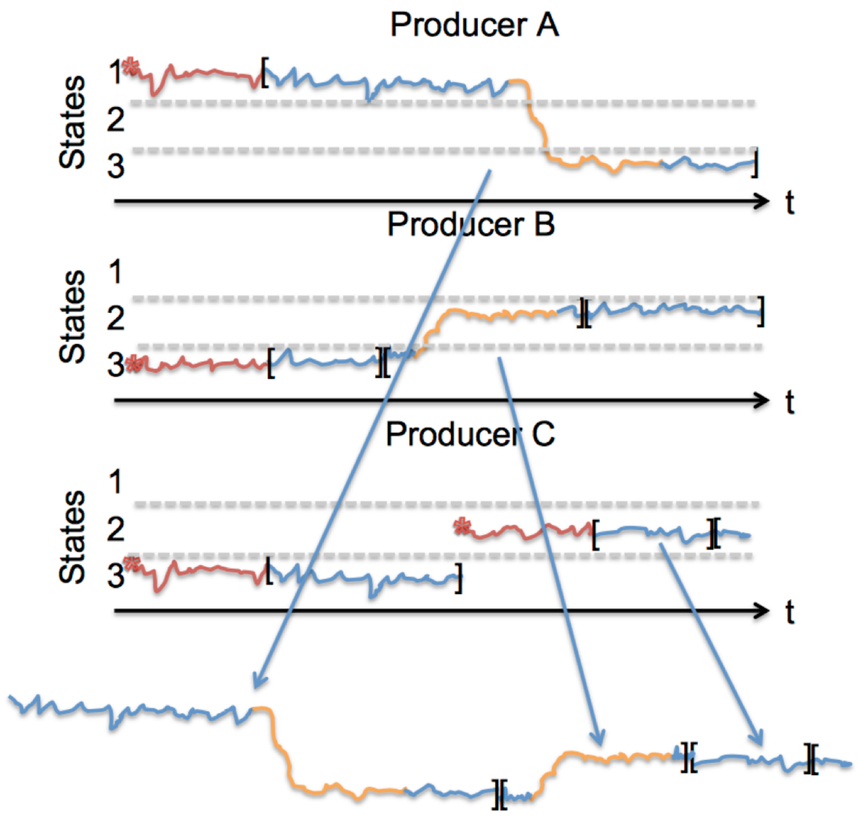


Figure 3: Illustration of the concept of segment splicing courtesy of Danny Perez. Each producer works in parallel to generate segments of trajectory which are then consumed into the single long-time trajectory.

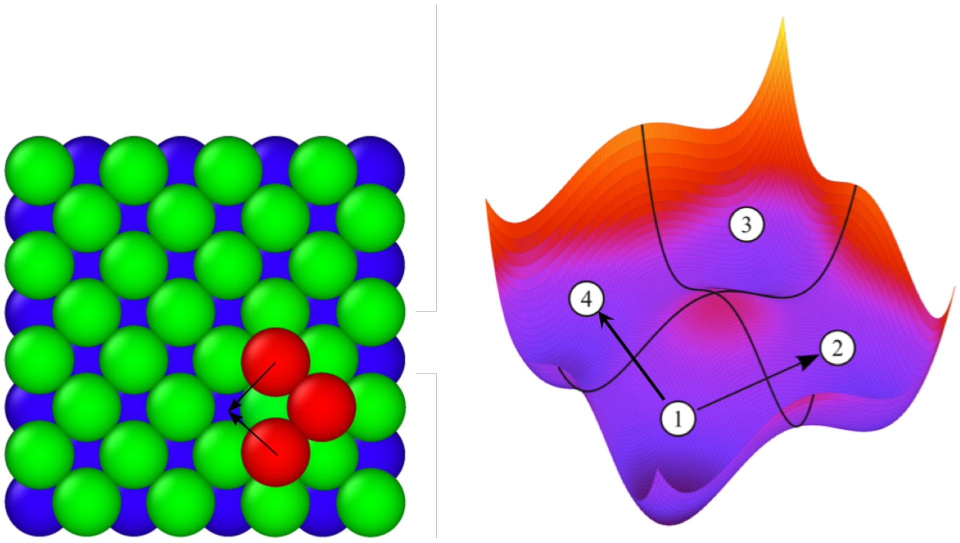


Figure 4: On left, illustration of the Ag/Ag(100) trimer system taken from original manuscript [1]. Atoms are colored by their position in the z direction. Red: trimer atoms; green:surface layer; blue: subsurface layer. Arrows show the possible displacements of trimer atoms during the two possible "spinning" transitions. On right, corresponding potential energy surface illustrating the four state super-basin with arrows indicating transitions

that is trapped within a super-basin of shallow states.

ParSplice, on the other hand, can assign workers to different states and save the work done, i.e segments generated, until they are needed by the state-to-state trajectory. When a state is relatively shallow and ParSplice assigns many workers to that state, generating more segments than were needed to escape from the state, all excess segments (that are not used in finding the first escape) are stored in a database where they can later be extracted and consumed into the trajectory if trajectory ever revisits that state. That is, ParSplice is able to capitalize on revisits to states and can oftentimes immediately escape from the state (again) due to the unused segments that were generated previously. As a result of these features, ParSplice is uniquely well-suited for handling super-basins of states.

Another advantage ParSplice has over PRD is the improved ability to scale. Recall, PRD was scale-limited by the constraint of needing very deep states in order to efficiently utilize very many cores, thereby satisfying $\langle \tau_{esc} \rangle /N \gg N\tau_c$. This served problematic as the greatest potential of parallel methods lies in their ability to scale. Both ParSplice and PRD possess ideal scaling in the presence of very deep states. In fact, ParSplice and PRD become identical in the limit of infinitely deep states. ParSplice, however, maintains ideal scaling in the presence of very deep super-basins (provided long enough simulation time) even if the constituent states are themselves very shallow. The ability of ParSplice to tackle super-basins through the amortization of previously generated (unconsumed) segments is quite profound, particularly with the realization that the depth of individual states composing the super-basin are completely irrelevant so long as the super-basin itself is sufficiently deep.

The ability of ParSplice to scale is unique in that it is not entirely dependent on the depths of states or super-basins. In principle, ParSplice can assign workers to states arbitrarily (a process often referred to as *scheduling* segments), producing any arbitrary distribution of segments. This implies that, in theory, workers could be assigned not only to the current state, but to whatever state the trajectory is going to escape to next, as well as the state after that, and so on. Scheduling segments ahead of the trajectory allows for the utilization of very many resources. In fact, ParSplice has the potential ability to strong-scale to arbitrarily large resources provided an accurate means of forecasting where the trajectory will be into the future.

Unfortunately, this dream of perfect strong-scaling requires perfect clairvoyance, which is often not attainable. We can, however, attempt to mimic the predictive behavior and employ some speculative scheduling; attempting to generate the number of segments that are expected to be needed to escape from a state, and continuing to do so where ever the trajectory is likely to go next. Doing this effectively requires an oracle capable of making predictions about the state-to-state dynamics. In this effort, ParSplice develops a statistical Markov model on-the-fly from generated segments and uses it to speculate on the trajectory's future. It does so by sampling *"virtual"* endpoints for "pending" segments (those segments which have been scheduled but have not yet been generated and returned to the database). This provides a potential future image of what the database might look like once all pending segments are completed. Using this future image, ParSplice can *"virtually"* splice segments to determine where the long-time trajectory would be if this potential future was realized. It then schedules the next segment to be generated at the end of this *"virtual"* trajectory, in what is referred to as Virtual End (VE) scheduling. Note: the word *"virtual"* is stressed to make the distinction that nothing **actual** is happening; no segments are being manipulated nor is anything spliced. The VE procedure can be interpreted as a thought experiment used to speculatively schedule ahead of the long-time trajectory. As a ParSplice simulation runs segments are constantly being generated and observed, thus improving the statistical Markov model which in turn improves the quality of the VE scheduling decisions.

In summary, ParSplice is an incredibly powerful AMD method which enables the generation of long-time, MD-accurate state-to-state trajectories. It accelerates the construction of trajectories via a time-wise parallel technique that does not bias the dynamics in any way. Through the use of ParSplice scientists can reach simulation times up to N times longer than direct-MD. This prospect of strong-scaling MD is extremely enticing given the continuously increasing scale of available computational resources: on current petascale platforms N can be in the hundreds of thousands, which opens the door to MD-accurate millisecond-long atomistic simulations; extending such a capability into the exascale era would be transformative. It is for this reason that ParSplice was chosen to be part of the Exascale Computing Project (ECP), a large DOE-funded collaborative project aimed at developing exascale-ready applications and solutions that address currently intractable problems of strategic importance and national interest. More specifically, ParSplice is the **T** in the **EXAALT** portion of the project, which aims fully utilizing the power of the **EXA**scale for MD simulations of increased **A**ccuracy, **L**ength, and **T**ime.

The work detailed in this thesis is to further develop a method for extracting accurate long-time atomistic dynamics in a way that remains faithful to the true underlying physics. It was done inline with the ECP mission, and to enable new science that will only be made possible through the next generation of computing. It is my hope that the work done will one day contribute toward the greater goal of bringing us closer to a deeper understanding of the physical world.

The work of this thesis is organized as follows: Motivated by the fact that the ability of ParSplice to scale is only limited by the ability to speculate, Chapter 1 details the work done to improve the statistical model that is used in the VE scheduling procedure. The work detailed in Chapter 2 describes a new scheduling procedure and optimization paradigm designed for employing and fully utilizing massively-parallel platforms. This is followed by some unpublished work in Chapter 3 which investigates a decision theoretic approach to optimal scheduling and calls into question the metric used for evaluating a scheduling decision. Lastly, Chapter 4 briefly outlines some potential applications and future work.

Improving the statistical Model

0.1 Introduction

Molecular Dynamics (MD) is a widely used computational workhorse that provides fundamental insights into the nanoscale behavior of materials. Crucial to the stability of the simulation is a numerical timestep small enough to resolve even the fastest thermal vibrations, i.e., a few femtoseconds. Unfortunately, this strongly limits MD-accessible timescales (to sub-microseconds on general purpose hardware) and hence the insights that can be obtained from MD. This limitation is especially problematic for systems that evolve through rare events, a very common situation in a range of materials, as it often precludes the observation of the mechanism that controls the behavior of the system over long times.

One might expect that each successive generation of massively-parallel computers would extend the reach of MD simulations to larger systems and longer times, but this is only partially true. Conventional parallel MD involves some form of spatial decomposition [19], allowing the utilization of large numbers of processors to simulate very large systems. Such strategy does not however significantly affect the accessible timescales because efficiency requires a sufficiently large number of atoms per processor in order to amortize communication and synchronization overhead. This requires that the system size scales proportionally with the number of computing units on the machine that is used, and hence does little to increase the simulation timescales. This limitation is especially problematic given that many material properties depend on thermally activated processes, such as defect nucleation, diffusion, and annihilation, that occur on long timescales. Therefore, if we are to fully harness the computational capabilities provided by these increasingly large computers and extend the accessible timescales of MD simulation, we must consider alternative parallelization approaches [20], or abandon direct MD simulations in favor of higher-level models, such as Kinetic Monte Carlo (KMC)-based techniques [8, 21].

One such alternative approach is Parallel Trajectory Splicing (ParSplice), which leverages parallel computing to parallelize the simulation in the *time domain*. It does so by concurrently and independently generating many short *segments* of MD trajectory in such a way that segments can later be *spliced* together to form a very-long state-to-state trajectory [1] that is statistically valid. States are usually defined as basins of attraction on the energy landscape, i.e., all points in configuration space that converge to the same minimum upon gradient descent belong to the same state. This is a good definition when all saddle points surrounding such basin are sufficiently high compared to the thermal energy scale k_BT. While this choice is very computationally convenient, more general definitions are also possible [22]. ParSplice can be seen as a generalization of the Parallel Replica Dynamics method (ParRep) introduced by Voter [17, 22] and it is related to the Distributed Replica Dynamics method of Henkelman *et al.*[23]. The code is open source and is freely available for download [24].

It can be shown that the distribution of any system that spends a sufficiently long time trapped within a state without escaping relaxes towards the state's unique quasi-stationary distribution (QSD). Once the QSD is reached, the first escape statistics out of the state become Markovian, i.e., memoryless [18]. In this context, the correlation time τ_c is defined as the characteristic time

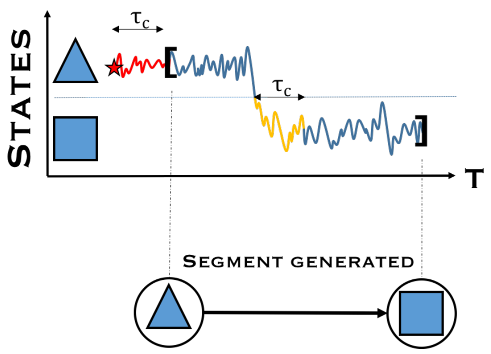


Figure 5: Illustration of the concept of a spliceable segment. The red star marks the beginning of an MD run in the triangle state. The horizontal line represents a state boundary between the triangle and square states. Any section of this trajectory that begins and ends in blue regions (meaning the trajectory has spent at least τ_c in the state it is currently in) is a spliceable segment. In practice, segments are ended at the first opportunity after reaching some minimum length[1].

needed to approach the QSD from any initial condition in the state. The choice of correlation time is in general not a simple question [22]. However, in the case where states are defined as individual basins of attraction of the energy landscape, it can be shown that de-correlation occurs on vibrational timescales, so a typical value of τ_c for hard solids is a few picoseconds.

ParSplice takes advantage of this notion to produce *segments* of MD trajectory whose endpoints are such that the trajectory spent at least (τ_c) in both the starting and end states (which might or might not be the same), before the beginning and end of the segment, respectively. These end points are therefore (approximately) samples from the respective QSD of the starting and end states (c.f., Fig. 5). As two endpoints sampled from the same QSD are statistically equivalent with respect to predicting the next escape from the state (due to the Markovian nature of the first escape statistics), *splicing* a segment that begins in a given state at the end of an independently generated segment that ends in the same state therefore preserves the proper state-to-state transition statistics. This approach has been proven exact in the limit $\tau_c \to \infty$; at finite τ_c, the errors vanish exponentially with increasing τ_c [18]. As a result, ParSplice can yield essentially MD-accurate *state-to-state* dynamics. Note that typically, segments are stopped at the first opportunity (according to the conditions described above) after they reach a minimum length, which is typically itself on the order of τ_c. In principle, shorter segments lead to high efficiency by allowing for segments generated by multiple workers to contribute to a single escape. Consider a situation where the states are very deep, so that each escape requires nanoseconds. A segment that runs until the first escape to a new state could require hours of wall-clock time to generate, while this time could be compressed down to seconds by splicing short segments independently generated by thousands of workers. Short segments also minimize waste, avoiding the generation of extremely large amounts of MD time in the same states, which would require a very large number of revisits to amortize. However, in practice, very short segments segments entail communication overhead; length of order τ_c usually strike a reasonable tradeoff.

The practical interest of ParSplice is that its simulation rate can exceed that of standard MD by up to a factor of N, the number of MD instances concurrently generating segments. This prospect is extremely enticing, given the continuously increasing scale of available computational resources: on current peta-scale platforms, N can be in the hundreds of thousands, which opens the door to MD-accurate millisecond-long atomistic simulations; extending such a capability into the exascale era [25] could be transformative. Maintaining efficiency at large computational scales is however contingent on the state-to-state transitions being sufficiently rare. In its most stringent form, this condition requires that the escape times of out each state be long compared to $N\tau_c$ [22]. In practice, this conditions can be relaxed for energy landscapes that contain so-called super-states, i.e., groups of states that are highly connected within the group, but weakly connected to states outside of the group. In this case, high efficiency is maintained so long as the escape time out of each *super-state* is long compared to $N\tau_c$ [26]. While the common occurrence of super-states in many systems somewhat alleviates this concern (super-states are so common that specialized Monte Carlo techniques specifically capitalize on their presence [27, 28, 29, 30]), theses requirements nonetheless restrict the range of systems that can be efficiently simulated as N increases: the larger the computer on which ParSplice is deployed, the rarer the events have to be in order to maintain a given parallel efficiency.

The scaling of ParSplice can however be further improved by accurately forecasting the future evolution of the trajectory. Consider a thought experiment where a perfect oracle predicts the exact state-to-state trajectory ahead of time. ParSplice could then concurrently generate exactly the right number of segments in each state without any waste, leading to perfect scalability even if state-to-state transitions are not rare. While such a perfect oracle of course does not exist in practice, this observation motivates the need to create practical approximations to such an oracle. In its original implementation, ParSplice develops an approximate representation of the state-to-state dynamics in terms of a discrete-time Markov Chain using a maximum likelihood estimator (MLE) of the transition probabilities estimated from previously completed segments. The future evolution of the trajectory is then forecasted using the MLE model, and the segments which are the most likely to be needed next by the state-to-state trajectory are scheduled for execution. In the present study, we show that the MLE estimator is sub-optimal in practice because i) it does not fully exploit the physical structure of the actual underlying Markov model, and ii) it does not take into account the incompleteness of the model. In the following, we show that significant performance improvements can be expected by addressing these two issues, thereby significantly improving the scalability of ParSplice.

The manuscript is organized as follows: Sec. 0.2 first reviews the basics of ParSplice and its use of Markov models for predictions; Sec. 0.3 then introduces improved statistical prediction models; the performance of these models is assessed on both simulated and actual ParSplice simulations in Sec. 0.4, before concluding.

0.2 ParSplice Review

ParSplice is best understood in terms of a Scheduler-Workers-Splicer paradigm (see Fig. 6). The role of the Scheduler is to assign segments to be completed by Workers. Each Worker receives a state from the Scheduler, generates a segment of MD trajectory beginning in the assigned state according to the prescription given above, returns that segment to a database, and repeats. The Splicer extracts segments from the database and splices them onto the end of the generated state-to-state trajectory. The goal of the Scheduler is to ensure that the segments needed to extend the state-to-state trajectory are available to the Splicer as quickly as possible, while simultaneously avoiding generating segments that won't be spliced into the trajectory.

With this conceptual picture of ParSplice in mind, we can clearly distinguish the respective sources of accuracy and efficiency. The ParSplice formalism guarantees that splicing any independently-generated set of segments produces a statistically correct state-to-state trajectory so long as τ_c is chosen properly and that the order in which segments are spliced is proper [31]. Many different segment orderings are possible in practice, as discussed in Ref [31]. In short, the order can be arbitrary so long as it is independent of the content of the segment (e.g., of its length, of whether or not it contains a transition, etc.). In the current implementation of ParSplice, segments can only be spliced in the order in which their generation was initiated (in wall-clock time). The accuracy of the trajectory is therefore completely unaffected by the Scheduler's decisions. However,

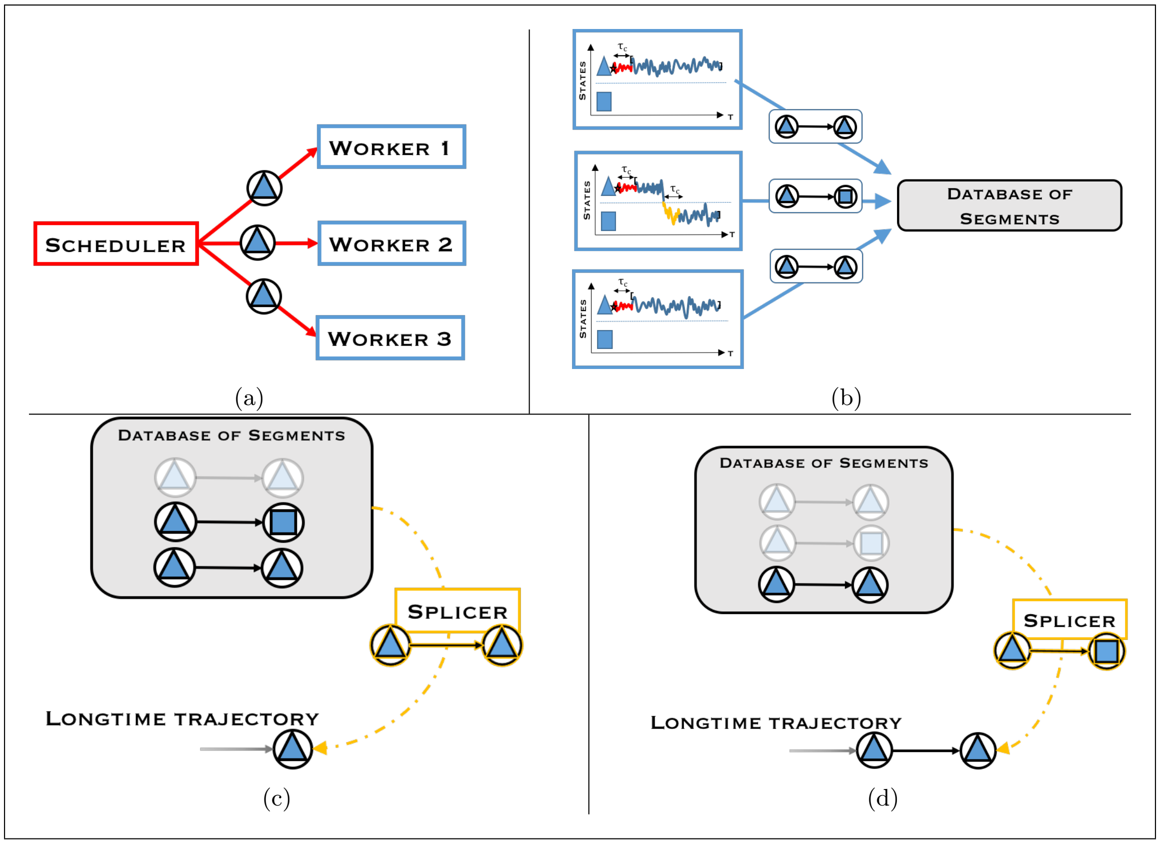


Figure 6: Conceptual illustration of ParSplice: (a) The Scheduler (red) assigns each Worker (blue) the task of generating a segment that begins in a particular state. (b) The Workers independently run MD in their assigned state, creating a segment which they return to the database. (c) and (d) The Splicer (yellow) extracts segments from the database and splices them onto the end of the long-time trajectory. In practice, all of these operations occur concurrently and asynchronously.

the parallel efficiency of ParSplice, which is measured by the proportion of the segments generated that are actually spliced into the long-time trajectory, is directly determined by the Scheduler's ability to properly allocate resources (i.e, Workers) to those states where segments will be needed. Only segments that are eventually extracted from the database and spliced into the trajectory provide direct utility; segments that are left in the database at the end of the simulation provide no direct utility (beyond contributing to the statistical model). To summarize, the efficiency of ParSplice therefore relies on making wise segment-generation decisions, but its accuracy does not. This opens the door to heuristic scheduling approaches without sacrificing accuracy, a fact that is exploited below.

A naive scheduling approach would be to simply instruct all available Workers to generate their segments beginning in the current end-state of the state-to-state trajectory, in what is referred to as Actual End (AE) scheduling [1]. This approach guarantees finding an escape from the current-state as quickly as possible, and excess segments not utilized in this escape would remain in the database for future use. The problem is that any excess segments remaining in the database after the first escape will only be utilized in the event that the trajectory revisits this same state again. If N is much greater than the number of segments needed for a single escape, the trajectory will need to revisit the state a proportional number of times in order to amortize the excess number of segments that are generated at the first visit. This problem becomes increasingly likely and severe as N increases.

One may note that, in an infinitely long trajectory, each state will be visited infinitely many times, and since the database will only contain a finite number of unused segments in each state, they will make up a vanishing proportion of the total segments produced and thus the parallel efficiency of AE scheduling will approach unity. Indeed, AE will generate new segments in a state only if the database does not already contain a segment that transitions from this state to another. Therefore, at worst, the database will contain only N segments in each state at any point in time. This may lead to the belief that being patient enough will allow any reasonable scheduling technique to suffice as the simulation will eventually reach ideal efficiency. While this is true in theory, it would involve running for a *very* long time. In practice, a simulation typically only visits a very small portion of the whole state-space of the system. Therefore, our focus in the present study is on achieving high parallel efficiency in this finite-time regime in which the need to intelligently schedule segments becomes essential.

A better scheduling approach is to only schedule in the current-state what is needed to escape, and with the remaining resources, schedule in those states which are likely to be visited next. Doing this requires an oracle capable of accurately forecasting the state-to-state dynamics produced by splicing segments. ParSplice builds this oracle on-the-fly by constructing a statistical model from the segments generated in the form of a discrete-time Markov Chain (which is provably the right class of statistical model in this case, as the end points of independently generated segments are by construction uncorrelated). The focus of the current work is to improve upon this statistical model as its accuracy directly pertains to the parallel efficiency achieved, as described below.

The original implementation of ParSplice relies on a maximum likelihood estimate (MLE) of the transition probabilities, where the probability p_{ij} that a segment starting in state$_i$ will end in state$_j$ is approximated by $p_{ij} = \frac{c_{ij}}{\sum_k c_{ik}}$, where c_{ij} is the number of observed segments that started in state$_i$ and ended in state$_j$. The process of scheduling segments makes use of a statistical model to sample virtual endpoints for pending segments; those which are presently being generated and for which the end point is not currently known. This is done in a Monte Carlo fashion where the virtual endpoint for a pending segment is selected according to the transition probabilities present in the model, i.e., a pending segment being generated in state$_i$ will have a virtual endpoint state$_j$ sampled from the model according to the probability p_{ij}. This is done for each pending segment in order to construct a potential future of what the database might look like once those segments have completed. Using this projected future of the database, a virtual trajectory can be spliced together in order to determine where the next segment will be needed. This virtual trajectory will only be so

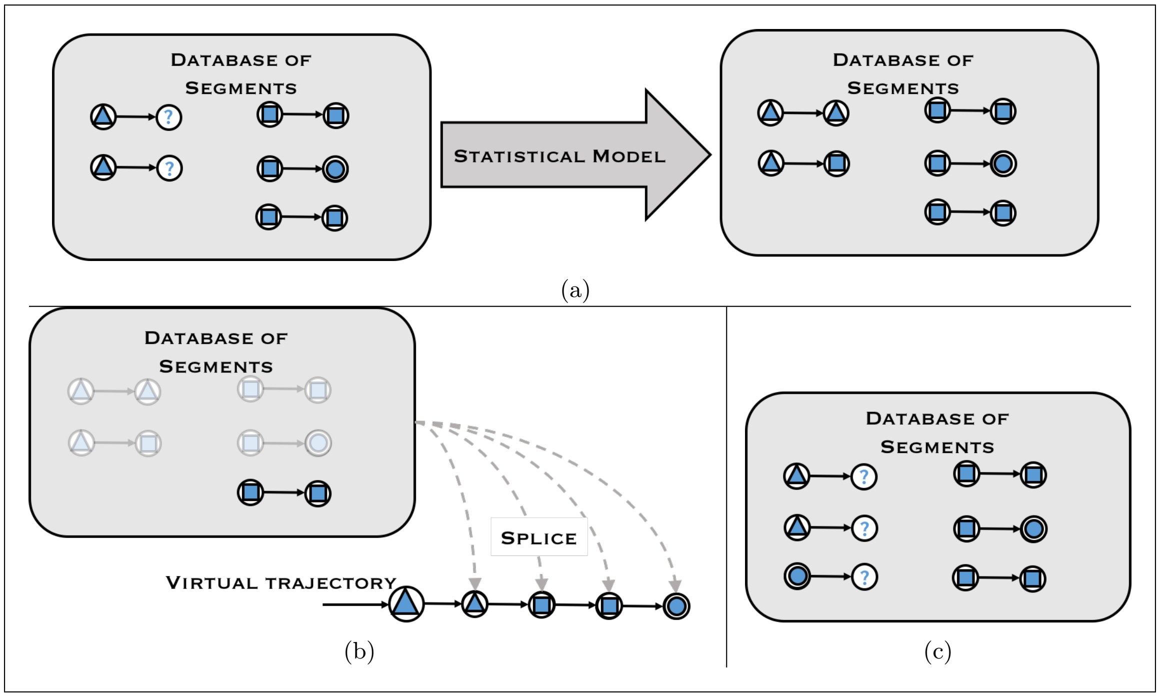


Figure 7: Illustration of Virtual End (VE) scheduling: (a) Endpoints of pending segments are *virtually* sampled using a statistical model. (b) Segments from this database are then *virtually* spliced onto the end of the long-time trajectory to predict where this *virtual* trajectory will run out of segments. (c) A segment is scheduled in the *virtual end* state and the process is repeated; sampling endpoints for all pending segments, but now including the one just scheduled. We stress the word *virtual* to differentiate from the *actual* segment generation and splicing done by the Workers and Splicer, respectively. The *virtual* process can be seen as a thought experiment conducted by the Scheduler as a means of forecasting ahead of the long-time trajectory.

long as the projected future database permits, and will end as soon as there are no more segments in the projected database which can be spliced onto the virtual trajectory. The state upon which this virtual trajectory ends is denoted as the *virtual end* state and is where the next segment will be scheduled. Scheduling another segment involves repeating the same procedure again: sampling (new) virtual endpoints for all pending segments, *including the one just scheduled.* This produces a (new) projected future database from which a virtual trajectory can be spliced together in order to determine the *virtual end* state where the next segment will be scheduled. This procedure for scheduling segments is thereby referred to as Virtual End (VE) scheduling (see Fig. 7).

In practice, ParSplice obtains transition statistics from the generated segments and uses this information to construct a model which is continually updated as soon as completed segments are returned to the database. Doing so on-the-fly means initially scheduling with a poor model that gradually improves as the simulation proceeds and further statistics are gathered. The cost of learning the model has a significant impact on parallel efficiency, as will be shown below. The compounding utility that is gained by having a good model earlier in a simulation cannot be understated, and is the motivation for the present work. It is important to note that the VE procedure for scheduling segments remains unchanged and is implemented as described throughout the entirety of this manuscript. The developments of our work are instead focused on enhancing the statistical model from which virtual endpoints of pending segments are sampled.

0.3 Methods

As described above, having an accurate statistical model from which trajectories can be sampled is crucial to reaching peak parallel efficiency. A key observation that motivated the present work is that the MLE model described above does not fully leverage the available information. Consider the following scenario: after splicing a segment from state$_i$ to state$_j$, the trajectory is now in state$_j$. If no segment has yet been generated in state$_j$, the MLE model has no information about transitions out of state$_j$ and cannot sample virtual outgoing transitions. The result is that all workers are scheduled in state$_j$. As discussed previously, this is usually not optimal, and becomes especially problematic at large N, where it reduces to AE. This type of suboptimal scheduling will occur every time the trajectory first discovers a new state.

0.3.1 The known-unknown

The statistical fallacy here is that *nothing* is known about state$_j$. Inherent in a segment that shows a transition from state$_i$ to state$_j$ is the information that a transition from state$_j$ to state$_i$, traversing along the same pathway, is also possible. Including this information in our model would mean that trajectories could escape from state$_j$ back to state$_i$, and from there, to the rest of the known state space. However, this involves estimating the probability to jump back, p_{ji}, without yet having generated any segments in state$_j$. While Bayesian priors could be introduced, we find that setting these in practice is difficult as poor priors can lead to lower efficiency than with the standard VE scheme.

Instead, p_{ij} and p_{ji} can be related to one another if one assumes that the underlying dynamics obeys detailed balance (DB), i.e., that

$$\pi_i p_{ij} = \pi_j p_{ji} \tag{1}$$

for every i and j, where the π's are the stationary occupation probabilities of each state. The DB assumption is commonly invoked in continuous-time/discrete-state representations of materials kinetics using Kinetic Monte Carlo [7]. When Transition State Theory (TST) [32] is used to compute state-to-state transition rates, the resulting continuous-time Markov chain naturally obeys detailed balance. Assuming that such a representation holds, it can be shown that the discrete-time/discrete-state Markov model of segment statistics used internally in ParSplice also inherits from DB [33]. Whether DB is strictly obeyed in general is not crucial, as this assumption does not affect the accuracy of the generated trajectory, but only the parallel efficiency with which it is generated. As demonstrated below, even an approximate model that is forced to obey detailed balance clearly improves predictions of the future evolution, and hence the parallel efficiency, compared to the original MLE model.

We therefore wish to construct a statistical model from the segments generated in such a way that DB is obeyed globally. To do this, we solve the optimization problem of finding the model P which maximizes the log-likelihood given the observed segments counts c_{ij}, but constrain our optimization to only include those models which obey DB. More formally, $\max_P \sum_{i,j} c_{ij} \log p_{ij}$ now subject to constraints

$$\sum_j p_{ij} = 1 \tag{2}$$

$$\pi_i p_{ij} = \pi_j p_{ji} \tag{3}$$

$$p_{ij} \geq 0 \tag{4}$$

We assume that the stationary probability in each state is proportional to the Boltzmann factor $\pi_i \propto e^{-\frac{E_i}{k_B T}}$ where E_i is the minimum energy in state$_i$, T is the thermodynamic temperature, and k_B is the Boltzmann constant. This assumption is reasonable when states are defined as deep basins of attraction on the energy landscape and when the vibrational contribution to the partition function does not significantly change from state to state. As metastable states will have a p_{ii} that is typically close to 1, meaning most segments will not contain any transition and hence will have exactly the same length, the equilibrium probability of the discrete process will be proportional to that of the continuous process. This roughly corresponds to a "constant prefactor" assumption that is widely used in the Kinetic Monte Carlo community. Recall, even if the approximations are not fully upheld, the introduction of a DB-based model in practice significantly improves the performance as compared to MLE, which is the key factor since accuracy of the spliced trajectory is unaffected by the assumptions used in building the statistical model. Note that it is also possible to enforce DB without specifying the π [34], but providing them speeds up convergence.

DB-obeying models are obtained following the prescription detailed in section 3 of Trendelkamp-Schroer *et al.* [34], reproduced here for completeness. The first constraint is enforced by introducing Lagrange multipliers λ_i and associated terms $\lambda_i(\sum_j p_{ij} - 1)$ to the objective function. The second constraint is explicitly accounted for into the likelihood expression by the change of variables

$$p^{'}_{ij} = \begin{cases} p_{ij} & i < j \\ \frac{\pi_j}{\pi_i} p_{ji} & \text{otherwise} \end{cases} \tag{5}$$

This produces a Lagrangian which, upon extremization and reversal of the change of variables, leads to

$$p_{ij} = \pi_j \frac{c_{ij} + c_{ji}}{\lambda_i \pi_j + \lambda_j \pi_i} \tag{6}$$

Summing over j and enforcing that the transition probabilities sum up to unity yields

$$\sum_j \pi_j \frac{c_{ij} + c_{ji}}{\lambda_i \pi_j + \lambda_j \pi_i} = 1 \tag{7}$$

Lagrange multipliers λ_i are then obtained by a fixed-point iteration:

$$\lambda_i^{(n+1)} = \sum_{j, c_{ij}+c_{ji}>0} \frac{(c_{ij} + c_{ji})\lambda_i^{(n)} \pi_j}{\lambda_i^{(n)} \pi_j + \lambda_j^{(n)} \pi_i} \tag{8}$$

Using the converged values of λ_i, the reversible model is finally obtained as:

$$p_{ij}^{DB} = \pi_j \frac{c_{ij} + c_{ji}}{\lambda_i \pi_j + \lambda_j \pi_i} \qquad i \neq j, \lambda_i + \lambda_j \neq 0 \tag{9}$$

$$p_{ii}^{DB} = 1 - \sum_k p_{ik} \qquad i = j \tag{10}$$

By enforcing DB, this model accounts for the *known-unknown* transitions which are known to exist, but have not yet been observed. A model that embodies this information is able to sample virtual escapes from newly discovered states, avoiding the trap of having to over-build segments. In addition, enforcing detailed balance greatly reduces the statistical error present within the model by reducing the number of independent variables by about half. A more accurate model can therefore be obtained using the same amount of segment information, allowing better scheduling decisions to be made sooner.

0.3.2 The unknown-unknown

Further improvement on this model involves accounting for the *unknown-unknown* transitions which have never been observed in either the forward or backward direction due to the limited number of segments generated. Given the limited amount of segments generated in each state, such transitions are virtually guaranteed to exist [35, 36, 10]. However, both MLE and DB models assign zero probability to such a transition. Rigorously accounting for the impact transitions of this nature is inherently difficult. We therefore again employ on a pragmatic *ad hoc* approach where success is measured by improved efficiency.

We first note that the MLE and DB estimators discussed above will lump the probability $p_{i?}$ (that a segment built in state$_i$ will contain a transition that has never been observed before) into the self-transition probability p_{ii}. Indeed, the transition probabilities p_{ij} are constructed from the number of observed segments through detailed balance (Eq. 9); the remaining probability is then assigned to p_{ii} to ensure row-stochasticity (Eq. 10). As a result, the estimator for p_{ii} is inflated (up to statistical errors) by the unknown-transition probability, $p_{i?}$, i.e.,

$$p_{ii}^{DB} = 1 - \sum_j p_{ij} = p_{ii} + \sum_? p_{i?}, \tag{11}$$

where p_{ii}^{DB} denotes the value of p_{ii} obtained by the DB procedure.

In order to disentangle these two contributions, we introduce pseudo-counts $\alpha_{i?}$ and α_{ii}. Together, these encode our *a priori* belief in the relative probability of a segment ending in the original state versus containing a transition to another state.

MLE estimators for p_{ii} and $p_{i?}$ can then be written as:

$$p_{i?} = \frac{\alpha_{i?}}{c_{ii} + \alpha_{i?} + \alpha_{ii}} p_{ii}^{DB} \tag{12}$$

$$p_{ii} = \frac{c_{ii} + \alpha_{ii}}{c_{ii} + \alpha_{i?} + \alpha_{ii}} p_{ii}^{DB} \tag{13}$$

Constraining the pseudo-counts such that $\alpha_{i?} + \alpha_{ii} = 1$, we obtain

$$p_{i?} = \frac{\alpha_{i?}}{c_{ii} + 1} p_{ii}^{DB} \tag{14}$$

$$p_{ii}^{True} = \frac{c_{ii} + 1 - \alpha_{i?}}{c_{ii} + 1} p_{ii}^{DB}. \tag{15}$$

p_{ii} and $p_{i?}$ are now expressed in terms of a single pseudo-count, $\alpha_{i?}$. Setting $\alpha_{i?}$ *a priori* is again difficult for the reasons discussed previously, but it is rather easy to assign one in retrospect: $\alpha_{i?}$ can be chosen such that p_{ii} remains unchanged as more segments are generated and observed. To do so, the initial value of p_{ii}^{DB} is recorded as soon as a state is discovered and compared to the current estimated value; $\alpha_{i?}$ is then set such that the estimate of p_{ii} remains unchanged.

This procedure allows us to *learn* a good value of $\alpha_{i?}$ for each state as segments are generated and statistics accumulate. For newly discovered states, we use the average $\alpha_{i?}$ over all states. Using these *learned* values of $\alpha_{i?}$ allows the Scheduler to scale p_{ii}^{DB} appropriately, separating p_{ii} from $p_{i?}$. It can then sample virtual trajectories from this more accurate model which accounts for the unknown-transition probability $p_{i?}$. This continuous adjustment of the "prior" pseudo-counts clearly deviates from Bayesian orthodoxy, but yields good results in practice. Again, in the current context, the statistical purity of the model is less important than the improved efficiency it delivers.

This naturally leads to the question of what should be done when the Scheduler samples a

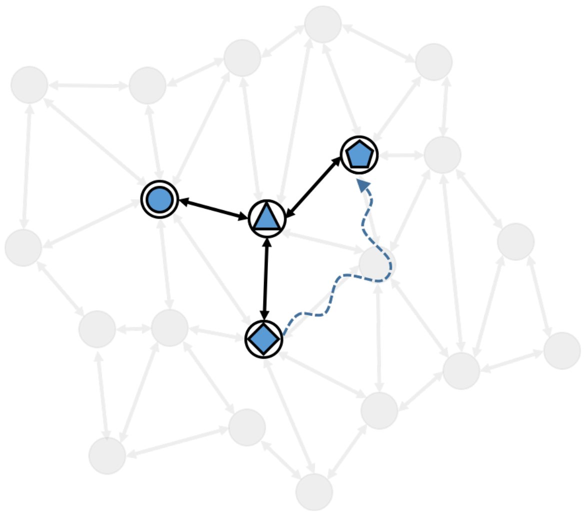


Figure 8: Conceptual view of optimal scheduling within the known state-space (blue/bold) given the underlying complete model (grey). When the segments currently scheduled are likely to yield an escape outside of the known state-space (marked by dashed blue line), it is then optimal to continue allocating the remaining resources according to where the trajectory is likely to return to the known state-space.

$p_{i?}$ virtual transition that is outside the current model. The Scheduler cannot schedule segments in states that are unknown, nor can it continue to virtually splice from unknown states.

To explore this question, consider optimal scheduling provided complete information (knowledge of all the states, transitions and probabilities), but where scheduling was confined to be only in those *known states* discovered from generated segments. Under this constraint, the optimal policy upon VE sampling an escape to an unknown state would be to continue sampling the trajectory throughout the unknown state-space until it returns to the known state-space, where segment scheduling would resume. Therefore, upon absorbing from the model, the Scheduler should continue to schedule segments conditional on the trajectory returning into the known state-space and should do so from the state in which it returns (c.f. Fig. 8).

In practice, the distribution of return probabilities into the known state-space upon exit are not known, so we resort to a simple heuristic rule. The rule is motivated by considering what this procedure would look like if it were contained in a blackbox and viewed from the perspective of the known state-space. It would seem that a virtual trajectory is sampled from the model just as before, but with probability $p_{i?}$ the trajectory suddenly disappears and reappears (i.e., *warps*) in some perhaps distant state. Following this depiction, we introduce a *warp move* designed to emulate optimal scheduling in the presence of complete information. We augment our model as described previously to account for the unknown-transition probability, and with that probability *warp* to a random state sampled uniformly over those states which can be reached in less than M jumps (according to the current model). The optimal value of M in principle depends on the connectivity of the system, but is here heuristically set to allow warping to "nearby" states and prevent warping to states which are "far" from the trajectory and are unlikely to be visited. This locality accounts for the fact that a trajectory is more likely to reenter the model close to where it left than from a completely random location. This assumption is reasonable in practice given the prevalence of super-state structures, but it is ultimately *ad hoc*, as counter-examples can be constructed.

0.4 Results

0.4.1 ParSplice Simulator

In order to test these ideas, we created a ParSplice simulator (ParSpliceSIM) where the computationally expensive MD force calls are abstracted out and replaced with stochastic dynamics via a pre-specified Markov chain defined in state-space rather than a physical configuration space. The ParSpliceSIM otherwise directly mirrors the actual ParSplice implementation. This allows for fast and systematic testing without consuming valuable computational resources. This approach has been successfully used [16, 15] to efficiently explore the design space of other Accelerated Molecular Dynamics [37] methods, to which ParSplice belongs.

ParSpliceSIM was used with toy models of varying topology. To highlight the factors that contribute to the efficiency of the proposed schemes, we consider three such models where states are arranged into a 3D lattice, a 1D lattice, and a fully connected network, respectively. These models are designed such that segments generated in a given state will remain in that state with probability p_{ii} or will transition to one of its K neighbors with equal probability $(1 - p_{ii})/K$. Each state-space consists of 729 states and is closed with periodic boundary conditions so that all states within a model have identical connectivities, and hence identical transition probabilities. In the following p_{ii} is set to 0.99, which corresponds to a mildly meta-stable state where escapes occur on a 100 ps timescale. This value was chosen to avoid the deeply metastable regime where speculation is not essential to achieve good performance. In practice, actual physical systems will be far more complex than these simplistic models, often exhibiting hierarchical structures of individual states, super-states, super-super-states, etc. In addition, they often contain a multitude of states such that the entire state-space is never fully discovered. These small toy models should then be thought of as a simplified representation of a single super-state that lies within a more complicated landscape. Therefore, while the most relevant regime in practice is when the state-space is only partially explored, these ParSpliceSIM results which show full state-discovery should be thought of as corresponding to a thorough exploration of a local super-state within a realistic system.

We report results from each of the three methods: VE scheduling with standard MLE model (corresponding to the original ParSplice method), VE scheduling with detailed balance (DB) model, and VE scheduling with detailed balance model and warp transitions (DB+W). Because these results are generated via ParSpliceSIM, we can also estimate the performance given perfect information (PI), i.e. VE scheduling using the actual Markov chain. While the underlying chain is not known in practice, this gives an upper bound on the possible performance. We simulate the use of a varying number of workers ($N = 10^3$, 10^4, 10^5) and measure the corresponding parallel efficiency (segments spliced/segments generated) throughout the generation of a trajectory containing 10^5 segments.

0.4.1.1 3D Model

Fig. 9 shows the expected decrease of the parallel efficiency with increasing N. However, the results also show that scalability is greatly improved by the use of DB and DB+W over the conventional MLE model: at 10^3 workers, the improved methods yield about a two-fold increase in performance over MLE, approaching the efficiency of perfect scheduling. This relative improvement further magnifies at 10^4 workers where DB and DB+W are providing nearly ten and fifteen times improvement in parallel efficiency over MLE (respectively) by the end of the run. The trend continues at 10^5 workers, where both DB and DB+W deliver a nearly thirty-five-fold improvement in efficiency compared to MLE. These results reflect the fact that the 3D connectivity provides a large enough number of potential paths that the trajectory is not very likely to revisit a particular state in a short amount of time, thus making the cost of overbuilding more and more severe. The improved methods are able to avoid this overbuilding through the use of DB to account for known-unknown escapes out of newly discovered states. DB+W is able to gain a further advantage by accounting

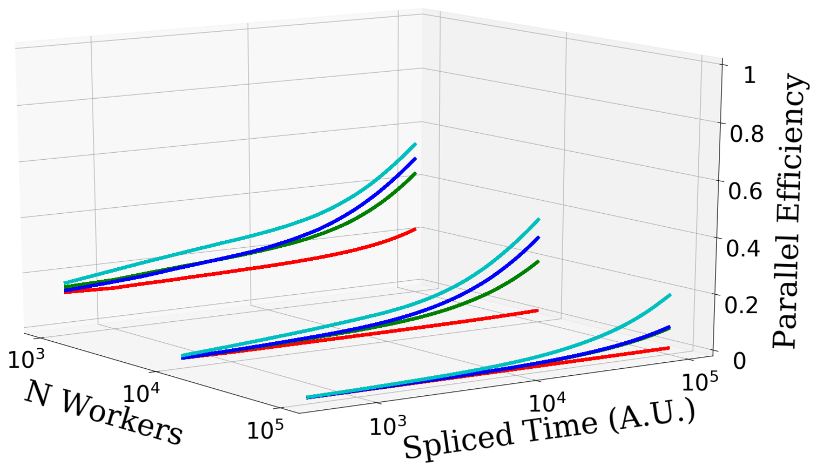


Figure 9: Parallel efficiency on the 3D model for varying N, as a function of the length of the trajectory. The different methods are shown: MLE in red, DB in green, DB+W in blue, and PI in cyan. When DB results are not visible, they overlap with DB+W.

for unknown-unknown transitions and effectively spreading out excess resources via warp transition. As resources are increased to $N = 10^4$ this slight edge becomes more pronounced, with DB+W splicing the 100,000 segment trajectory at a 50% improvement in efficiency over DB. But this edge seemingly vanishes as resources are further increased to 10^5 workers. In this case, the large surplus of resources results in DB being able to schedule *far away* from the current state, in those states that where previously only accessible through warp moves. In other words, at large enough N, both methods are scheduling in all known states, and efficiency is hence limited by the rate at which new states are discovered. As shown in Fig. 10, at $N = 10^5$ both DB and DB+W actually explore state space as fast as the topology of the transition network allows, while VE lags far behind.

It is important to note that while the rate of state discovery and parallel efficiency are often positively correlated, they can also be in opposition. Consider a Scheduler that aims at maximizing exploration by scheduling segments in those states which are least well sampled. As shown in Fig. 11, this strategy produces a higher rate of state discovery compared to DB and DB+W, but it would do so at a cost to the parallel efficiency, as the likelihood that the generated segments will be spliced into the trajectory is not be taken into account in the scheduling decision. Exploration in states that are far from the current end of the trajectory may yield the most information, but this information is of little utility if these states are not likely to be visited in the near future.

0.4.1.2 1D Model

Fig. 12 shows that the topology of the reaction network plays an important role in the performance of the different models. Now, the parallel efficiency achieved on the 1D system with 10^3 workers is only slightly improved by the use of our information-centric methods. At this N, MLE is able to amortize its overbuilding because the 1D topology naturally favors frequent revisits. In this case, all three methods approach the efficiency of PI. As more resources are utilized, the overall performance decreases rapidly while the relative difference between the three methods remain roughly unchanged. In this case, the rate at which information is acquired is strongly limited by the topology of the system. The addition of DB allows the scheduler to spread resources and expand at both

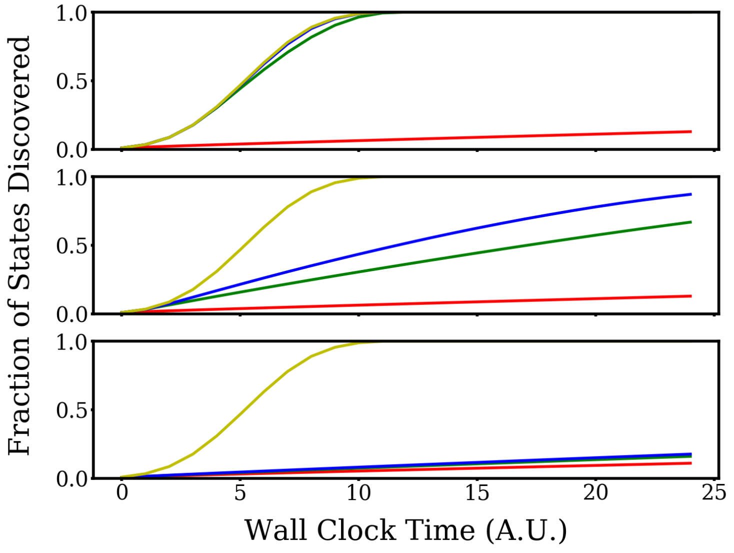


Figure 10: Proportion of states discovered as a function of pseudo WCT (where each unit is the amount of time for each worker to generate a segment). From bottom to top: $N = 10^3$, 10^4, 10^5. The different methods are shown: MLE in red, DB in green, DB+W in blue, and PI in cyan. The yellow line gives the $N \to \infty$ limit.

ends of the known state-space simultaneously, in contrast to MLE which will tend to concentrate on only one newly discovered state. This provides a slight gain in parallel efficiency. The addition of warp moves provides no significant benefit. At the highest N, the gain from DB is masked by the low parallel efficiency. DB still yields a 20% increase over MLE, but only improves the parallel efficiency from 2.5% to 3%.

0.4.1.3 Fully Connected Model

At the other end of the spectrum, we now consider a fully connected topology. As in the 3D case, Fig. 14 shows a significant improvement in the parallel efficiency by using DB and DB+W over MLE: at 10^3 workers, DB and DB+W produce a two and a half to nearly three-fold increase in performance over MLE, trailing just behind the efficiency of perfect scheduling. This relative improvement is further amplified at 10^4 workers where DB and DB+W are providing nearly fifteen and twenty times improvement in parallel efficiency over MLE (respectively) by the end of the run. The effects are even more dramatic at 10^5 workers, where DB and DB+W are delivering a fifty-fold and ninety-fold improvement in efficiency compared to MLE.

The fully-connected model now has the highest possible state-discovery rate, as each state is able to transition to any other state, and hence the entire state-space can be discovered from generating segments within a single state. In fact, on this system, all methods explore state-space at the same rate (Fig. 13 top). However, once the entire state-space is discovered, the model still must obtain accurate state-to-state transition statistics in order to properly estimate transition probabilities. This put MLE at a disadvantage as it must observe both forward and backward transition before it is able to construct accurate predictors. Meanwhile DB and DB+W are able to account for new transitions at twice the rate (Fig. 13 middle). The fully connected model, having maximal connectivity, will also present the maximal unknown-unknown transition probability in

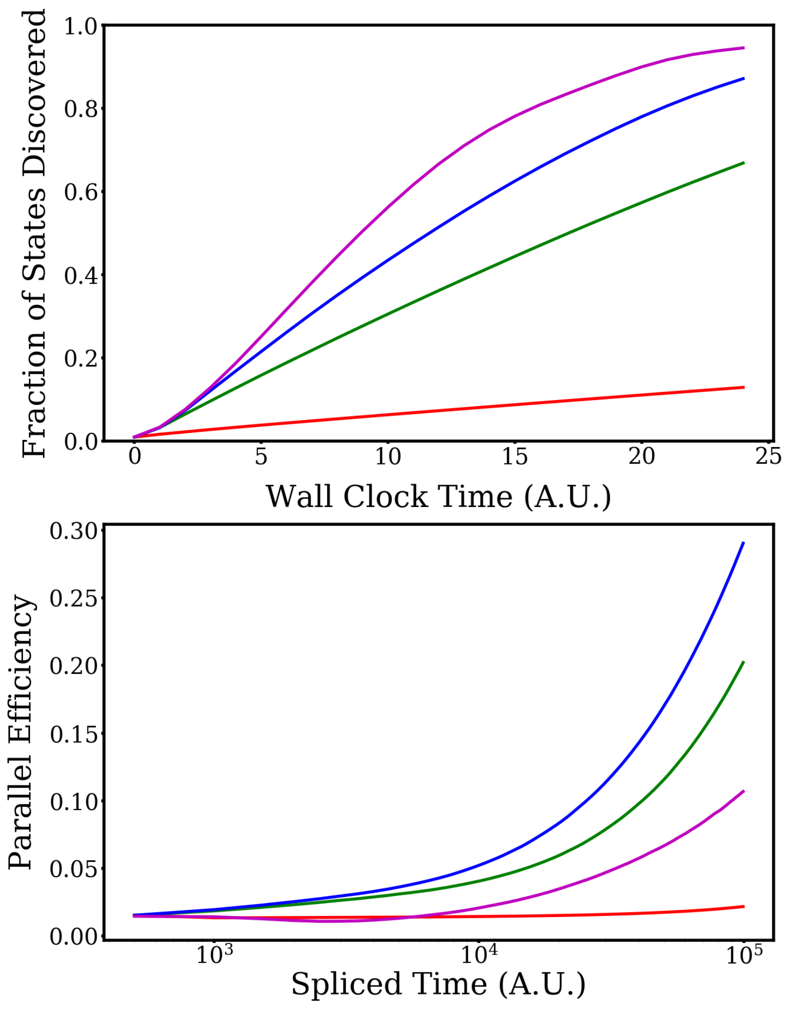


Figure 11: Performance of a pure exploration strategy (magenta) on the 3D model for $N = 10^4$ in contrast to the MLE, DB, and DB+W methods (red, green, and blue, respectively). Such a strategy has the largest state discovery rate, but nonetheless yields poor parallel efficiency, especially early in the trajectory. See text for details.

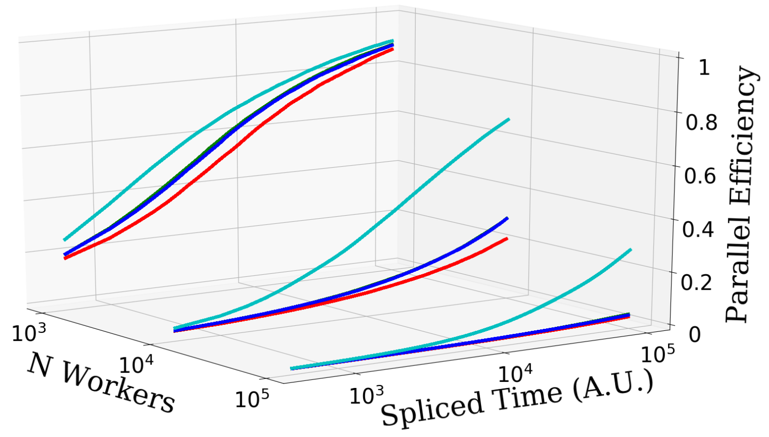


Figure 12: Parallel efficiency on the 1D model for varying N, as a function of the length of the trajectory. The different methods are shown: MLE in red, DB in green, DB+W in blue, and PI in cyan. When DB results are not visible, they overlap with DB+W.

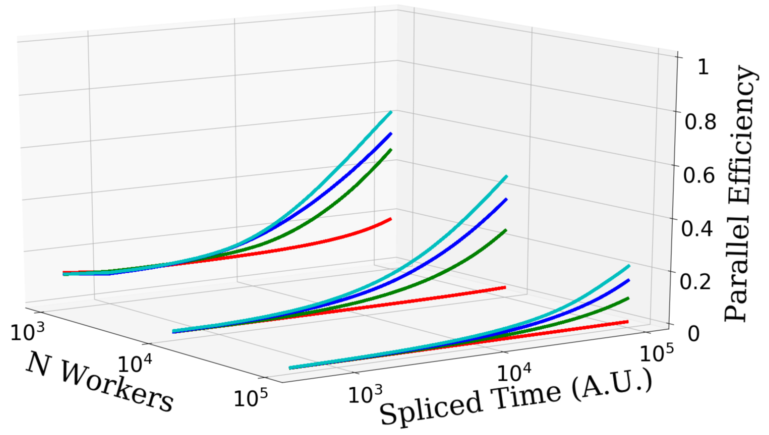


Figure 13: Parallel efficiency on the fully-connected model for varying N, as a function of the length of the trajectory. The different methods are shown: MLE in red, DB in green, DB+W in blue, and PI in cyan.

newly discovered states. Accounting for this unknown-unknown probability allows the warp-move to allocate excess resources in other states, resulting in more efficient utilization of its resources. Not accounting for this uncertainty leads DB to overestimates p_{ii}, and thus to over-schedule in the current state (Fig. 13 bottom). While the DB model gradually adjusts its prediction, DB+W does so almost instantaneously.

It is this advantage that allows DB+W to do so well (Fig. 14), tracking PI very closely, an effect that continues to holds at large N. In fact, a highly connected system is the ideal environment for DB+W; with so many transitions to be discovered, the unknown-unknown transition probability in each state will be quite significant. Conversely, this is the worst possible environment for AE scheduling; with the plethora of potential futures, the expected number of revisits to a single state over a short horizon is minimal, and thus the cost of overbuilding in a single state is maximal. Since the MLE model must resort to AE scheduling upon each first visit to a state, it struggles to deliver performance throughout the entire run as it is unable to amortize the cost of overbuilding.

0.4.2 Physical Application

We now demonstrate the improvements provided by the new statistical models with simulations of an actual physical system. We consider the evolution of a cluster of 19 vacancies in bulk Tungsten (W) at a temperature of 1400K. The simulations were carried out with the interatomic potential of Juslin and Wirth [38]. This system is relevant to the evolution of first-wall materials in upcoming fusion reactors, where tungsten is a primary candidate for the construction of so-called "divertors". The conditions in this application are extremely harsh, with the materials being exposed to high thermal loads (5–20 MW/m^2), high fluxes of He and H ions, and of fusion neutrons [39]. These neutrons can create collision cascades deep in the material, leading to the creation of point defects, which can subsequently cluster. Interstitial defects tend to be more mobile, allowing them to be absorbed at sinks such as grain boundaries or free surfaces. In comparison, clusters of vacancies are often much less mobile and tend to be left behind. Accumulation of vacancy clusters can lead to void swelling [40, 41], which is detrimental to engineering applications. Understanding the evolution of small vacancy clusters is therefore critical in order to predict micro-structural evolution of materials under irradiation [39]. ParSplice has been previously used to predict the diffusivity of small $V_m He_n$ complexes which are commonly found close to the surface of the divertor where He is abundant [42]; deeper down into the material, bare vacancy clusters are more common. In the present context, we use this system as a example of a typical application in order to quantify potential gains stemming from the use of improved statistical model. The physics of this particular system will be discussed in an upcoming publication.

All simulations were conducted using a cubic cell containing 10x10x10 unit cells (corresponding to a total of 1981 atoms). Periodic boundary conditions were applied in all directions. States were defined in terms of potential energy basins, as decribed in the Introduction. In practice, configurations extracted from MD are periodically minimized and labeled using connectivity graph techniques to identify and distinguish each state, as described in the original paper [1]. Each simulation used a correlation time of 2 picoseconds. The MD calculations were performed in the canonical ensemble using a Langevin thermostat with a friction coefficient of 1 ps^{-1}. Simulations were carried out for a fixed wall-clock-time (WCT) of 3 hour using 1,800, 3,600, or 10,800 workers. Results are averaged over between 5 and 10 independent realizations.

The landscape of states for this system is significantly more complex than the simple models discussed above. For example, Fig. 15 shows that the distribution of residence times in individual states spans about four orders of magnitude, in contrast with the model systems considered earlier where all states were identical. In this new context, naive scheduling will be adequate for some states, but will dramatically overbuild in some others. Over all the simulations, about 11,000 states and 24,000 (undirected) transitions were observed, or about 4 transitions pathways out of each state, making the topology roughly 2D-like.

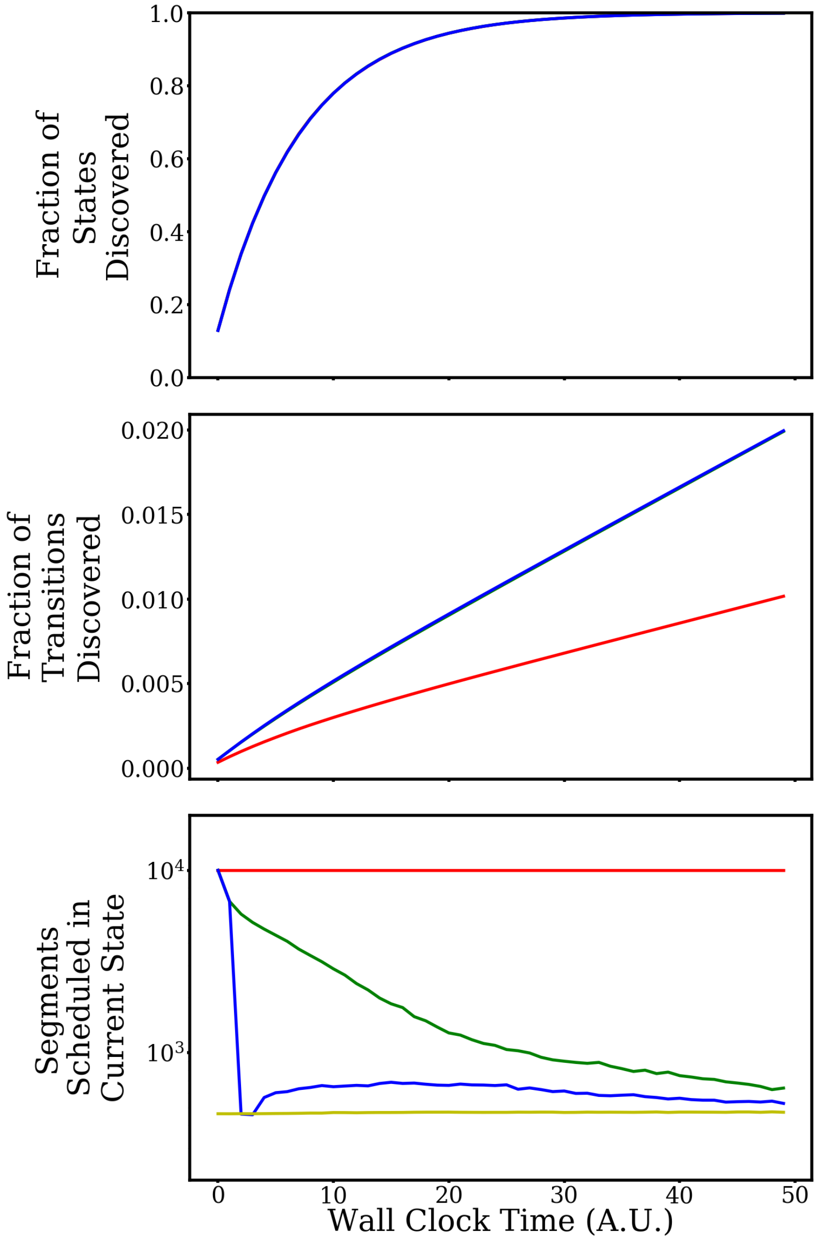


Figure 14: State-space exploration statistics gathered from 1600 independent samples of each method on the fully connected model with 10^4 workers. The different methods are shown: MLE in red, DB in green, and DB+W in blue. Where MLE/DB results are not visible, they overlap with DB+W. Top: fraction of states discovered. Middle: fraction of transitions discovered. Bottom: number of segments allocated to the current state of the trajectory (as compared to PI in yellow).

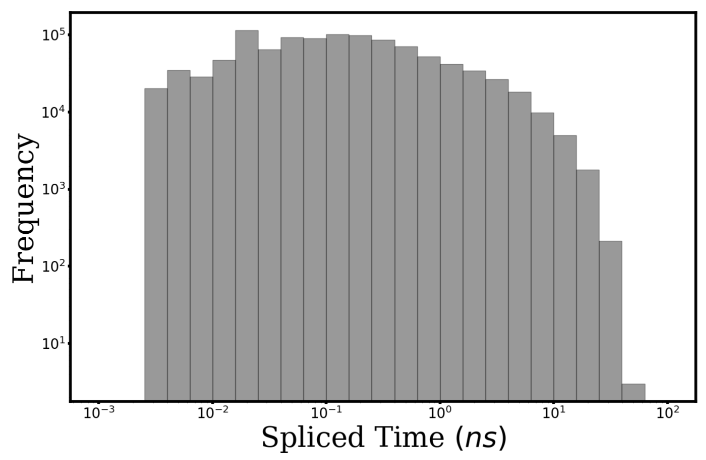


Figure 15: Frequency distribution of single-state escape times for the V_{19} W system.

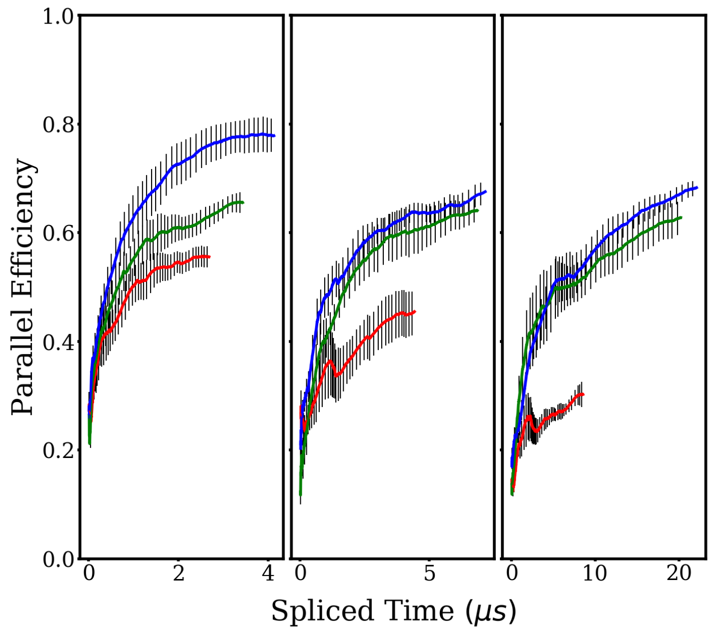


Figure 16: ParSplice parallel efficiency on the V_{19} W system for 1,800, 3,600, and 10,800 workers, respectively (from left to right). On each panel, MLE (red), DB (green), and DB+W (blue) are shown. Error bars denote the standard error of the mean over multiple simulations.

The results, shown in Fig. 16, follow the broad trends obtained from ParSpliceSIM: the parallel efficiency tends to decrease with increasing number of Workers, but the decrease is much more pronounced for MLE than for DB and DB+W. Again, DB+W provides the greatest speedup over DB at the smaller (1,800) worker count, where the contribution from the warp moves is dominant. As N increases further, the bulk of the performance improvement stems from the DB's ability to avoid over-building segments in newly discovered states. At 10,800 workers, the efficiency improvement compared MLE grows to about 2, following the steady decrease of the MLE performance. The performance of DB and DB+W is in contrast much less sensitive to the number of workers, which translates to high parallel efficiency even as the number of workers is increased past 10,000. The results are consistent with the system being an intermediate between the 1D and 3D models considered with ParSpliceSIM. Even larger improvements can therefore be expected for more inter-connected landscapes.

0.5 Conclusion

The short timescale amenable to conventional molecular dynamics are extremely crippling to many important applications in material sciences where the relevant micro-structure evolution mechanisms are thermally activated, and hence very slow. By employing a time-wise parallelization strategy, ParSplice is a promising approach to alleviate this restriction. However, ensuring the scalability of ParSplice often relies on the accurate forecasting of the future evolution of the simulation, especially as the number of workers increases. In this work, we show that the statistical models used in ParSplice can be improved by accounting for known features of the actual underlying physical model. First, we accounted for reversibility of atomistic transitions by imposing detailed balance, thereby decreasing the number of free parameters to learn by a factor of two and avoiding the serious issue of over-building segments in newly discovered states. Second, we accounted for the known presence of unobserved transitions and introduced a non-local "warp-move" that heuristically mimics excursions out of the known state-space. Our results demonstrate that significant improvement in the performance of ParSplice can result from these modifications, but that the magnitude of these improvements depends on the topology of the reaction network. These results are a promising step towards efficiently leveraging massively-parallel computational resources to extend MD simulation timescales.

Improving the scheduling procedure

0.6 Introduction

As the scale of parallel computers constantly grows, it becomes increasingly difficult for application developers to maintain strong-scalability. For example, on the Summit supercomputer at Oak Ridge National Laboratory, on the order of 100 million operations need to be executed simultaneously in order to fully utilize all processing elements. As the number of processing elements is expected to steeply increase as we approach the exascale era [25], it is paramount to develop novel strategies to maximize the amount of parallelism exposed by the applications. A now common programming model in scientific applications is task-based programming, where the execution of the application is factored into tasks of varying granularity that are then scheduled for execution using a runtime system [43, 44]. This model has proved to be powerful in a range of contexts, and his now deployed in production scientific applications [45, 46, 47, 48]. A potentially promising generalization of task-based programming to further performance at very large scales is task-level speculative execution, akin to a distributed-memory version of Thread Level Speculation [49, 50]. In our approach, (coarse) computational tasks are made available for execution before it can be established that they will definitely be used as part of the calculation. If speculations can be made accurately so that the results of most executed tasks are eventually used, this strategy has the potential to enable higher concurrency, and hence to improve scalability.

This manuscript considers the problem of optimal resource allocation in a speculative task-execution setting where a task usefulness probability, i.e., a probability that the results of a speculatively executed task will be consumed as part of the calculation (hereby referred to as the *task probability*), can be explicitly computed or estimated. This is a rich problem as, in practice, task probabilities will often be conditioned on the current state of the application, and can therefore dynamically change as execution proceeds. Further, the run time of individual tasks is often much shorter than the run time of the application as a whole. As a result, as tasks complete, freeing-up previously allocated resources, new tasks have to quickly be identified to take their place. Finally, optimal allocation of resources to tasks (i.e., how much computing resources are dedicated to the execution of each given task) is not only dependent on the individual task probabilities, but is also tied to the distribution of task probabilities of all other tasks that are available for execution.

In the following, we consider such a dynamic setting where tasks are assumed to be preemptable and restartable with a different resource allocation. It then becomes possible to periodically reevaluate the optimal allocation, pausing the execution of all running tasks, and reallocating resources as needed. This can either be done at fixed time intervals, when tasks complete, or when a change of context dictates. With each update, a newly derived optimal allocation can be executed, resuming paused tasks (with a potentially different resource allocation) as well as starting new ones if needed. This pausing and resuming of tasks allows for the optimal allocation to adapt to the dynamic variability of the system, maintaining an optimal (expected) throughput at all times. In the following, we present a generic analysis of this approach as well as a case study to a specific scientific application called Parallel Trajectory Splicing [1], which is adapted to a setting where task probabilities can be explicitly estimated.

0.7 Previous work

Modern parallel computing architectures have complex memory hierarchies as well as heterogeneous processors. In order to achieve high performance on such architectures, programming models such as Legion [44] are organized into logical regions that expresses locality and independence of data and tasks. The instances of these logical regions can be assigned to specific memories and processors in the machine during run-time. Similar logical hierarchies are also introduced in OpenMP 5.0 [51], Chapel [52], Charm++ [53], etc. for task-based parallelism. These task-based systems are capable of dynamic load balancing for scheduling and mapping tasks for optimal performance on the underlying hardware. Locality-aware parallelism has been well studied in non-speculative systems, and only a select few speculative systems utilizing parallelism via thread-level speculation (TLS) [54] or hardware transactional memory (HTM) [55] can scale beyond a few nodes. One such system that we came across is described in the work by Jeffrey et al., [56] where program knowledge is leveraged to provide *spatial hints* to indicate the data that is likely to be accessed by a speculative task. In this work, we adapt and augment this idea and speculatively schedule tasks based on their usefulness in contributing to the overall computations in order to increase throughput.

Many resource allocation strategies have been explored in the context of load balancing to efficiently use the existing hardware. A naïve way to allocate resources is to base it on peak utilization. However, designing a resource allocation strategy based on worst-case needs is not a viable approach as it results in excessive resource estimates. Many static and dynamic approaches [57, 58, 59, 60] have been proposed to distribute the problem pieces optimally over different nodes with an objective of balancing the execution time. However, the issue with most of these optimization problems is the curse of dimensionality as the search space grows exponentially with the size of the problem and the potential impact of emerging hardware, such as smart interconnects [61, 62] with advanced traffic monitoring hardware. Static approaches distributing the load during compile time have limitations as the performance is not only dependent on problem size but also over many dynamic factors. Adaptive resource management techniques [63] try to overcome these limitations by dynamically allocating resources to different processes. To provide software support, the MPI-2 standard also introduced dynamic process creation using the MPI_Comm_spawn function [64]. This function enables to create new processes during the program run-time. To mitigate poor resource allocation and load balancing in dynamic MPI spawning, fuzzy scheduling algorithms [65] for dynamic processes have been explored.

Many control-theory based techniques are also used for adaptive resource allocation that use standard feedback controllers with an auto-regressive prediction model to predict the resource allocation [66]. Many resource monitoring, prediction, and allocation strategies have been explored in cloud computing environments [67, 68, 69]. Solutions including genetic algorithms [70], neural networks, etc. are explored for prediction and allocation of resources in cloud data centers [71]. All of these approaches are based on learned behavior from heuristics and do not consider the inherent probabilities of individual tasks at an application level.

In any resource allocation problem [72], limited resources are to be allocated to a set task to maximize effectiveness. Dynamic programming has also been explored in this setting and it can be shown [73, 74, 75] that the problem can be solved using a simple sequential multi-stage dynamic programming algorithm in $\mathcal{O}(N^2M)$ time. Pipeline based algorithms [76, 77] that mimics instruction pipelines within processors have also been attempted, however, most of these approaches have a high communication cost.

0.8 Methods

0.8.1 Optimal Throughput

Consider the problem of allocating resources between a (potentially infinite) number of candidate tasks in a speculative task execution setting on a machine containing N hardware slots on which tasks can be assigned (which can be nodes, cores, GPU, etc). Tasks can be run in parallel over a certain number of slots w_i, in which case, task completion requires an expected time $T(w_i)$. For simplicity, it is assumed that all tasks are computationally equivalent; however, a task-specific $T_\alpha(w_i)$ can be introduced in the derivation below without additional complication.

Each of the candidate tasks are assigned a probability p_i of ultimately being used as part of the overall execution of the calculation, which is abstractly conceived as a workflow that progresses by consuming completed tasks. This probability can either be a rigorously derived value or a heuristic estimate. Note that each task can also be assigned a weight that reflects how much it would contribute to the calculation by scaling the corresponding p_i accordingly to obtain an expected utility. In the following, however the p's will still be referred to as probabilities, for simplicity.

To simplify and accelerate the allocation process (which is important in the context where probabilities are adjusted and resources re-allocated at a high rate), the resource assignment problem is solved in a continuous setting where the w_i are real numbers instead of integers. This enables an extremely efficient solution scheme. These values can then be discretized after optimization is complete, yielding an approximate solution but at a greatly reduced computational cost. In what follows, it is assumed that the tasks are ordered by decreasing probability. The optimal allocation of resources consists of determining the number of tasks that should be executed, M (i.e., the M tasks with the highest probabilities are selected for execution) as well as the resources assigned to each task, w_i.

The objective function to be optimized is the instantaneous expected throughput from the M tasks that are selected for execution:

$$R(M, \{w_i\}) = \textstyle\sum_i^M p_i/T(w_i)$$

where $T(w_i)$ is the expected time to complete a task when provided w_i resources. $R(M, \{w_i\})$ measures the expected rate at which useful results are generated for a given allocation $M, \{w_i\}$. The problem is constrained by requiring that the allocation fully utilizes available resources, so that $\sum_i^M w_i = N$. In pathological cases where there are more resources available than could possibly be used ($N > M * w_{\max}$, where $w_{\max}$ is the maximum allocation which a single task can fully utilize, as will be defined below) N is replaced with $M * w_{\max}$. This constraint can be enforced by introducing a Lagrange multiplier λ to the objective function.

$$R(M, \{w_i\}) = \textstyle\sum_i^M p_i/T(w_i) + \lambda(\sum_i w_i - N)$$

Extremizing the Lagrangian with respect to w_i and λ yields

$$p_i F(w_i) + \lambda = 0$$
$$\textstyle\sum_i w_i = N$$

respectively, where the function F is defined $F(w_i) \coloneqq -T^{'}(w_i)/T^2(w_i)$.

Therefore, given an explicit expression for $T(w_i)$, one can invert the function F and obtain an explicit expression for w_i: $w_i = F^{-1}(-\lambda/p_i)$, which depends only on the Lagrange multiplier λ and on the task probabilities p_i. At a given value of M, the allocation problem is reduced to solving a 1D root-finding problem in λ, $\sum_i w_i = N$, which yields the values of $\{w_i\}$ that maximizes the throughput for this value of M. Note that this formulation can yield $w_i = 0$, so that considering the first M tasks in the optimization problem is not guaranteed to allocate resources to all of them. Finally, the optimal number of tasks to consider, M^*, is taken to be the value which maximizes the

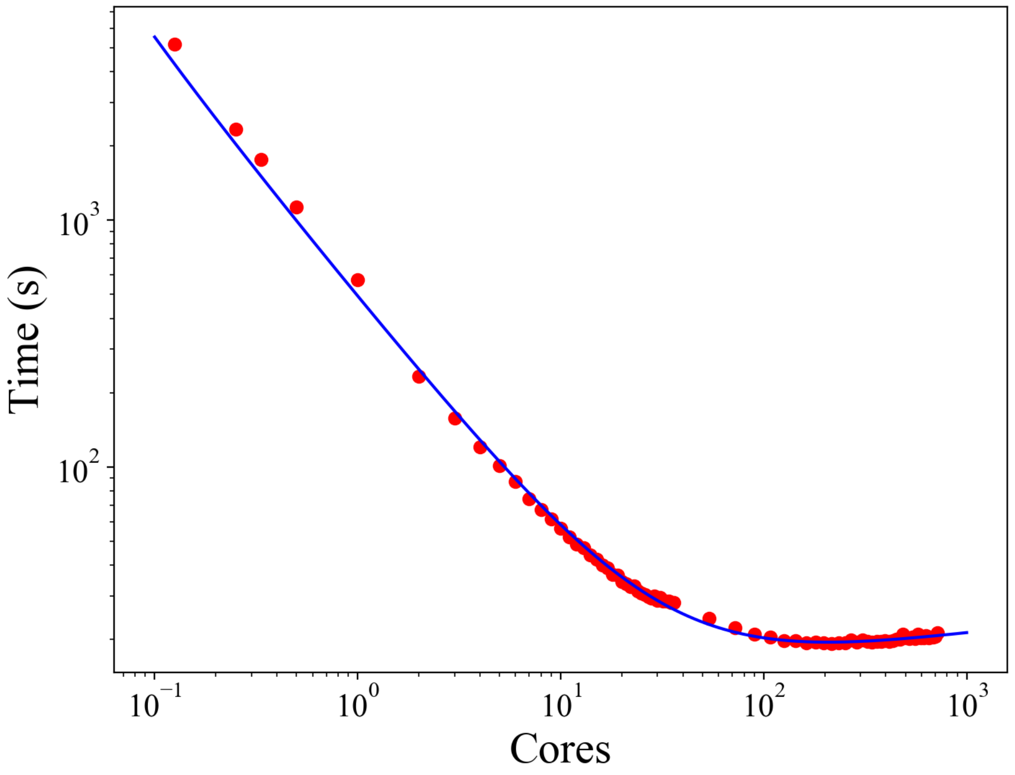


Figure 17: Benchmark analysis of the molecular dynamics code LAMMPS [2]. Run times in red were measured for an identical task executed over a varying number of cores. Fractional core counts were obtained via oversubscribing the hardware slots. In blue, a functional form $a + b/x + d\log(gx) + h/x^2$ was fit to the data to produce the invertible function $T(w)$ with coefficients $a = -2.38, b = 481.42, d = 2.32, g = 21.76$, and $h = 7.10$

expected throughput over all values of M. The allocation problem is therefore reduced to solving two embedded 1D problems, which can be done very efficiently.

In practice, an explicit expression for $T(w_i)$ is obtained by fitting to benchmark results. Benchmarks were carried out on dual sockets Intel Broadwell E5-2695V4 nodes. In section 0.9, an application to parallelized materials simulation is considered. For this work, a benchmark analysis of the molecular dynamics code LAMMPS [78, 2] was conducted as shown in Figure 17. The function $T(w_i)$ was obtained by running an identical task in parallel over a varying number of cores and recording the time to complete said task. Fractional core counts are obtained when oversubscribing the hardware slots. The recorded times were then fit to the functional form $a+b/x+d\log(gx)+h/x^2$ which was loosely based on Amdahl's law [79], adding a heuristic log term to account for the cost of synchronization and a $1/x^2$ term to provide an oversubscription penalty. The specific functional form is not crucial; other smooth approximations could have been used instead.

As shown in Figure 17, $T(w)$ possesses a minimum, after which the time to execute a task begins to increase with increasing resources due to communication and synchronization overheads. As no optimal resource allocation can include w's in this regime (because a higher throughput could always be obtained with even fewer resources) this branch of the function $T(w)$ is ignored when obtaining F^{-1}. The minimum of $T(w)$ therefore defines the maximum allocation ($w_{\max}$, roughly 200 cores in this case) which can be fully utilized by a task, and hence a corresponding minimum time in which a task can be completed; here $T(w_{\max})$ is roughly 19.5 seconds. This quantity becomes important in conjunction with $T(1)$, the time to complete a task at the maximum parallel efficiency, as their ratio will be shown to correspond to an upper bound of achievable performance gains. In addition to ignoring the $w > w_{\max}$ branch, the domain of F is restricted to those values of w where F is monotonically decreasing, which is required for the solution to be a maximum of the throughput, in contrast to a minimum. F is therefore invertible so that F^{-1} is well defined.

0.8.2 ParSplice

In the following, the potential benefits of optimal resource allocation in a speculative task execution setting are demonstrated by studying an existing scientific application called Parallel Trajectory Splicing, or ParSplice [1, 26]. ParSplice is a method in the family of Molecular Dynamics (MD) simulations. MD numerically integrates the classical equations of motion of atoms using interatomic forces derived from the gradient a user-provided potential that describes the interactions between atoms. MD is broadly used in the computational sciences, with applications to materials science, biology, chemistry, etc. MD is extremely powerful, but also computationally intensive. While domain-decomposition approaches enable the use of massively-parallel computers to extend the simulation length-scales [78], similar approaches do not allow for significant extension in timescales except for very large systems, due to communication and synchronization overhead. Extending timescales instead requires specialized techniques [8, 11, 14, 20, 22]. ParSplice is one such technique where parallelization is carried out in the time domain, thereby avoiding synchronization costs inherent to domain decomposition. It however comes with a tradeoff: instead of generating a trajectory that is continuous in phase space, it produces a discrete state-to-state trajectory, where a state corresponds to a finite volume in the phase space of the problem. States are usually defined to correspond to long-lived metastable topologies of the system (such as the attraction region of deep local energy minima), and so state-to-state trajectories are sequences of transitions between such long-lived states.

ParSplice works by concurrently and asynchronously generating many short “segments” of MD trajectory in such a way that they can later be spliced together to create a single state-to-state trajectory. Generating a “segment” involves creating an independent realization of the system’s trajectory (by solving a stochastic differential equation) that is initialized in some assigned starting state and evolved through MD until a physics-motivated stopping condition is achieved, after which the final state visited by the trajectory (which may or may not be the same as the starting state) is noted. So, in short, a segment is composed of an initial and a final state, separated by some MD time (see Figure 18). These segments are then returned to a database where they are stored until they can be spliced. Due to the specially-designed protocol by which segments are produced and stored [31], any segment in the database can be spliced onto any other so long as it began in the same state that the other finished (see Figure 19). This allows for a single state-to-state trajectory to be formed by extracting individual segments from the database and splicing them onto the end of the trajectory. For further details on how the independent generation and splicing of segments is guaranteed to produce statistically correct state-to-state trajectories, the reader is referred to the original manuscript [1].

Because the individual segments are independently produced in parallel, ParSplice can offer a potential wall-clock speedup proportional to the number of MD instances. Achieving this ideal level of parallel efficiency however requires that every segment generated is eventually spliced into the state-to-state trajectory. Therefore, while the accuracy of a trajectory is ensured solely by the independent generation and splicing of segments according to the ParSplice prescriptions, the efficiency of ParSplice is a function of its ability to forecast ahead of the trajectory and assign segments to be generated in those states where they are most likely to be needed. As such, ParSplice

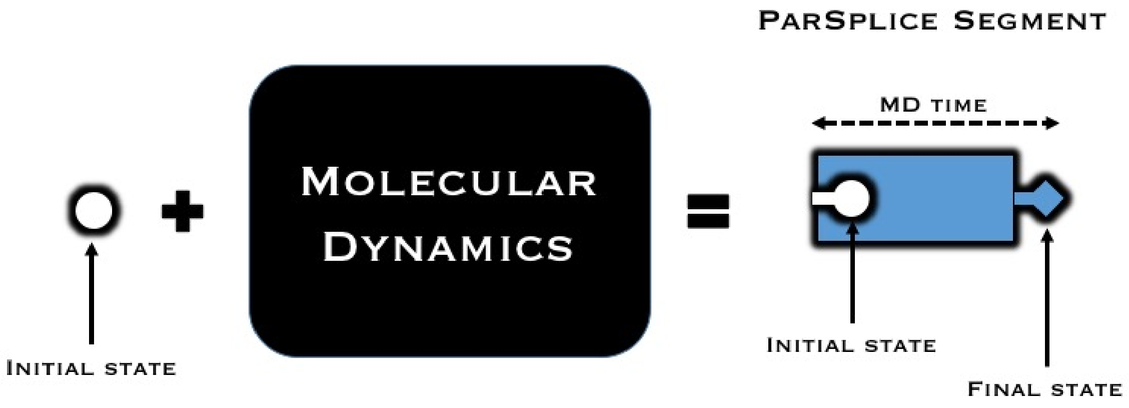


Figure 18: Conceptual illustration of segment generation: An MD trajectory is initialized in some assigned “circle” state and then dynamically evolved forward in time through MD. After some stopping criteria is met, the final state of the MD trajectory is noted and used to produce a ParSplice “segment”.

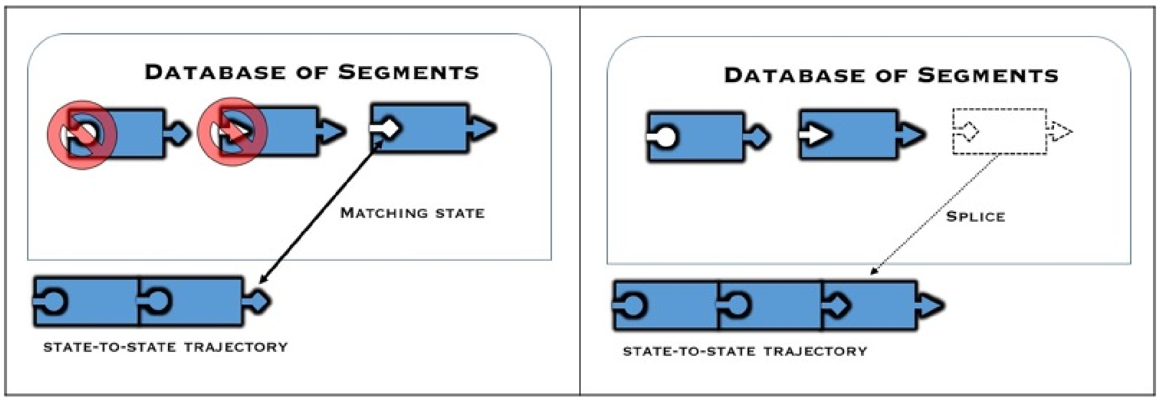


Figure 19: Conceptual illustration of segment splicing: Left panel, only segments which start in the same state that the previous spliced segment ended (here the “diamond” state) can be spliced. Right panel, splicing a segment involves extracting it from the database and appending it to the state-to-state trajectory.

follows the speculative execution paradigm discussed above: at any point in time, only one segment is strictly guaranteed to be spliced into the trajectory (a segment that begins in the state where the trajectory currently ends), but one can identify a much larger number of segments that could potentially be spliced in the future. Towards this goal, ParSplice develops a discrete time Markov Model (MM) on-the-fly from the previously generated segments and uses this model to assign starting states for new segments to be generated. The MM encodes the estimated probability that a segment generated from state i will end in state j. In actual simulations, the MM is usually empty at the beginning of the simulation and it is dynamically updated as more segments are generated.

The original ParSplice method selects segments for execution through a procedure referred to as virtual end (VE) scheduling. VE accounts for completed but unspliced segments which are stored in the database as well as those "pending" segments which have been assigned to some computing resources, but have not yet been completed. The process of VE assigning the state in which the next segment should be generated is outlined in Figure 20. 1) The MM is used to sample "virtual" endpoints for all of the pending segments, creating a prediction of what the database might look like once all of the pending tasks are completed. 2) It then "virtually" splices from this database-prediction onto the end of the state-to-state trajectory until it runs out of segments to splice. 3) It assigns the next segment to be generated starting in the state where the state-to-state trajectory "virtually" ended. This process is then repeated for next segment state-assignment, and so on until a segment has been assigned to any idle MD instance. The word "virtual" is used to denote that this process is not actually manipulating segment endpoints or splicing onto the actual physical trajectory. This process is simply used as a means of forecasting where to assign the next segment and only segments that were actually completed can be spliced into the physical trajectory.

In the present context, an important limitation of the VE procedure is that it samples from the ensemble of possible tasks according to their probabilities, but does not directly give access to the individual task probabilities themselves. In order to address this limitation, a new variant of ParSplice is proposed where instead the task probabilities are first explicitly estimated, and then tasks with the largest probabilities are selected for execution. This new variant is referred to as MaxP (maximum probability) scheduling. The derivation of MaxP is based on the formalism of discrete time Markov Chains, and is detailed explicitly in Appendix A. The general concept involves calculating the probability that particular segments will be spliced into the state-to-state trajectory over some finite time horizon, as an average over all paths that the spliced trajectory could take. These probabilities can be computed analytically or approximated via a computationally cheaper Monte Carlo approach. See Appendix A for details.

The MaxP formulation provides a natural estimate of the task probabilities for each segment that could be generated, i.e, each potential task. It is important to note that the probabilities derived from the MaxP formalism are dependent both on the instantaneous content of the database and on the current end point of the trajectory, as it was the case for the VE variant. The probabilities therefore continually change as the simulation proceeds, which suggests that it might be advantageous to periodically re-adjust/recalculate the probabilities and re-assign resources to tasks so as to maintain an optimal expected throughput. Further, MD is inherently preemtable and restartable: the only information needed to checkpoint and restart a simulation is the list of the current positions and velocities of the atoms. Using this checkpoint, the simulation can be restarted with a different domain decomposition, and hence with a different w. The resource allocation approach discussed above is therefore directly applicable to ParSplice-MaxP.

0.9 Application

To gain a better intuition of the solutions resulting from different task probability distributions, and of the potential performance improvements that can be expected by allocating resources based on task probabilities, we first discuss results on various synthetic distributions. More specifi-

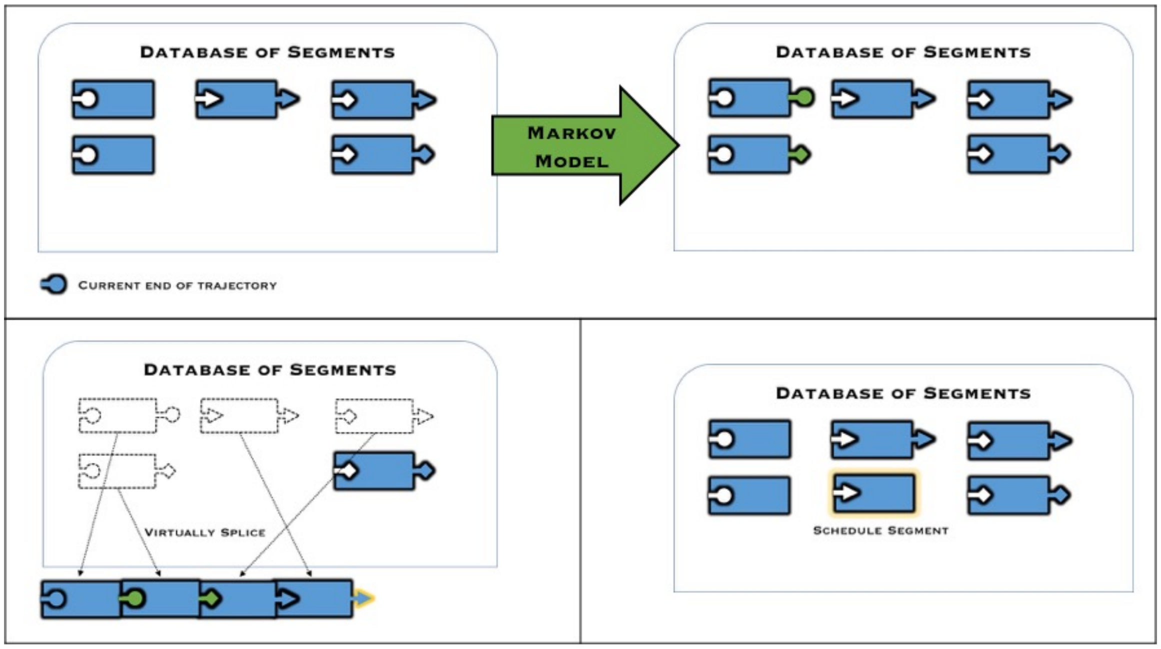


Figure 20: Virtual End (VE) scheduling of segments: Top panel, the statistical Markov Model (in green) is used to sample "virtual" end states (also green) for all pending-segments, speculating on what the database might look like once these pending-segments are completed. Bottom-left panel, segments are then "virtually" spliced from the speculative database, extending the "virtual" trajectory as far as possible. Bottom-right panel, a new segment (outlined in yellow) is scheduled to begin in the state where the "virtual" trajectory ended. We stress the word "virtual" here to differentiate from anything actual. All segment manipulation is only carried out as a thought experiment for determining where to generate the next segment.

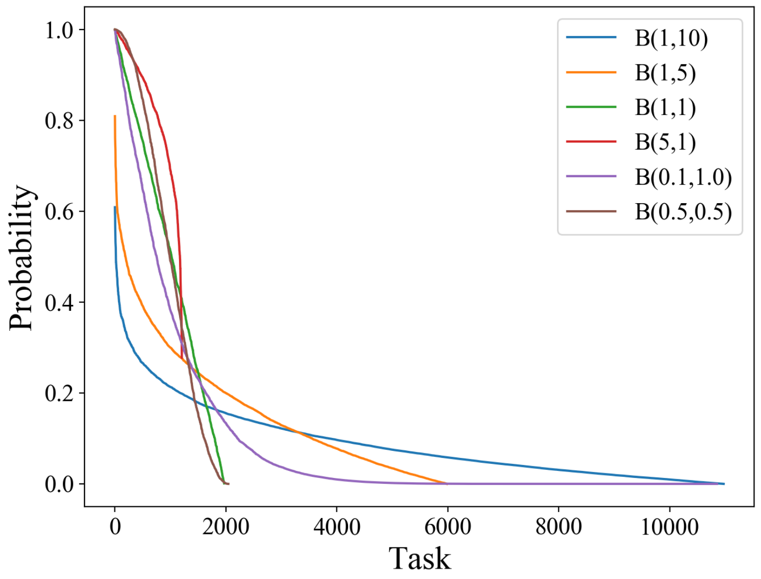


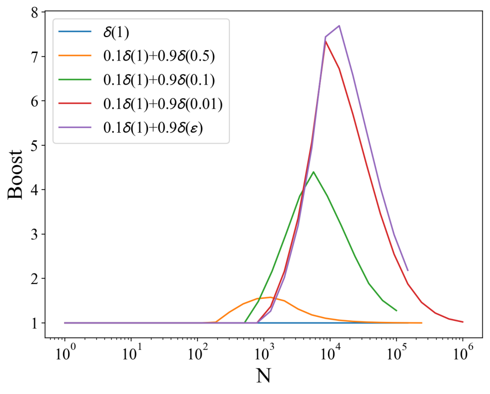
Figure 21: Synthetic task probability distributions sampled from different $\mathrm{B}(\alpha, \beta)$ distributions as depicted in the legend.

cally, we focus on the characterization of the instantaneous throughput obtained by optimizing the resource allocation as a function of the characteristics of the task probability distribution. Each of the following distributions were created by drawing random p_i samples from a probability density until a given total $\sum p_i = 1000$ was reached. While this process resulted in each synthetic distribution containing a different number of potential tasks, the constrained value of 1000 ensures that the maximum expected throughput given infinite resources is identical for each distribution, and hence comparisons can be made easily.

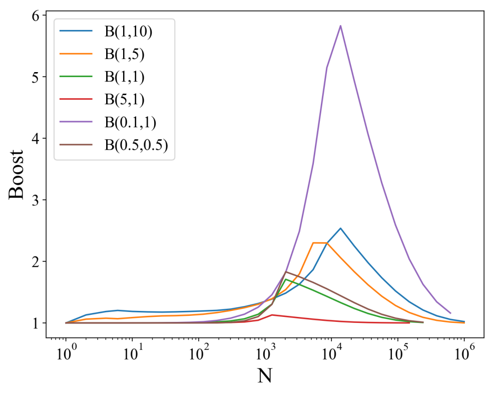
The probability densities from which the synthetic distributions were sampled belonged to one of two generic classes. The first was a delta distribution or composition of two delta distributions from which only particular values of p_i could be sampled. Each composition of delta distributions contained a non-zero peak at $p = 1$, corresponding to having a certain number of tasks which are known to be essential (i.e $p = 1$), and another non-zero peak at lower p, corresponding to a certain number of speculative tasks which are assigned a generic probability. As one would expect to generally have a large number of speculative opportunities, and thus of speculative tasks, the magnitude of the peaks were weighted in favor of the lower probability by a 9:1 ratio. Sampling from these distributions yields a task probability distribution which exhibits a “step” from the $p = 1$ tasks to the speculative probability. In addition to a single delta distribution at $p = 1$, which generated a trivial distribution containing only $p = 1$ tasks, several composite distributions are analyzed with varying values for the lower-probability speculative tasks.

The second class of probability densities were beta distributions, $\mathrm{B}(\alpha, \beta)$. Adjusting the shape parameters α and β allows for the creation of a wide range of different distributions, as illustrated in Figure 21. Sampling from the continuous probability densities produced nearly continuous task probability distributions capable of spanning the entire $[0, 1]$ probability domain. This assortment of synthetic task probability distributions provide a reasonable collection for surveying the performance landscape of the proposed optimal resource allocation method.

The most important question in practice is whether the effort of deriving and implementing



(a)



(b)

Figure 22: Boost in performance as a function of resources N, where Boost is defined as the ratio of expected throughput provided the optimal allocation to the expected throughput provided the naive allocation. Results shown for synthetic distributions sampled from both the delta distributions (a) and the beta distributions (b).

a probability-aware optimal allocation scheme is worthwhile as compared to a naive approach which does not consider the probability of tasks. Such a naive scheme would assign resources in equal sized chunks corresponding to the maximal parallel efficiency, so as to maximize throughput in the non-speculative setting. It would only deviate from this chunk size if the resources available enabled all tasks to be run at maximum parallel efficiency and excess resources remained. In such a case, the constant chunk size allocated to each task would uniformly increase to fully utilize all available resources. Therefore, the naive allocation is to assign each task with $w_{\text{const}} = \max(1, N/M)$ resources to each task. In the following, it is shown that the increase in throughput due to optimal allocation can in fact be quite substantial.

We first recognize from the blue curve in Figure 22a that the trivial probability distribution where all tasks are of equal probability ($p_i = p$), corresponding to task probabilities sampled from a single delta distribution, obtains unit boost in performance compared to naive scheduling throughout the entire range of N. This was expected given that when all probabilities are equal, the throughput is maximized for a uniform allocation of resources. The natural extension to this trivial case of uniform task-probabilities is the case of binary probability values, where tasks are assigned one of two probabilities: p_a and p_b (where $p_a > p_b$). Such synthetic task probabilities are sampled from a composition of delta distributions, e.g., constructing a list of containing certain ($p_a = 1$) and speculative ($p_b < 1$) tasks. An example of a step distribution arising in practice might be an application which identifies a certain number of tasks as provably necessary (and hence for which $p_a = 1$), and a certain number of speculative tasks, which are assigned a generic probability p_b. Synthetic task probability distributions were sampled for four different values of speculative task probability ($p_b = \{0.5, 0.1, 0.01, 10^{-10}\}$). It is seen in Figure 22a that the boost obtained through the optimal allocation varies inversely with p_b and saturates as p_b approaches zero.

Take for example the resource allocation of $N = 10,000$ to a probability distribution characterized by $p_a = 1$, $p_b = 0.01$. The naive allocation distributes resources evenly across all possible tasks, producing an expected throughput of roughly 2.3. This is in stark contrast to the optimal allocation, which concentrates resources only to p_a tasks. As a result, the optimal allocation executes fewer tasks, but does so yielding a higher expected throughput of roughly 16.8. The optimal allocation therefore results in a substantial boost in performance, producing nearly 7.3 times the expected throughput of the naive allocation. We note that the boost also affects only an intermediate

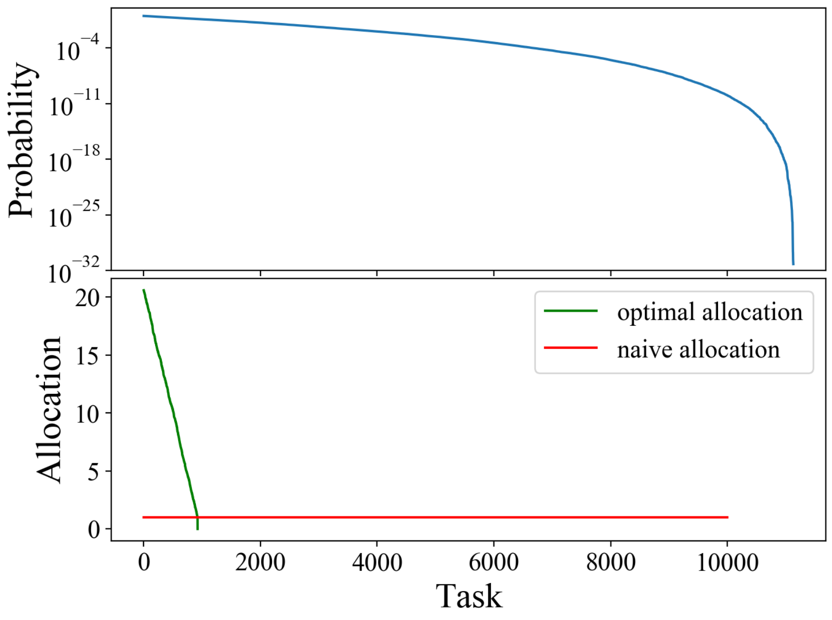


Figure 23: Top: Task probability distribution sampled from $B(0.1, 1)$ distribution. Bottom: Allocation of $N = 10,000$ resources among tasks.

range of values of N, as, in both the small and large N limits, the optimal and naive allocation are identical.

The value of p_b relates to the potential boost obtained as it affects the sampled task probability distribution in two key ways: 1) The smaller values of p_b present steeper decays in the probability distributions as they cover a larger range in values. The naive allocation struggles to handle a large range in values as its allocation is uniform, meanwhile the optimal allocation is specifically tailored to the individual probabilities of each task. 2) Because the values (p_a, p_b) are sampled in a 1:9 ratio, a smaller value of p_b implies that more tasks will be sampled before reaching the $\sum p_i = 1000$, and thus the distribution of tasks will have a longer tail. This, again, is not handled well by the naive allocation as the high $p = p_a$ tasks will receive the constant $w_{\text{const}} = 1$ allocation unless N is such that all tasks can be executed, at which point w will start to increase uniformly. The longer the tail in the distribution, the more resources are needed before the naive allocation will increase the uniform chunk size, and hence the throughput.

These results illustrate the intuitive idea that ignoring the task probabilities and invoking a uniform allocation often involves running lower probability tasks with resources that could be better spent increasing the allocation to higher probability tasks. These two key features (steep decay and long tail) are particularly detrimental to the performance of the naive allocation scheme.

Considering the task probability distributions sampled from the Beta probability density, one can see that this rule of thumb is upheld. Figure 23 illustrates allocation solutions for $N = 10,000$ given a task probability distribution sampled from $B(0.1, 1)$. The sampled probability distribution consisted of 11,132 tasks and spanned a range of probabilities from $p \sim 1$ to $p \sim 10^{-32}$. The naive allocation allocated resources uniformly, executing $10,000$ tasks with $w = 1$. This resulted in an expected throughput of just over 2. The optimal allocation provided resources in greater chunks to fewer tasks. It only executed 923 tasks, but did so yielding an expected throughput of nearly 12, providing a boost in expected throughput of nearly 6 times the naive allocation.

We see here again how the long tail and steep decay of the task probability distribution meant that the naive solution would allocate resources to low probability tasks that contribute little to the expected throughput. It is instead optimal to allocate additional resources to those higher probability tasks, running fewer tasks but generating a higher expected throughput.

The maximum attainable boost one could possibly obtain can be determined by the following analysis. Consider a task probability distribution consisting of N_a tasks of probability $p_a = 1$ and N_b tasks of probability $p_b = \epsilon$. The optimal allocation would divide resources among those N_a tasks, ignoring the N_b tasks. This would yield an expected throughput of $N_a/T(N/N_a)$. The naive allocation would spread resources among all tasks, yielding an expected throughput of $N_a/T(1) + N_b\epsilon/T(1)$ if N was sufficiently large such that $N = N_a + N_b > w_{\max}N_a$. In the case where $N_b\epsilon << 1$ this expression would simplify and a trivial relation for the boost in expected throughput would result as the ratio $T(1)/T(N/N_a)$. This expression is maximal when $N/N_a = w_{\max}$. Thus, the maximum attainable boost one could obtain is equal to $T(1)/T(w_{\max})$, which for our application was roughly 25.

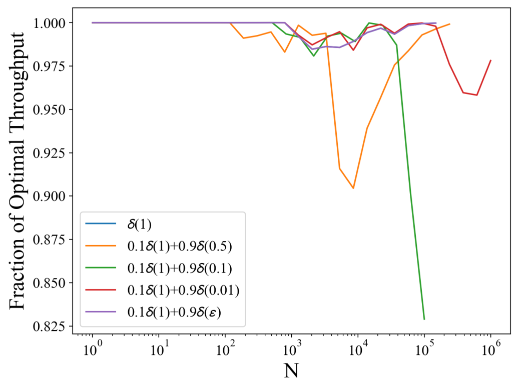
0.9.1 Constant w

The assortment of synthetic distributions surveyed above shows a diverse range of optimal task allocations and the corresponding boost compared to the naive scheduling strategy. These task allocations are guaranteed to provide the greatest expected throughput for a given distribution, at the cost of increased code complexity. One may instead consider a simpler approximate solution where each executed task is provided the same allocation, but this allocation is allowed to differ from the naive strategy. Certainly this simplified scheme would be suboptimal, but it is unclear by how much. In the trivial case of constant probability, for example, the optimal allocation was a constant allocation. The same was true for the step distributions when resources are limited. In fact, it is often the case that a constant allocation can achieve a throughput close to that of the optimal allocation. Figures 24c and 24a show what fraction of the optimal expected throughput can be achieved when an optimal constant allocation is provided for each of the distributions surveyed above. For most values of N there exists a constant value of w which can provide upwards of 90% the expected throughput that the optimal allocation would yield. This is, however, not always the case. When the task probability distribution possesses a major discontinuity (as seen in the step distributions), there exists a range of N values where even the best constant value is largely suboptimal.

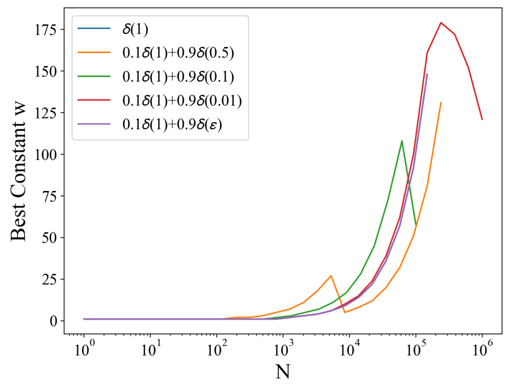
Furthermore, as seen in figures 24d and 24b, the value of the best constant w can greatly vary depending on the resources available and on the specifics of the task probability distribution. Furthermore, in a setting where the task probability distribution is dynamic and/or context dependent, the precise value of the best constant allocation will change in time. This presents a major difficulty in assigning a single constant value of w that will be allocated to each executed task; what may be a “good” value of w at one point in time might be a poor value sometime later. To maintain an expected throughput that is near optimal for an evolving task probability distribution, one would have to consistently tune the constant value of w. Devoting this effort to maintain a near-optimal solution is uneconomical as one could ensure the truly optimal solution with similar efforts.

0.9.2 ParSplice Simulator

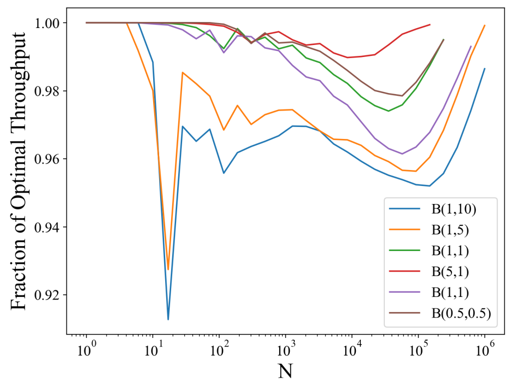
In order to assess the potential performance gains accessible with this new approach without the considerable time investment required to rewrite the ParSplice production code, we instead chose to make use of a simulator; a strategy which has proved beneficial [15, 16] in developing Accelerated Molecular Dynamics [37] methods, to which ParSplice belongs. The ParSplice Simulator (ParSpliceSIM) was designed to directly mirror the logic of the actual ParSplice code with the exception that dynamics are generated from a user-specified Markov Chain that can be used



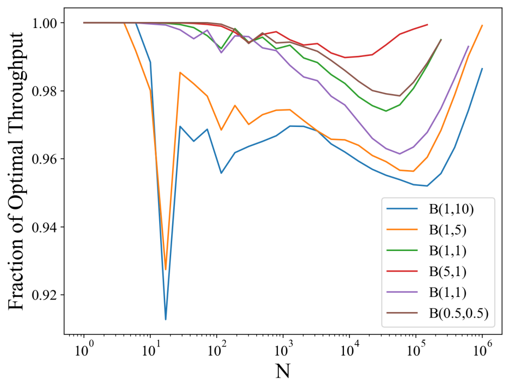
(a)



(b)



(c)



(d)

Figure 24: On left, fraction of optimal throughput which can be achieved with a constant allocation for the Delta sampled distributions (24a) and Beta sampled distributions (24c). On right, the corresponding value of w required to obtain this fraction of optimal throughput for the respective Delta sampled (24b) and Beta sampled (24d) distributions.

to statistically sample segment endpoints, rather than using computationally expensive MD calculation [80]. ParSpliceSIM is therefore computationally light, simple, and runs in serial. As the matter of interest is measuring the potential improvements in performance, rather than obtaining correct atomistic trajectories, the ParSpliceSIM provides an ideal framework for testing the proposed ParSplice variant described in this work.

ParSpliceSIM was used to compare the performance of the existing VE method to the newly developed MaxP method, which was then further enhanced to allow the periodic pausing of segments and reallocation of resources following the optimization procedure described above. The effect of these successive developments are shown on a range of different model systems to illustrate the expected gains in performance. As to separate the effect of the current work from that of intrinsic model uncertainty, it was assumed that the MM created on-the-fly by ParSplice would provide a sufficiently accurate representation of the dynamics, and was instead replaced with the actual underlying Markov Chain within the ParSpliceSIM. The metric of performance in evaluating the different methods is the amount of pseudo-MD spliced for a given wall clock time (WCT), which is a direct measure of the scientific value of a simulation.

The following ParSpliceSIM results were generated using an assortment of Markov Chains with varying state connectivity. Each state within a particular Markov Chain was endowed with a ρ_{ii} probability of not escaping the current state, and had an equally likely probability of transitioning to one of it's K neighbors ($\rho_{ik} = (1-\rho_{ii})/K$) with K defined by connectivity. Periodic boundaries were implemented to ensure each state within a particular Markov Chain had the same state connectivity. To mimic the environment of a true atomistic simulation, the number of states in a particular Markov Chain was set to 8000 such that the vast majority of state-space remained un-visited by the trajectory throughout the duration of the simulation.

In order to evaluate the true potential of the developed methods, each simulation was provided a resource allotment N that was many times greater than the expected number of segments needed to escape ($\langle n_{\text{escape}} \rangle$) from a state. This is the regime of greatest interest as it is where speculation significantly affects the efficiency of ParSplice. When the resources are not greater than $\langle n_{\text{escape}} \rangle$ all resources can be allocated to the task(s) of building in the current state and will be amortized with high probability. Prior to the current work, ParSplice attempted to best utilize those additional resources (those which were not likely to be needed in escaping the current state) to speculate on where the trajectory was likely to go next. The main purpose of this section is to assess whether further efficiency gains are possible by dynamically assigning resources based on expected utility.

In what follows, each simulation was carried out with assigned $\rho_{ii} = 0.99$, corresponding to $\langle n_{\text{escape}} \rangle = 100$, and a resource allotment of $N = 5000$. With an N being 50x greater than $\langle n_{\text{escape}} \rangle$ ParSplice can trade-off resources in order to obtain an escape from the current state faster. This tradeoff would be beneficial in cases where speculation was futile, but could be poor in cases where accurate speculation were possible. To illustrate this effect, results are shown for three different Markov Chain toy models of increasing connectivity: 1D representing dynamics on a line, 3D representing dynamics on a cubic lattice, and fully-connected, where each state is connected to every other state. The greater the connectivity, the more difficult it is to speculate on the trajectory's future as the possible paths exhibit exponential branching. Conversely, when the connectivity is low speculation can be quite accurate. While these models of state connectivity are much simpler than what would be observed in actual applications they do however provide relevant information and general guidelines on the potential performance of the method in different scenarios.

These three toy models present a good assortment for testing the new methods. The 1D model provides rather-predictable dynamics for which speculation will be quite fruitful. The 3D model presents dynamics which are somewhat less predictable, and where effective forecasting will likely be limited to within a small neighborhood from the current state. Finally, speculation is futile for the fully-connected model where the number of branching paths is immense.

Performance results for each of the toy models are shown in Figure 25, displaying the pseudo-MD spliced as a function of WCT. Each subfigure shows the results of five different methods for its particular model. The different methods consist of the existing VE formalism, the newly introduced MaxP formalism, and MaxP with preemption and restarts. The last method is shown implementing three distinct allocation policies: 1) The (naive) maximum throughput allocation; distributing resources evenly ($w = w_{\mathrm{const}}$) to execute the most tasks at the highest throughput, thus producing the maximum number of segments for a given N. 2) The minimum time allocation; distributing resources evenly to execute tasks with the maximum allocation (as defined previously, $w = w_{\max}$) thus producing $N/w_{\max}$ segments as quickly as possible. 3) The (optimal) maximum expected throughput allocation; distributing resources according to how likely tasks are to be spliced onto the state-to-state trajectory, thereby balancing the tradeoff between time and throughput to produce the most spliced segments as quickly as possible.

The first thing to note in Figure 25 is the small but appreciable increase in performance that results from transitioning from the VE to the MaxP formalism. While MaxP was introduced to allow for the implementation of our derived methods, it is worth noting that the transition does not come at the cost of performance, to the contrary.

Further improvement resulting from the ability to pause and reschedule segments can be substantial depending on the topology of the state space. As was discussed previously, the 1D toy model presents very limited connectivity, therefore corresponding to a system which is highly susceptible to speculation. As a result, the distribution of task probabilities will decay slowly, and the balance between running a few tasks very quickly and maximizing the overall task-completion throughput will be more heavily skewed toward throughput. This is seen from the 1D results in Figure 25 as the maximum throughput allocation outperforms the minimum time allocation (which even under-performs the standard VE and MaxP) by a factor of three. However, even in a highly predictable system like the 1D toy model, the balance between time and throughput is not completely one-sided. This is seen as the optimal allocation (which aims to maximize the *expected* throughput, or segments spliced) is able to further improve performance by a factor of two as it strikes the optimal balance. Overall, the implementation of our derived methods applied to the 1D model are able to more than double the pseudo-MD spliced over the same WCT as compared to the standard VE method.

The 3D model presents a slightly different picture as long-time speculation is somewhat difficult due to the increased connectivity, yet short-time speculation can still be profitable, thus this model requires a more delicate balance between throughput and execution time. This is seen from the 3D results in Figure 25 as now the minimum time allocation outperforms the maximum throughput allocation by over 50%. The optimal allocation adapts to the new model and achieves the best performance, more than doubling the efficiency of the the minimum time allocation strategy and providing nearly a six-fold improvement as compared to the standard VE method.

Lastly, the fully-connected model is considered, for which speculation is futile and escaping from the current state as quickly as possible is the only sound strategy. As expected, the fully-connected results in Figure 25 show how the minimum time allocation greatly outperforms the maximum throughput allocation by nearly an order of magnitude. However, the optimal allocation is able to further improve performance by nearly doubling the throughput achieved over the simulated times shown here. Although the minimum time allocation utilizes resources to generate segments as quickly as possible, it does not achieve the desired result of escaping from the current state as quickly as possible. This is because the number of segments it produces is likely insufficient to escape the state, i.e less than $\langle n_{\mathrm{escape}} \rangle$. It is actually better to generate more segments (greater throughput) at a slightly slower rate (but higher efficiency) such that a sufficient number of segments to escape from the current state are generated. Overall, our derived method applied to this toy model of greater connectivity enabled nearly twenty times the pseudo-MD to be spliced over the same WCT as compared to the standard VE method.

The resulting improvement of our derived methods, as compared to the existing VE scheduling method, showed a nearly 2.5x, 6x, and 20x boost in performance for 1D, 3D, and fully connected

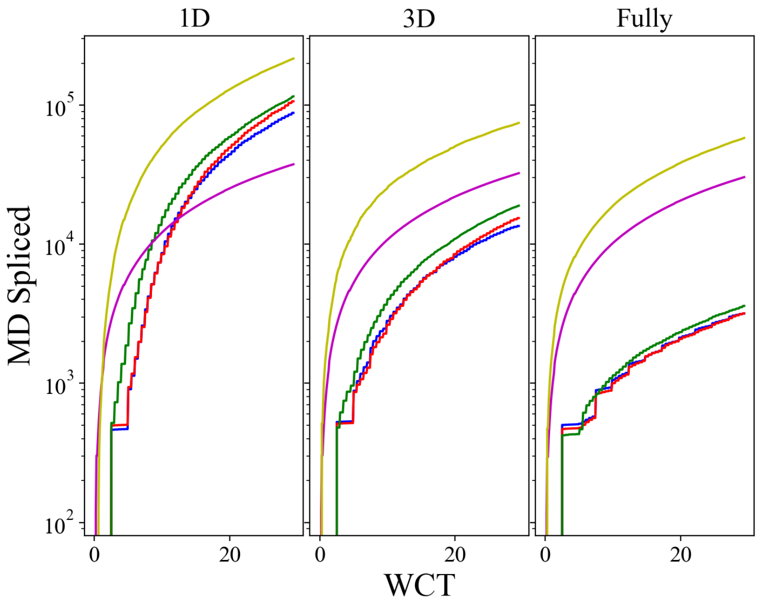


Figure 25: ParSpliceSIM results for the 1D, 3D, and fully-connected toy models showing pseudo-MD spliced as a function of WCT. Each panel displays performance of VE in blue, MaxP in red, MaxP(w_{const}) in green, MaxP($w_{\max}$) in maroon, and MaxP(w^*) in yellow. The results shown represent an average of roughly 500 independent simulations conducted for each method on each model.

toy models, respectively. These ParSpliceSIM results can be better understood by analyzing the task probability distributions that are characteristic of each toy model. Figure 26 shows an example initial probability distribution for each of the toy model systems as was constructed by the MCMaxP procedure described in appendix A. One can see how the task probabilities for the 1D model exhibit a very gradual decay over the first 5,000 tasks down to $p \sim 0.8$. This reflects the limited state connectivity which makes speculation fruitful. The 3D task probabilities exhibit a very steep decay over the first ∼100 tasks (corresponding to an escape from the current state), followed by a more gradual decay over the following 600 tasks (corresponding to an escape from the 6 neighbors of the current state), followed by a more gradual decay out to 5,000 tasks. Overall, task probabilities remain non-negligible out to 5,000 tasks with long tail around $p \sim 0.2$. Considering the fully connected model, one can see a sharp decay in probability corresponding to the first escape from the current state, after which the probability of tasks drops down to $p < 0.1$. One can see how these results in performance generally adhere to the inference made while studying the synthetic distributions, i.e. performance gains are largest when the probability distribution exhibits steep decays and long tails.

Figure 27 shows the evolution of task probability distributions which were sampled throughout a simulation for each of the toy model systems. This occurs for two reasons: 1) The effect of the database; the stored segments which have been generated but not yet consumed by the trajectory play a role in the MCMaxP sampled trajectory and therefore affect which segments are expected to be "needed" by a future trajectory. And 2) the time horizon over which the MCMaxP trajectory is sampled; a greater time horizon corresponds to a higher likelihood that a particular segment will eventually be spliced into the trajectory, thus resulting in a shallow decay of the probability distribution. The dynamic nature of the probability distributions is quite relevant as it pertains to a changing allocation which must be maintained in order to achieve optimal performance. Recognizing this fact, one may consider the difficulties of maintaining the optimal allocation and note how

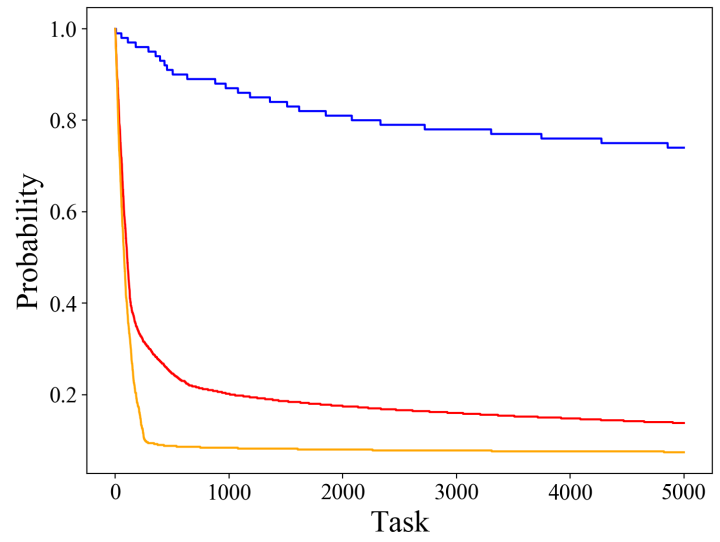


Figure 26: Initial task probability distributions taken from simulations on the 1D (blue), 3D (red), and fully-connected (orange) toy models. These task probability distributions were constructed at the start of the simulation; having no contribution from the then-empty database of stored segments, and are therefore a reflection of the state connectivity.

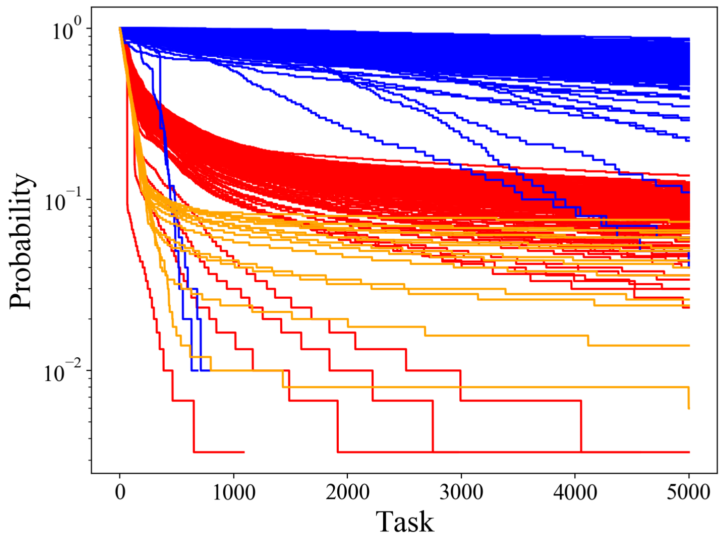


Figure 27: All task probability distributions generated during a single simulation on the 1D (blue), 3D (red), and fully-connected (orange) toy models.

the frequency at which segments are paused and resources are reallocated can have a large impact on performance. Constantly maintaining the optimal allocation involves pausing and reallocating resources whenever any new information is available, which is not always feasible. In practice, the user may choose to pause and update the allocation at some fixed interval, which can be tuned for the user's particular system. In the present study, the allocation was updated whenever a segment was completed. This can be thought of as the most aggressive scenario that will produce the upper bound in performance. In the ParSpliceSIM results above this condition could be relaxed to update whenever a segment contained a transition as the true underlying Markov Chain was being used rather than developing a model from segment data, and therefore segments which did not yield escapes would have no effect on the current MCMaxP-constructed task probability distribution. In a true ParSplice simulation however, each segment would contribute to the development of the statistical Markov Model being produced on-the-fly and thereby (possibly) change the statistics of the MCMaxP sampled trajectories, hence affecting the task probability distribution.

The question of when it is appropriate and necessary to pause and reallocate resources is not easy to quantify. In general, the user would want to do so whenever the task probability distribution substantially changes. In the example of ParSplice, this would certainly occur whenever the current state of trajectory changes as all tasks and probabilities are generated from the MCMaxP procedure and are thus conditional on starting in a particular state. The distribution could also change without the trajectory changing state, however, when unused segments which contain transitions are stored in the database and can contribute to the MCMaxP sampled trajectories. For this reason the ParSpliceSIM results were aggressively updated whenever a segment yielding a transition was returned to the database. The question becomes even more fuzzy for a real ParSplice simulation which develops it's statistical model on the fly. Any segment which changes the model in a substantial way will likely change the MCMaxP sampled trajectories and thus affect the sampled probability distribution. It is not easy to systematically catch changes of this type. A single segment will likely have a negligible effect on the model, but cannot be ignored outright as enough of these negligible changes can account for a significant effect. In addition, a segment which drastically changes the model but does so far from the current state of the trajectory will have little effect on the sampled task probability distribution and therefore does not warrant pausing and reallocating resources. In practice, it is pragmatic to periodically pause and update the allocation at some fixed interval, which is chosen so as to limit scheduling and pausing/restarting overhead.

0.10 Conclusion

The advent of exascale computing platforms will be accompanied by a need for specially designed software and algorithms that are capable of utilizing the large availability of resources simultaneously. As maintaining strong-scalability on such platforms will be quite difficult, the use of speculative task-based paradigms are promising; enabling higher concurrency and improved scaling. In this work, we derived the optimal allocation of resources for task execution in this speculative setting. The utility of this approach was then assessed on assortment of synthetic task probability distributions, comparing the expected throughput of our derived optimal allocation of resources to more naive allocation policies. While a uniform allocation of resources can often be found to produce a nearly optimal expected throughput, it was shown that determining the particular value for the constant allocation size is in practice just a difficult as computing and employing the optimal allocation.

A dynamic setting was then considered where task probabilities were influenced by some underlying variable (state, context, time, etc.) and were therefore changing throughout the run-time of the application. This setting was explored by examining the effect of our derived methods applied to a specific scientific application, ParSplice, which operates in this domain. In order to implement our methods, we first had to design a new application-specific technique for accessing the speculative probability that potential tasks would be useful. This technique not only allowed for our derived methods to be implemented, but was also shown increase the performance of the scientific application. The potential gains in performance resulting from our derived methods were assessed through the use of a simulator. While the boost achieved varied with physical system (ranging from 2.5x to 20x), it was found to be greatest when the system of study was most complex; resulting in lower speculative task probabilities and a greater ability to leverage the trade-off between throughput and time. By considering the speculative task probabilities, the optimal balance could be struck to produce the maximum rate of expected throughput. This novel optimization scheme stands to improve performance of speculative task-based applications, particularly when run at large computational scales.

Appendix: Maximum-Probability (MaxP)

In the following, the probability that a candidate task (e.g, the generation of a segment starting in a given state) will be consumed as part of the calculation is derived in the context of discrete time Markov Chains, which is the natural setting for a trajectory composed of segments generated following the ParSplice prescription. In other words, the problem at hand is to compute the probability that a trajectory of a given length, sampled from this Markov Chain, would contain the generated segment. As will be shown, these probabilities can be evaluated analytically from the Markov jump process or approximated through a Monte Carlo sampling procedure. To clearly distinguish from any variables defined in the main text, we have chosen to express this derivation using a double-struck font.

This derivation utilizes a Markov model $\mathbb{M}$; a stochastic matrix whose elements $\mathbb{p}_{ij}$ represent the probability of moving from state i to state j, thereby governing the discrete Markov process. In the example of our scientific application (ParSplice) these are the probabilities that a segment starting in state i will end in state j. Note that these probabilities encode the potential outcome of the task, not the potential task's usefulness, and are therefore distinct from task probabilities p_i defined in the main text. The work detailed herein describes our method for extracting these latter probabilities p_i from the former probabilities $\mathbb{p}_{ij}$.

We define $\mathbb{f}_{ij}^{(n)}$ as the probability that the first passage from state i to state j takes exactly n steps. This can be written recursively as

$$\begin{aligned}
\mathbb{f}_{ij}^{(1)} &= \mathbb{p}_{ij}^{(1)} = \mathbb{p}_{ij} \\
\mathbb{f}_{ij}^{(2)} &= \sum_{k!=j} \mathbb{p}_{ik}\mathbb{f}_{kj}^{(1)} \\
\mathbb{f}_{ij}^{(n)} &= \sum_{k!=j} \mathbb{p}_{ik}\mathbb{f}_{kj}^{(n-1)}
\end{aligned}$$

We then also define $\mathbb{f}_{ii}^{(n)}$ to be the probability that the first return to state i upon leaving state i takes exactly n steps, which can similarly be written recursively as

$$\mathbb{f}_{ii}^{(n)} = [\mathbb{M}^n]_{ii} - \sum_{k=1}^{n-1} \mathbb{f}_{ii}^{(k)}[\mathbb{M}^{n-k}]_{ii}$$

Where $[\mathbb{M}^n]_{ii}$ represents the i,i element of the Markov Model $\mathbb{M}$ raised to the power n. Then, summing over the index n allows one to write the probability of a return to state i sometime in the next $\mathbb{N}$ steps:

$$\mathbb{F}_{ii}^{(n)} = \sum_{n}^{\mathbb{N}} \mathbb{f}_{ii}^{(n)}$$

Lastly, let $\mathbb{v}_j$ be the number of visits to state j, and $\mathbb{P}_i(\mathbb{X})$ denote the probability of $\mathbb{X}$ conditional on starting in state i. The probability of making exactly m visits to state j over the next $\mathbb{N}$ steps from the current state i can then be expressed recursively as

$$\mathbb{P}_i(\mathbb{v}_j = 1|\mathbb{N}) = \sum_{k=1}^{\mathbb{N}} \mathbb{f}_{ij}^{(k)}[1 - \mathbb{F}_{jj}^{\mathbb{N}-k}]$$

$$\mathbb{P}_i(\mathbb{v}_j = 2|\mathbb{N}) = \sum_{k=1}^{\mathbb{N}} \mathbb{f}_{ij}^{(k)} \mathbb{P}_j(\mathbb{v}_j = 1|\mathbb{N} - 1)$$

$$\mathbb{P}_i(\mathbb{v}_j = m|\mathbb{N}) = \sum_{k=1}^{\mathbb{N}} \mathbb{f}_{ij}^{(k)} \mathbb{P}_j(\mathbb{v}_j = m - 1|\mathbb{N} - m)$$

Summing over the index m and subtracting from 1 yields the probability of having more than S visits to a state over a horizon of $\mathbb{N}$ steps:

$$\mathbb{P}_i(\mathbb{v}_j > S|\mathbb{N}) = 1 - \sum_{m=1}^{S} \mathbb{P}_i(\mathbb{v}_j = m|\mathbb{N})$$

Therefore, given a current state of the trajectory i and the number of pending/unconsumed segments in state j, this prescription provides a means of extracting the probability that the next segment generated in state j will be consumed into the trajectory over the finite time horizon $\mathbb{N}$. Denoting the number of pending/unconsumed segments in state k as S_k allows the the probability of each potential task to be written as

$$p_k = \mathbb{P}_i(\mathbb{v}_k > S_k|\mathbb{N}), \forall k$$

Having this derivation in mind, we propose a new ParSplice scheduling scheme referred to as MaxP (maximum probability). In MaxP, the probability that each task will be spliced into the trajectory over a given finite time horizon is calculated. While computing these probabilities can be done analytically, it is often far more practical and efficient to do so via a Monte Carlo procedure, especially when the number of states is large.

This can be done in a similar fashion to VE where an ensemble of future state-to-state trajectories are sampled, accounting for the pending/unconsumed segments in the same way, but, instead of stopping when running out of segments, each trajectory continues until the preset time horizon, keeping track of how many segments would have to be generated in each state to reach said horizon. This ensemble is then used to calculate the probability that particular segments built in particular states are to be used by the state-to-state trajectory over the time horizon. ParSplice can then assign segments to be generated in decreasing order of probability, thus generating the segments which have the “maximum probability” of being spliced. It can actually be shown[81] that the MaxP allocation scheme formally minimizes the expected number of database “misses”, i.e., the number of times splicing has to be interrupted because a segment that is required to move forward is not found in the database.

One may note that MaxP is substantially more expensive than VE for assigning an initial state to a single segment. While this is true, the ensemble average required by MaxP can be used to make state-assignments for a large number of segments all at once. This is compared to VE which can make one state-assignment for each virtual-trajectory. Furthermore, the VE process for assigning states to several segments must each be done in serial, meanwhile the ensemble trajectories used by MaxP can be generated in parallel. These differences can become quite significant as simulations scale to larger and larger machines, employing a greater number of MD instances.

Decision theoretic approach to optimal scheduling

What is "optimal" scheduling? Considering that the measure of performance in a ParSplice simulation is the boost obtained through splicing a long-time trajectory, then the optimal scheduling *policy* should be to schedule the segments which are most likely to be spliced into the trajectory. As was previously stated: segments which are generated but not spliced do not extend the long-time trajectory and therefore do not provide any boost. While this statement is true, the issue of optimal scheduling is a bit more subtle. To carefully investigate the subtleties herein we will employ the framework of normative decision theory for which actions are analyzed in terms of *utility* and an *optimal policy* can be rigorously defined.

Provided the work detailed in the previous chapter one may propose the policy which schedules segments according to the maximum probability of being consumed into the state-to-state trajectory within the time horizon of the simulation (MaxP). The subtly here arises from the fact that the statistical Markov model used to derive the probabilities is imperfect as it is developed on-the-fly. If the Markov model were perfect this would certainly be the optimal scheduling policy. However, since the model is constructed from observing segments it will be extremely inaccurate initially and will continue to improve throughout the runtime of the simulation as segments are generated and observed. As the model becomes more accurate, and thus the derived probabilities become reliable, the MaxP policy approaches optimal. Herein lies the subtly; it will be optimal to schedule those segments which have the highest probability of being spliced, however, these probabilities are derived from a statistical model which improves as samples (segments) are generated in particular states. Improving the statistical model involves generating segments in the states with the highest uncertainty, i.e fewest number of samples.

This question of whether to utilize resources to exploit a given model, or to use those resources to explore the problem domain and improve the model, is a longstanding question in decision theory and is known as the exploration-exploitation dilemma. In probability theory this type of problem is commonly referred to as a multi-armed bandit [82]. The name stems from a conceptual view of the problem involving a row of slot machines. Due to the nature of slot machines (being that typically money is inserted and an arm is pulled, yielding a payout with an expected value that is less than the money inserted) they are sometimes called "one-armed bandits". This expression was then generalized to a row of slot machines and used describe the following conceptual problem. Imagine having some finite allotment of resources which must be spent playing a row of slot machines, each of which operates under some unknown probability distribution governing it's payout. The only way to gather information regarding a machines probability distribution is by playing it and observing the outcome. Therefore, the question arises: How does one optimally play in order to maximize their overall expected payout?

A strategy might be to purely exploit the machine which is believed to provided the greatest expected payout. However, the expected payout for each particular machine is derived from the observations made playing that machine. Is it really more wise to choose a machine with a slightly higher expected payout but which has been played very few times, rather than a machine with a slightly lower expected payout but which you have played very many times? How does this compare to selecting a machine with a very low expected payout that has only been played once? Or what about choosing a machine which has not been played at all? Clearly the variance of the estimated payout should play a role in the decision of whether or not to play a particular machine. Depending on the time horizon and corresponding level of risk-aversion, it may be advantageous to select machines with higher variance. In finance, long positions in securities are commonly valued using the Capital Asset Pricing Model (CAPM) for which the value is increased by a term proportional to beta; the normalized co-variance between a given asset and a standard benchmark like the S&P500 [83]. In multi-armed bandit problems, a long time horizon corresponds to having a large availability of resources. In this case, it might be wise to explore the different machines in order to find the one which provides the highest expected payout. Once found, all remaining resources can then be used to exploit that machine. If resources are quite limited however, it might be better to exploit the machine which is currently believed to be best; because finding the best machine without the resources to then exploit it is of no good. In the end, it is the overall payout obtained which stands as the metric of performance. Hence, the balance between exploration (utilizing resources to gather reliable statistics) and exploitation (utilizing resources to obtain the highest expected value – based on the gathered statistics) can be quite delicate.

The objective of normative decision theory is derive the optimal policy corresponding to the maximum expected utility obtained throughout a decision making process. It is often helpful to introduce the concept of *regret* under a given policy, which is defined to be the difference between the expected cumulative utility of the optimal policy and that of the given policy. Therefore, the optimal policy is identically defined to be the policy of zero regret. It is important to note, however, that both the optimal policy and the notion of regret correspond to the expected utility of a policy rather than the utility obtained. The concept of regret is thereby congruent with the layman definition, i.e after having employed the optimal policy the agent, despite the payout obtained, would have no regret and given the opportunity to do it all again would not do anything differently. In this way the optimal policy is analogous to the game theory concept of a Nash equilibrium [84], for which nothing can be gained by invoking a different strategy.

Decision theory is used to analyze choice under uncertainty, focusing on the expectation rather than the sample value. This distinction is made clear by considering the "all-in" policy of choosing an initial random machine and spending all of your resources playing it, never switching to explore other machines. This policy will provide the maximum possible payout IF you happen to randomly select the best machine initially. However, it is equally likely that the worst machine will be randomly selected and this policy will provide the minimal possible payout. Given that this "all-in" policy could provide both the maximum possible payout and the minimum possible payout it is more meaningful to evaluate the policy in terms of the expected payout. In fact, due to the stochastic nature of the problem, two agents acting under identical policies will most likely obtain different payouts. It is for this reason that decision theory considers choices in terms of expectation rather than sample outcome.

There have been a plethora of methods devised for handling multi-armed bandit problems [85], involving varying levels of complexity. The simplest strategy invokes a policy called ϵ-greedy, where at each decision-step the agent chooses between the action corresponding the maximum expected payout (exploit) with probability $1 - \epsilon$, or a random action (explore) with probability ϵ. Albeit remarkably simple, this strategy performs surprisingly well in practice. There are other more complex methods which focus the exploration according to the variance of an action's expected payout and can further improve performance [86, 87]. While these methods can be quite good they are still found to be sub-optimal. This is because they fall into a class of policies which aim to satisfy other metrics of performance, like minimizing the maximum regret or maximizing the likelihood of obtaining the greatest payout. Meanwhile, the optimal policy is that which maximizes the overall expected payout. These sub-optimal policies typically involve a much smaller/larger variance along with their slightly smaller expected values. The fact that these sub-optimal policies can often outperform the optimal policy (by yielding a greater payout) might call into question whether the expected value is the correct objective function to be optimizing. It will be so long as the metric of performance is obtaining the largest cumulative payout (i.e, not outperforming some other policy). The multi-armed bandit problem would be radically changed if you were competing against another agent, and only the agent with the greatest payout would receive their bounty. This type of problem however would evoke many elements of non-cooperative game theory and is well outside the scope of this thesis. The problem we are analyzing is a single player game involving one agent whose goal is to maximize expected utility. That being said, there has been long-standing skepticism regarding the focus on expected value, including paradoxical questions which seem to arise [88]. The resolution to these queries often come in the form of time-valued discounting.

When a problem involves a series of decisions, e.g multi-armed bandit problems, it becomes necessary to discount future utility with each decision made. When done correctly, the optimal policy for a multi-armed bandit problem can be rigorously defined and expressed in terms of dynamic allocation indices [89, 90]. The general concept is that each action will produce some direct utility (in the form of payout) as well as some information utility describing the particular machine (because the payout is a sample from that machines probability distribution). While the direct utility will be of constant value throughout the decision process, the information utility will be of greater value earlier in the process as there are more resources available to exploit it. Consider for example the information utility produced on the last play of the bandit process. While the resulting payout will tell the agent more about that particular machine, there are no more resources available and thus that information is worthless to the agent. Therefore, the information utility obtained from a given play must be discounted based on the horizon of the decision process.

Equating the direct utility and the information utility is oftentimes difficult as it is not always clear how the two interrelate. A technique for comparing the two is to describe the information utility as it relates to the potential increase of direct utility cumulatively obtained. This is done by estimating the Expected Value of Sample Information (EVSI), that is, the expected increase in utility that could obtained from gaining access to an additional sample observation before making a decision. Said differently, it is the expected value that the sample information will provide as it pertains to improving the cumulative expected utility. The EVSI is thereby defined as the difference in the expected utility of decision D and that of the decision given sample information SI.

$$EVSI = EU[D|SI] - EU[D]$$

Since EVSI attempts to estimate what this improvement would be before seeing actual sample data it is referred to as preposterior analysis. Calculating the expectation value involves integrating over all possible outcomes for the sample information. It is often impractical, if not impossible, to integrate over the space of possible observations analytically. The computation therefore typically involves a Monte Carlo approach where samples are generated and used to extract an EVSI.

Having this powerful technique in mind, we now turn back to the application of interest (ParSplice) in hopes to employ the EVSI method for optimal scheduling. When a ParSplice segment is generated in a particular state it will produce a binary value of direct utility (based on whether or not the segment generated is consumed into the long-time trajectory) as well as some amount information utility about the particular state (via the sample segment information that is incorporated into the Markov Model). The expected value of the direct utility of a segment can be computed as was done in the MaxP formalism described in the previous chapter. It can be shown that the EVSI from generating a segment in a particular state can be expressed as the increased *residence time* expected in an absorbing Markov Model. Therefore, an absorbing Markov Model is simultaneously constructed on-the-fly along with the MLE model. This absorbing model incorporates uncertainty into an absorbing state which represents the trajectory escaping from the current model. Upon discovering each new state, a Bayesian prior absorption probability of 1 is assigned to the state, reflecting the complete uncertainty of segments generated in that new state. As segments are generated in this new state and observed the Bayesian prior becomes diluted and in the limit of vanishing uncertainty the two models converge.

The question then becomes: How would sample information from a segment change the absorbing model and hence increase the residence time? It is clear that obtaining the segment information would decrease uncertainty within the state and thus decrease the probability of absorbing from that state, hence increasing the residence time. However, given that the probabilities are stochastic, all transition probabilities (including absorbing) must sum to 1. This implies that a decrease in absorption must result in an increase somewhere else. Where this corresponding differential in probability should go is unclear. Intuitively, the observation a sample segment would result in a transfer of some probability from the absorbing state to the state where the segment ended. The preposterior analysis would require averaging over all the possible outcomes of the sample segment, however, this is not possible when all of the possible outcomes are not known. Strictly averaging over the known possible outcomes is a poor approximation as the greatest increase in information utility comes from discovering a new state. This begs the question: How does one perform a model update to include unobserved sample segment information in a way that is accurate and effective.

In our quest for an EVSI optimal ParSplice scheduling routine we have implemented many approximations and techniques for encoding the sample information into the Markov Model. Each of which relying on some heuristic or rule that requires a lot of hand-waving. In addition this dilemma, we have faced several other challenges. During a ParSplice run, if Markov Model is inaccurate due to poor statistics it might suggest an artificially high residence time. When this poor model is improved by obtaining segment information, the resulting residence time could decrease. This would imply a negative change in residence time and therefore a negative EVSI. Given this example, the method would actively avoid scheduling in those states with the greatest uncertainty as it tries to "ride the high" of a poor model.

Yet another obstacle that stood in the way of developing an EVSI optimal scheduling routine was a means of rigorously accounting for stored segments in the database. Our analytical solution involved diagonalizing the transition matrix to compute the difference in residence times, meanwhile the expected value of direct utility was greatly dependent on the stored segments in the database. We attempted to account for them in a generic way, without the consideration of where each segment ended, but this proved futile. In some preliminary ParSpliceSIM results it was discovered that the standard VE scheduling routine would regularly outperform our heuristically-optimal EVSI method. Further investigation showed this was due to the way VE scheduling "virtually spliced" the exact segments stored in the database. This small bit of information enabled massive gains over the course of a simulation. While we could account for the exact database segments using a Monte Carlo technique as in MCMaxP, the lack of a provably optimal analytical method seemed to defeat the purpose of the venture.

We then moved to considering other more simple, sub-optimal policies (like ϵ-greedy) that did not have as many intricacies but could still improve performance. Unfortunately, employing a less general method would not serve the complexities of our problem. This is because the problem of scheduling segments to explore and exploit is more complicated than a standard multi-armed bandit problem. The likelihood that a given segment will be spliced into the long-time trajectory is conditional on the current state of the trajectory. Throughout the runtime of a simulation these probabilities will dynamically evolve. Fortunately, this is a type of multi-armed bandit problem referred to as restless bandits. More so, our problem involves a constantly increasing number of states, or lever arms to be pulled. Fortunately, this is also a type of multi-armed bandit problem referred to as arm-acquiring bandits. Even more so, our problem involves a database of segments which evolves throughout the runtime of the simulation, having a large effect on the probabilities. Fortunately, you guessed it, this is also a type of multi-armed bandit problem referred to as contextual bandits. It seemed as though mathematicians had thought of everything. With this classification, we just needed to find a method which satisfied a relentless arm-acquiring contextual multi-armed bandit problem. Unfortunately, our problem contains an additional complexity that not even the mathematicians dreamed up. Our problem is inherently non-local. The information utility produced from generating a segment in a state with high uncertainty is large, however, if that state is in a region of state space that is very far from the current state and is not likely to ever be visited then this information is of little value. Unlike with slot machines, where switching between any two machines is easy, we are only interested in information utility in so far that it contributes to the generation of direct utility further down the line by improving the model in regions of state space which will be visited by the trajectory. Improving the model in a region which will never be visited is of zero utility. Further complicating the situation is the fact that determining which regions of state space will be relevant a priori is impossible as it depends on state connectivity which is inherently unknown.

Considering the complications faced in developing an optimal scheduling method that balances the trade-off between exploration and exploitation, this research direction has been put on hold. There are plans in place to invite some collaborating mathematicians from France to LANL for a week long visit, but until then this work will remain shelved. We would like to note that we remain hyper-optimistic about the potential work as it naturally presents as a Markov Decision Process (MDP) and can be posed in the context of Reinforcement Learning (RL) where an agent chooses an action based on the current environment. The action then produces some reward and evolves the state of the system, which is fed back to the agent and used to inform the next decision. The field of RL is a rapidly growing research area as it has many applications from science to industry. Although this direction did not pan out (yet), we are hopeful for the progress that can be made as an optimal RL method for learning a statistical Markov Model from sample data would have far-reaching applications beyond ParSplice.

Potential applications & future work

This final chapter is dedicated to identifying the type of physical applications which stand to benefit most as a result of the work in this thesis. Improvements in scaling will be discussed and used to generically identify ideal candidate systems in terms of their state-space characteristics. Future work involving applications to nuclear fuels will be investigated and discussed as it pertains to potential post doctoral research.

The original parallel method designed by Voter in the late 90's (ParRep) had a restriction on the amount of resources N which could be efficiently utilized. This limitation could be characterized by an inequality relating the cost of the method to the boost obtained: $N\tau_c \ll \langle n_{\text{escape}} \rangle / N$. The constraint imposed by this inequity corresponds to an inherent scaling limitation on the method, requiring proportionally deeper states in order to utilize greater resources (i.e, larger $\langle n_{\text{escape}} \rangle$ required to utilized greater N). As a consequence of this scaling limitation, ParRep would yield disastrous efficiency whenever the trajectory visited shallow states. This implied that systems which evolved through a series of quick transitions, e.g low barriers, could not be sped up, and parallel resources could not be utilized.

This limitation was somewhat relaxed with the implementation of ParSplice where each MD instance now created a discrete 'segment' of trajectory which could eventually be spliced into the long-time trajectory. Recall, those segments which were created but not immediately spliced would be stored in a database where they could later be extracted and consumed into the trajectory. This amortization of segments previously generated enabled greater efficiency to be achieved in shallower states so long as the trajectory revisited the state sufficiently many times (e.g superbasin of states). As a result, ParSplice enabled better scaling which was limited by superbasin depth rather than state depth. However, the general scaling limitation still remained, requiring proportionally deeper superbasins in order to utilize greater resources.

The problem here is that "depth" is relative to the number of MD instances. As we scale to larger computing platforms, the same state (or superbasin of states) will require a smaller fraction of the total resources in order to find an escape. This is problematic as the relates to the low-barrier problem. As resources scale all barriers begin seem low by comparison. In the large N limit, all states and superbasins will only require an ϵ-portion of the total resources in order to find an escape.

Fortunately, due to the distributed nature of the ParSplice workflow, resources do not have to be solely utilized generating a single escape. Instead, resources can be allocated among various states in an attempt to speculate ahead of the trajectory and generate the next several escapes concurrently. In theory, this factor enables strong scaling, conditional on accurate speculation. This is the ground upon which I began my thesis work. The prospect of strong scaling MD simulation with exascale resources would be transformative, enabling scientific discoveries far beyond our current grasp. Doing so, however, involves answering difficult questions, like addressing the low-barrier problem.

As detailed in chapter 1, I began by working to improve the statistical model which is used to speculate on the long-time trajectory's future and distribute resources. Constraining the model to obey detailed balance is an enhancement that improves the performance of simulations where state depth is less than the number of resources, $\langle n_{\text{escape}} \rangle < N$, a case that is common in practice, particularly when N is large. In the limit of exascale resources this will become the predominant case as all states will begin to obey this condition.

In addition to the detailed balance model, the implementation of the warp move was a general enhancement designed to aid the model in the presence of uncertainty and help distribute resources. The warp move has been shown to be most effective on complex systems with high state connectivity. In low-barrier, highly connected systems the warp move tends to act as uncertainty triggered exploration, allowing the local state-space to be surveyed in less time. In the presence of high model uncertainty the warp move enables speculation and increases the likelihood of scheduling ahead of the trajectory. This tends to be especially useful when the underlying state connectivity is far more complex that the current model suggests.

Both the warp move and detailed balance model were developments designed for increasing parallel efficiency to improve ParSplice performance at scale. Due to the relative nature of state depth and resources, these developments also enable better performance in simulations utilizing a smaller allocation to study systems which evolve through shallower states. Ideal candidates for study are physical processes which contain an intermediate regime of shallow states that separate more strongly meta-stable states. Previously these systems were out of reach for ParSplice as the parallel efficiency would tank in the intermediate regime, causing the trajectory to get "stuck" there as the boost within that regime would tend toward unity.

The work detailed in chapter 2 can be separated into two separate developments. The first was the creation of the MaxP procedure for scheduling segments. This method is statistically superior to the VE scheduling method as it schedules according to the maximum probability that a segment will be spliced rather than the probability distribution of the next segment to be needed. This statistical superiority is shown to produce a small but appreciable improvement in parallel efficiency. In addition to improving the efficacy of scheduling MaxP also improves the functionality of scheduling. This was not highlighted in chapter 2 as it was not central to the paper, but is nonetheless an important development to note. While the VE scheduling routine is inherently serial the MaxP method is trivially parallelizable. This fact contributed to the further developments of resource optimization, but is a substantial improvement itself. In practice, the utilization of a massively parallel computing resource would require a 'batch' approximation to VE scheduling where a batch of segments are all assigned to a single state. This is needed because the scheduling of segments one-by-one would create a scheduling bottleneck resulting in idle resources waiting to be assigned to states. However, the batch approximation to VE only distances the limitation of the scheduling bottleneck, it does not eliminate it. This due to the fact that the batch size must be less than the most shallow state in order to not effect the efficiency. The fixed batch size means that scaling resources to large enough N will reproduce the scheduling bottleneck. This is not the case for the MaxP scheduling procedure since all of the work generating the ensemble of virtual trajectories (which can be done in parallel) is done up front, thus scheduling one segment is as computationally expensive as scheduling one million segments. Therefore, the MaxP scheduling procedure contributes generally to simulations by providing a minor improvement in parallel efficiency due to better scheduling decisions, as well as a major improvement in the scalability due to the distributed functionality of the method.

The second of the two developments detailed in chapter 2 is the resource optimization scheme, providing the ability to trad-off throughput for time in order to maximize the rate of expected throughput. This optimization scheme is generally very powerful, and has the potential to provide substantial improvements in parallel efficiency, particularly for difficult systems which are normally considered outside the realm of ParSplice. Systems of this type include those with a very high state connectivity, for which speculation ahead of the trajectory is near futile due to the overwhelming number of potential pathways. Traditionally, ParSplice would attempt to speculate as best it could, and would simply suffer a poor parallel efficiency due to the unpredictable nature. However, with the optimization scheme, ParSplice is able to trade-off throughput for time by producing fewer segments (those which would have normally been used to speculate) and instead allocating more resources to generating the higher probability segments faster.

The rescheduling aspect of the optimization scheme is a development which stands to have a substantial effect in simulations where the trajectory is not likely to revisit states. During a simulation, when the ParSplice trajectory is trapped in a very deep state all resources are allocated to generating segments in that state in order to find an escape. Traditionally, it is only when a segment is completed that the resource(s) assigned to generating that segment are assigned to a new state. This implies that, upon finding an escape from the very deep state, the remaining resources will continue generating their segments in that deep state until those segments are completed, at which point those segments will be stored in the database until they are needed (i.e the trajectory returns to that state). However, in the case that state revisits do not occur, those segments will not be spliced into the trajectory and the parallel work will go unrealized. In this case, it would be more effective to terminate those pending segments once an escape from the deep state was found, freeing up the resources so they could be allocated to generating other (more useful) segments. Therefore, this development is ideally suited for the study of processes which evolve through a unique series of configurations, rarely revisiting the same configuration twice.

While the work of this thesis can be applied generally to enhance the simulations of many systems, we now turn to discussing a particular system of interest as it pertains to future work. The most prominent material commonly used as a fuel within nuclear reactors is Uranium dioxide (UO_2) due to its ability to endure the harsh conditions present within the reactor (e.g high temperature gradients, irradiation damage, and chemical changes). As the fuel is burned, the U atoms fission into lighter elements. The fission products, particularly the noble gas atoms among them, cause concern as they relate to thermal conductivity[91, 92], fuel swelling[93, 94], and changes in the micro-structure [95]. A more thorough understanding of the fission gas diffusion within the fuel is desired as it has been shown to have tremendous impact on these performance metrics[96, 97]. Therefore, in order to push for higher fuel burn up, as well as maintain the safety of regular burn up, a fundamental understanding of the physics must be obtained. Having this knowledge would enable scientists and engineers to make informed decisions regarding the safety and operation of nuclear reactors.

Identified by Turnbull et al. [98], there are three regimes of gas diffusion that present within UO_2: 1) the high temperature regime (>1500 K) of intrinsic diffusion, 2) the intermediate temperature regime of radiation enhanced diffusion and 3) the low temperature regime (<1000 K) of irradiation induced athermal contribution (See fig. 28). It is the radiation enhanced diffusion that is of greatest interest as it is currently not well understood. It seems to be very complicated due to the microscopic effects of irradiation damage. The high energy of fission products evolving through collision cascades and thermal spikes give way to an abundance of Frenkel pairs. It is thought that the atomic mixing, which results from these damage induced Frenkel pairs, is the dominant mechanism underlying gas mobility.

Understanding the mechanisms which govern fission gas diffusion within nuclear fuels is a meaningful pursuit, and one that many lead to discoveries of significant consequence. The ideal tool for learning about the physical phenomena is of course atomistic simulation. However, many complexes that contain fission gasses move very slowly, presenting a timescale problem for direct MD simulation. Motivated by this fact, we believe ParSplice will be ideally suited for this materials application and could provide greater insight into the nature of these fuels. The implementation of ParSplice will not be trivial however as this system is quite complicated. The fact that the oxygen sub-lattice within UO_2 evolves much faster than the U sub-system will make simulations very challenging. In addition, the local atomic mixing will create an ever changing topology making it unlikely to capitalize on state revisits. This system will likely have the type of highly complicated state space that was traditionally very difficult for ParSplice. We expect the warp move and the

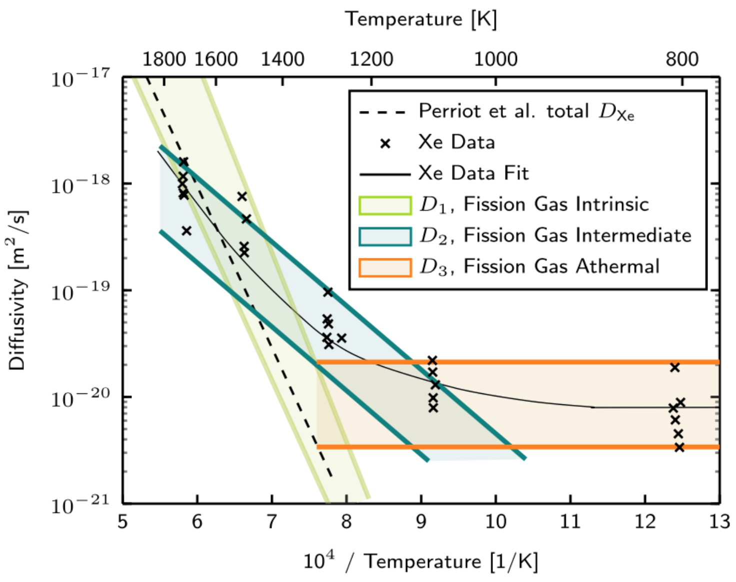


Figure 28: This figure was taken from [3] and is shown to illustrate the three separate regimes of gas diffusion

detail balance model to play substantial role in improving performance, although it is unclear by exactly how much. If the atomic mixing is too dominant it will create a situation in which speculation is futile and revisits rarely ever occur. In this case, it might be beneficial (and perhaps necessary) to employ the optimization methods of chapter 3 in order to utilize resources effectively.

The ideal place to begin this work is by first studying the diffusion of vacancy clusters within the fuel (in the absence of rare gas atoms). Using the specialized ‘canonical’ setting in ParSplice, which recognizes states to be identical under a renumbering of the atoms, should minimize the effect of the atomic mixing and enable high parallel efficiency. These simulations can then be used to draw relations between cluster size and mobility. After this baseline study we can introduce single fission gas defects and study the diffusion, working our way up in complexity. Our studies will begin by examining the diffusion of Xe as it is the most common type of defect. Once these simulations are underway it should be a smooth transition to studying other types of gasses (Kr, etc). Conducting studies of varying complexes, cluster sizes, temperatures, etc should provide some much needed insight, and could help advance the field of nuclear fuels.

Bibliography

[1] Danny Perez, Ekin D Cubuk, Amos Waterland, Efthimios Kaxiras, and Arthur F Voter. Long-time dynamics through parallel trajectory splicing. *Journal of chemical theory and computation*, 12(1):18–28, 2016.

[2] *LAMMPS Website*. `https://lammps.sandia.gov`.

[3] Christopher Matthews, Romain Perriot, WMD Cooper, Christopher R Stanek, and David A Andersson. Cluster dynamics simulation of xenon diffusion during irradiation in uo2. *Journal of Nuclear Materials*, 540:152326, 2020.

[4] Keith J Laidler and M Christine King. Development of transition-state theory. *The Journal of physical chemistry*, 87(15):2657–2664, 1983.

[5] Graeme Henkelman, Blas P Uberuaga, and Hannes Jónsson. A climbing image nudged elastic band method for finding saddle points and minimum energy paths. *The Journal of chemical physics*, 113(22):9901–9904, 2000.

[6] Graeme Henkelman and Hannes Jónsson. A dimer method for finding saddle points on high dimensional potential surfaces using only first derivatives. *The Journal of chemical physics*, 111(15):7010–7022, 1999.

[7] Arthur F Voter. Introduction to the kinetic monte carlo method. In *Radiation effects in solids*, pages 1–23. Springer, 2007.

[8] Graeme Henkelman and Hannes Jónsson. Long time scale kinetic monte carlo simulations without lattice approximation and predefined event table. *The Journal of Chemical Physics*, 115(21):9657–9666, 2001.

[9] Lijun Xu and Graeme Henkelman. Adaptive kinetic monte carlo for first-principles accelerated dynamics. *The Journal of chemical physics*, 129(11):114104, 2008.

[10] Thomas D Swinburne and Danny Perez. Self-optimized construction of transition rate matrices from accelerated atomistic simulations with bayesian uncertainty quantification. *Physical Review Materials*, 2(5):053802, 2018.

[11] Arthur F Voter. Hyperdynamics: Accelerated molecular dynamics of infrequent events. *Physical Review Letters*, 78(20):3908, 1997.

[12] Soo Young Kim, Danny Perez, and Arthur F Voter. Local hyperdynamics. *The Journal of chemical physics*, 139(14):144110, 2013.

[13] Steven J Plimpton, Danny Perez, and Arthur F Voter. Parallel algorithms for hyperdynamics and local hyperdynamics. *The Journal of Chemical Physics*, 153(5):054116, 2020.

[14] Mads R So/rensen and Arthur F Voter. Temperature-accelerated dynamics for simulation of infrequent events. *The Journal of Chemical Physics*, 112(21):9599–9606, 2000.

[15] Richard J Zamora, Arthur F Voter, Danny Perez, Nandakishore Santhi, Susan M Mniszewski, Sunil Thulasidasan, and Stephan J Eidenbenz. Discrete event performance prediction of speculatively parallel temperature-accelerated dynamics. *Simulation*, 92(12):1065–1086, 2016.

[16] Susan M Mniszewski, Christoph Junghans, Arthur F Voter, Danny Perez, and Stephan J Eidenbenz. Tadsim: Discrete event-based performance prediction for temperature-accelerated dynamics. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 25(3):1–26, 2015.

[17] Arthur F Voter. Parallel replica method for dynamics of infrequent events. *Physical Review B*, 57(22):R13985, 1998.

[18] Claude Le Bris, Tony Lelievre, Mitchell Luskin, and Danny Perez. A mathematical formalization of the parallel replica dynamics. *Monte Carlo Methods and Applications*, 18(2):119–146, 2012.

[19] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117(1):1–19, 1995.

[20] R. J. Zamora, Danny Perez, E. Martinez, Blas Pedro Uberuaga, and Arthur F. Voter. *Accelerated Molecular Dynamics Methods in a Massively Parallel World*, pages 745–772. Springer International Publishing, Cham, 2020.

[21] Fedwa El-Mellouhi, Normand Mousseau, and Laurent J Lewis. Kinetic activation-relaxation technique: An off-lattice self-learning kinetic monte carlo algorithm. *Physical Review B*, 78(15):153202, 2008.

[22] Danny Perez, Blas P Uberuaga, and Arthur F Voter. The parallel replica dynamics method–coming of age. *Computational Materials Science*, 100:90–103, 2015.

[23] Liang Zhang, Samuel T Chill, and Graeme Henkelman. Distributed replica dynamics. *The Journal of chemical physics*, 143(17):174112, 2015.

[24] *ParSplice Code*. `http://gitlab.com/exaalt/parsplice`.

[25] Francis Alexander, Ann Almgren, John Bell, Amitava Bhattacharjee, Jacqueline Chen, Phil Colella, David Daniel, Jack DeSlippe, Lori Diachin, Erik Draeger, et al. Exascale applications: skin in the game. *Philosophical Transactions of the Royal Society A*, 378(2166):20190056, 2020.

[26] Danny Perez, Rao Huang, and Arthur F Voter. Long-time molecular dynamics simulations on massively parallel platforms: A comparison of parallel replica dynamics and parallel trajectory splicing. *Journal of Materials Research*, 33(7):813–822, 2018.

[27] MA Novotny. Monte carlo algorithms with absorbing markov chains: Fast local algorithms for slow dynamics. *Physical review letters*, 74(1):1, 1995.

[28] Brian Puchala, Michael L Falk, and Krishna Garikipati. An energy basin finding algorithm for kinetic monte carlo acceleration. *The Journal of chemical physics*, 132(13):134104, 2010.

[29] Abhijit Chatterjee and Arthur F Voter. Accurate acceleration of kinetic monte carlo simulations through the modification of rate constants. *The Journal of chemical physics*, 132(19):194101, 2010.

[30] Kristen A Fichthorn and Yangzheng Lin. A local superbasin kinetic monte carlo method. *The Journal of chemical physics*, 138(16):164104, 2013.

[31] David Aristoff. Generalizing parallel replica dynamics: Trajectory fragments, asynchronous computing, and pdmps. *SIAM/ASA Journal on Uncertainty Quantification*, 7(2):685–719, 2019.

[32] George H Vineyard. Frequency factors and isotope effects in solid state rate processes. *Journal of Physics and Chemistry of Solids*, 3(1-2):121–127, 1957.

[33] David Aristoff. Private communication.

[34] Benjamin Trendelkamp-Schroer, Hao Wu, Fabian Paul, and Frank Noé. Estimation and uncertainty of reversible markov models. *The Journal of chemical physics*, 143(17):11B601_1, 2015.

[35] Samuel T Chill and Graeme Henkelman. Molecular dynamics saddle search adaptive kinetic monte carlo. *The Journal of chemical physics*, 140(21):214110, 2014.

[36] David Aristoff, Samuel Chill, and Gideon Simpson. Analysis of estimators for adaptive kinetic monte carlo. *Communications in Applied Mathematics and Computational Science*, 11(2):171–186, 2016.

[37] Danny Perez, Blas P Uberuaga, Yunsic Shim, Jacques G Amar, and Arthur F Voter. Accelerated molecular dynamics methods: introduction and recent developments. *Annual Reports in computational chemistry*, 5:79–98, 2009.

[38] N Juslin and BD Wirth. Interatomic potentials for simulation of he bubble formation in w. *Journal of nuclear materials*, 432(1-3):61–66, 2013.

[39] Steven J Zinkle and Lance Lewis Snead. Designing radiation resistance in materials for fusion energy. *Annual Review of Materials Research*, 44:241–267, 2014.

[40] LK Mansur. Theory and experimental background on dimensional changes in irradiated alloys. *Journal of Nuclear Materials*, 216:97–123, 1994.

[41] SJ Zinkle. 1.03-radiation-induced effects on microstructure. *Comprehensive nuclear materials*, 1:65–98, 2012.

[42] Danny Perez, Luis Sandoval, Sophie Blondel, Brian D Wirth, Blas P Uberuaga, and Arthur F Voter. The mobility of small vacancy/helium complexes in tungsten and its impact on retention in fusion-relevant conditions. *Scientific reports*, 7(1):1–9, 2017.

[43] Michael P. Robson, Ronak Buch, and Laxmikant V. Kale. Runtime coordinated heterogeneous tasks in charm++. In *Proceedings of the Second Internationsl Workshop on Extreme Scale Programming Models and Middleware*, ESPM2, pages 40–43, Piscataway, NJ, USA, 2016. IEEE Press.

[44] Michael Bauer, Sean Treichler, Elliott Slaughter, and Alex Aiken. Legion: Expressing locality and independence with logical regions. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE, 2012.

[45] Mario Di Renzo, Lin Fu, and Javier Urzay. Htr solver: An open-source exascale-oriented task-based multi-gpu high-order code for hypersonic aerothermodynamics. *Computer Physics Communications*, page 107262, 2020.

[46] James C. Phillips, David J. Hardy, Julio D. C. Maia, John E. Stone, Joao V. Ribeiro, Rafael C. Bernardi, Ronak Buch, Giacomo Fiorin, Jerome Henin, Wei Jiang, Ryan McGreevy, Marcelo C. R. Melo, Brian K. Radak, Robert D. Skeel, Abhishek Singharoy, Yi Wang, Benoit Roux, Aleksei Aksimentiev, Zaida Luthey-Schulten, Laxmikant V. Kale, Klaus Schulten, Christophe Chipot, and Emad Tajkhorshid. Scalable molecular dynamics on cpu and gpu architectures with namd. *The Journal of Chemical Physics*, 153(4):044130, 2020.

[47] Hilario Torres, Manolis Papadakis, and Lluís Jofre Cruanyes. Soleil-x: turbulence, particles, and radiation in the regent programming language. In *SC'19: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–4, 2019.

[48] Nikhil Jain, Eric Bohm, Eric Mikida, Subhasish Mandal, Minjung Kim, Prateek Jindal, Qi Li, Sohrab Ismail-Beigi, Glenn Martyna, and Laxmikant Kale. Openatom: Scalable ab-initio molecular dynamics with diverse capabilities. In *International Supercomputing Conference*, ISC HPC '16 (to appear), 2016.

[49] Paraskevas Yiapanis, Demian Rosas-Ham, Gavin Brown, and Mikel Luján. Optimizing software runtime systems for speculative parallelization. 9(4), 2013.

[50] Paraskevas Yiapanis, Gavin Brown, and Mikel Luján. Compiler-driven software speculation for thread-level parallelism. *ACM Trans. Program. Lang. Syst.*, 38(2), December 2015.

[51] Openmp application programming interface. `https://www.openmp.org/specifications/`.

[52] Bradford L Chamberlain, David Callahan, and Hans P Zima. Parallel programmability and the chapel language. *The International Journal of High Performance Computing Applications*, 21(3):291–312, 2007.

[53] Laxmikant V Kale and Sanjeev Krishnan. Charm++ a portable concurrent object oriented system based on c++. In *Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*, pages 91–108, 1993.

[54] J Greggory Steffan, Christopher B Colohan, Antonia Zhai, and Todd C Mowry. A scalable approach to thread-level speculation. *ACM SIGARCH Computer Architecture News*, 28(2):1–12, 2000.

[55] Jayaram Bobba, Kevin E Moore, Haris Volos, Luke Yen, Mark D Hill, Michael M Swift, and David A Wood. Performance pathologies in hardware transactional memory. *ACM SIGARCH Computer Architecture News*, 35(2):81–91, 2007.

[56] Mark C Jeffrey, Suvinay Subramanian, Maleen Abeydeera, Joel Emer, and Daniel Sanchez. Data-centric execution of speculative parallel programs. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13. IEEE, 2016.

[57] F Almeida, F Garcia, J Roda, D Morales, and C Rodríguez. A comparative study of two distributed systems: Pvm and transputers. *Transputers Applications and Systems 95*, pages 244–258, 1995.

[58] Rumen Andonov, Frédéric Raimbault, and Patrice Quinton. Dynamic programming parallel implementations for the knapsack problem. 1993.

[59] D Morales, J Roda, Francisco Almeida, Casiano Rodríguez, and F Garcia. Integral knapsack problems: Parallel algorithms and their implementations on distributed systems. In *Proceedings of the 9th International Conference on Supercomputing*, pages 218–226, 1995.

[60] Efficient Parallel Algorithms. A. gibbons/w, 1988.

[61] Ramakrishnan Rajamony, L Baba Arimilli, and K Gildea. Percs: The ibm power7-ih high-performance computing system. *IBM Journal of Research and Development*, 55(3):3–1, 2011.

[62] Greg Faanes, Abdulla Bataineh, Duncan Roweth, Tom Court, Edwin Froese, Bob Alverson, Tim Johnson, Joe Kopnick, Mike Higgins, and James Reinhard. Cray cascade: a scalable hpc system based on a dragonfly network. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–9. IEEE, 2012.

[63] Daniela Rosu, Karsten Schwan, Sudhakar Yalamanchili, and Rakesh Jha. On adaptive resource allocation for complex real-time applications. In *Proceedings Real-Time Systems Symposium*, pages 320–329. IEEE, 1997.

[64] Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, Torsten Hoefler, Sameer Kumar, Ewing Lusk, Rajeev Thakur, and Jesper Larsson Träff. Mpi on millions of cores. *Parallel Processing Letters*, 21(01):45–60, 2011.

[65] Ahmed Shawky Moussa, Sherif AbdElazim Embaby, and Ibrahim Farag. Intelligent real-time scheduling of dynamic processes in mpi. In *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–4. IEEE, 2017.

[66] Xianghua Xu, Yanna Yan, and Jian Wan. Grey prediction control of adaptive resources allocation in virtualized computing system. In *2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*, pages 109–114. IEEE, 2009.

[67] Dorian Minarolli and Bernd Freisleben. Distributed resource allocation to virtual machines via artificial neural networks. In *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 490–499. IEEE, 2014.

[68] Lei Wei, Chuan Heng Foh, Bingsheng He, and Jianfei Cai. Towards efficient resource allocation for heterogeneous workloads in iaas clouds. *IEEE Transactions on Cloud Computing*, 6(1):264–275, 2015.

[69] Hongyi Ma, Liqiang Wang, Byung Chul Tak, Long Wang, and Chunqiang Tang. Auto-tuning performance of mpi parallel programs using resource management in container-based virtual cloud. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 545–552. IEEE, 2016.

[70] Fan-Hsun Tseng, Xiaofei Wang, Li-Der Chou, Han-Chieh Chao, and Victor CM Leung. Dynamic resource prediction and allocation for cloud data center using the multiobjective genetic algorithm. *IEEE Systems Journal*, 12(2):1688–1699, 2017.

[71] Hanxiong Chen, Xiong Fu, Zhongrui Tang, and Xinxin Zhu. Resource monitoring and prediction in cloud computing environments. In *2015 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence*, pages 288–292. IEEE, 2015.

[72] Domingo Morales, Francisco Almeida, F Garcıa, Jose L Roda, and C Rodrıguez. Design of parallel algorithms for the single resource allocation problem. *European Journal of Operational Research*, 126(1):166–174, 2000.

[73] Salah E Elmaghraby. Resource allocation via dynamic programming in activity networks. *European Journal of Operational Research*, 64(2):199–215, 1993.

[74] Warren B Powell, Joel A Shapiro, and Hugo P Simão. An adaptive dynamic programming algorithm for the heterogeneous resource allocation problem. *Transportation Science*, 36(2):231–249, 2002.

[75] Eric V Denardo. *Dynamic programming: models and applications.* Courier Corporation, 2012.

[76] Daniel González, Francisco Almeida, L Moreno, and C Rodrıguez. Towards the automatic optimal mapping of pipeline algorithms. *Parallel Computing*, 29(2):241–254, 2003.

[77] Janmartin Jahn, Santiago Pagani, Sebastian Kobbe, Jian-Jia Chen, and Jörg Henkel. Runtime resource allocation for software pipelines. *ACM Transactions on Parallel Computing (TOPC)*, 2(1):1–23, 2015.

[78] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics*, 117(1):1–19, 1995.

[79] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485, 1967.

[80] Andrew Garmon and Danny Perez. Exploiting model uncertainty to improve the scalability of long-time simulations using parallel trajectory splicing. *Modelling and Simulation in Materials Science and Engineering*, 28(6):065015, 2020.

[81] Mouad Ramil. Private communication.

[82] Michael N Katehakis and Arthur F Veinott Jr. The multi-armed bandit problem: decomposition and computation. *Mathematics of Operations Research*, 12(2):262–268, 1987.

[83] Eugene F Fama and Kenneth R French. The capital asset pricing model: Theory and evidence. *Journal of economic perspectives*, 18(3):25–46, 2004.

[84] John F Nash Jr. The bargaining problem. *Econometrica: Journal of the econometric society*, pages 155–162, 1950.

[85] Volodymyr Kuleshov and Doina Precup. Algorithms for multi-armed bandit problems. *arXiv preprint arXiv:1402.6028*, 2014.

[86] Odalric-Ambrym Maillard, Rémi Munos, and Gilles Stoltz. A finite-time analysis of multi-armed bandits problems with kullback-leibler divergences. In *Proceedings of the 24th annual Conference On Learning Theory*, pages 497–514, 2011.

[87] Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

[88] Gilbert W Bassett Jr. The st. petersburg paradox and bounded utility. *History of Political Economy*, 19(4):517–523, 1987.

[89] Apostolos N Burnetas and Michael N Katehakis. Optimal adaptive policies for markov decision processes. *Mathematics of Operations Research*, 22(1):222–255, 1997.

[90] John C Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society: Series B (Methodological)*, 41(2):148–164, 1979.

[91] C Ronchi, M Sheindlin, D Staicu, and M Kinoshita. Effect of burn-up on the thermal conductivity of uranium dioxide up to 100.000 mwd t- 1. *Journal of Nuclear Materials*, 327(1):58–76, 2004.

[92] PG Lucuta, IJ Hastings, et al. A pragmatic approach to modelling thermal conductivity of irradiated uo2 fuel: review and recommendations. *Journal of nuclear materials*, 232(2-3):166–180, 1996.

[93] MWD Cooper, SC Middleburgh, and RW Grimes. Swelling due to the partition of soluble fission products between the grey phase and uranium dioxide. *Progress in Nuclear Energy*, 72:33–37, 2014.

[94] SC Middleburgh, RW Grimes, KH Desai, PR Blair, L Hallstadius, K Backman, and P Van Uffelen. Swelling due to fission products and additives dissolved within the uranium dioxide lattice. *Journal of nuclear materials*, 427(1-3):359–363, 2012.

[95] Heiko Kleykamp. The chemical state of the fission products in oxide fuels. *Journal of Nuclear Materials*, 131(2-3):221–246, 1985.

[96] DA Andersson, P Garcia, X-Y Liu, G Pastore, M Tonks, P Millett, B Dorado, DR Gaston, D Andrs, RL Williamson, et al. Atomistic modeling of intrinsic and radiation-enhanced fission gas (xe) diffusion in uo2±x: Implications for nuclear fuel performance modeling. *Journal of Nuclear Materials*, 451(1-3):225–242, 2014.

[97] Charles Richard Arthur Catlow. Fission gas diffusion in uranium dioxide. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 364(1719):473–497, 1978.

[98] JA Turnbull, CA Friskney, JR Findlay, FA Johnson, and AJ Walter. The diffusion coefficients of gaseous and volatile species during the irradiation of uranium dioxide. *Journal of Nuclear Materials*, 107(2-3):168–184, 1982.