2006

Time Parallelization Studies in Bio-Molecular Dynamics Simulations

Lei Ji

THE FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

TIME PARALLELIZATION STUDIES IN BIO-MOLECULAR DYNAMICS SIMULATIONS

By

LEI JI

A thesis submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Master of Science

Degree Awarded:
Summer Semester, 2006

The members of the Committee approve the thesis of Lei Ji defended on May 26, 2006.

Ashok Srinivasan
Professor Directing thesis

Hugh Nymeyer
Committee Member

Namas Chandra
Committee Member

The Office of Graduate Studies has verified and approved the above named committee members.

This thesis is dedicated, to all those amazing people in my life who helped me and wanted me to succeed.

ACKNOWLEDGEMENTS

I would like to acknowledge my gratitude to Prof. Ashok Srinivasan, Prof. Hugh Nymeyer, Prof. Namas Chandra, Jeffrey McDonald, and Yanan Yu. Without them, this is impossible.

— Lei

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Molecular Dynamics (MD) is an important simulation technique with widespread use in computational chemistry, biology, and materials. An important limitation of MD is that the time step size is limited to around a femto (10^{-15}) second. Consequently, a large number of iterations are required to simulate to realistic time spans. This is a major bottleneck in MD, and has been identified as an important challenge in computational biology and nano-materials. While parallelization has been effective in dealing with the computational effort that arises in simulating large physical systems (that is, having a large number of atoms), conventional parallelization is not effective in simulating small or moderate sized physical systems to long time spans. We recently introduced a new approach to parallelization, where data from prior simulations are used to parallelize a new computation along the time domain. We demonstrated its effectiveness in a nano-materials application, where this approach scaled to a larger number of processors than conventional parallelization. In this thesis, we parallelize a computational biology application - the AFM pulling of a protein - using this approach. The significance of this work arises in demonstrating the effectiveness of this technique in a soft-matter application, which is more challenging than the hard-matter applications considered earlier.

CHAPTER 1

INTRODUCTION

1.1 Importance of Molecular Dynamics

Biology has presented us with an array of nanoscale machines capable of converting energy between various optical, chemical, and mechanical forms. For example, Rhodopsin is a protein that serves as the primary light sensor of the eye. It uses light energy to pump protons across a membrane, which ultimately triggers a signal to the brain. At present, we have only a rudimentary knowledge of how these biological machines – constructed from RNA and protein – function. This difficulty in understanding stems from the fact that these biological machines are soft materials driven by weak non-bonded interactions and thermal energy. Experiments have demonstrated [1, 2, 3, 4, 5, 6] that RNA and proteins exist in a huge number of conformational states. (A *conformational state* here is defined by the positions of all the atoms. If there are N atoms in the system, then a conformational state is a point in $\Re^{3N}$. If we look at a protein at different times, then it will be in different conformational states, occupying different points in $\Re^{3N}$. We refer to $\Re^{3N}$ as *phase space*, in this context.) Experimental methods, such as x-ray crystallography, nuclear magnetic resonance, and cryo-electron microscopy, have managed to show us the *average* conformation of a number of proteins and RNA, but have had limited success describing conformational heterogeneity and dynamics, which are important for function. Experimental methods have also been largely unsuccessful at characterizing proteins that occupy a large number of highly diverse conformational states under normal conditions, despite the fact that a third of our proteins are these so-called *intrinsically disordered proteins* [7, 8].

Computation provides an excellent method to identify the individual conformational states of biological systems and their transitions between different conformational states. Both can be predicted using Molecular Dynamics (MD). In MD, forces on atoms, due to

interactions with other atoms, are computed using certain *empirical* force fields. Once force can be computed, Newton's laws of motion are used, almost always with an explicit time integration scheme, to determine the trajectory of the system. The force fields, in turn, are pre-determined by approximating the results of quantum mechanical calculations and experiments on small protein fragments. Quantum mechanical calculations can be used to directly determine forces for use in MD, rather than to determine an empirical force field, but current quantum mechanical methods are too slow to be feasible. The objectives of MD simulations are two-fold: (i) to determine a statistically representative set of conformational states, and (ii) to reproduce the dynamical transitions between these states. Monte Carlo methods can, in principle, be used to determine such ensembles of states. However, they have shown poor efficiency for large, dense systems compared with molecular dynamics.

1.2 Limitation on Time Scale

A limitation of molecular dynamics is the short timescale that can be accessed via simulation. Large-scale protein conformational changes, such as folding and allosteric transitions, normally don't occur in under a millisecond. Molecular dynamics on proteins, limited by high frequency motions to time steps of about a femto second (10^{-15}s), can currently access only about a microsecond of real time. In fact, this limitation of MD has been identified as one of important challenges in computational biology [9].

To illustrate this problem, consider a $10,000$-atom simulation with time step size 1 femto second, carried out to a millisecond of simulation time. Each iteration will require approximately 50 ms of wall-clock time with a fast code, such as GROMACS. The 10^{12} iterations will then require 5×10^{10}s, which is around $1,600$ years of sequential computing time. Using massive parallelism, on say $32,000$ processors, we can solve the same problem in eighteen days, *if we obtain high efficiency*.

1.3 Limitation of Conventional Parallelization

Emerging computing platforms promise to provide the enormous raw computing power required for the above applications. For example, petaflop computers are expected to be available by the end of the decade, systems with more than ten thousand processors are expected to be fairly commonplace in the near future, and an IBM Blue Gene already has

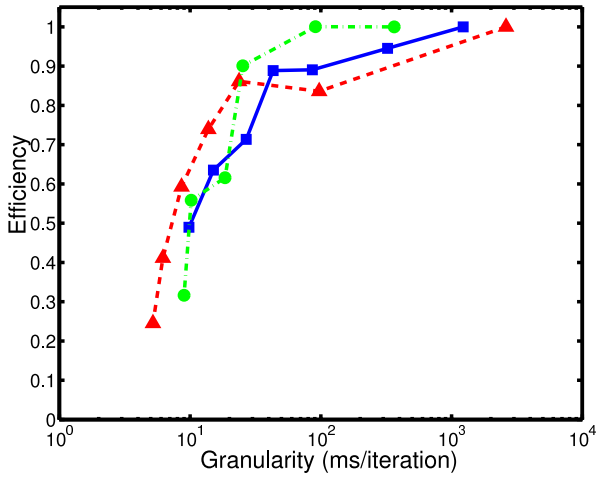

Figure 1.1: *Scaling results on an IBM Blue Gene/L. The solid and dashed lines show results for NAMD, based on data from [16]. The solid line is for a* $327,000$ *atom ATPase PME simulation, and the dashed line for the* $92,000$ *atom ApoA1 PME simulation. The dash-dotted line shows results for IBMs Blue Matter code on a* $43,000$ *atom Rhodopsin system, based on data from [17].*

over a hundred thousand processors. Distributed computing environments, such as grids, also promise abundant computing power. The difficulty is in obtaining high efficiencies with current parallelization strategies.

Conventional spatial decomposition methods (including atom and force decompositions) yield high efficiencies only when the time per iteration per processor is of the order of tens of milliseconds, as shown in fig. 1.1 (the drop in efficiency is even more dramatic on a linear scale). Typical protein simulations have on the order of $30,000$ particles with explicit solvent (that is, when the water molecules are explicitly represented). Furthermore, significant development has gone into implicit solvent methods [10, 11, 12, 13, 14, 15], which replace the explicit water molecules with an effective force field, thereby reducing the number of particles in most proteins to a few thousand atoms. Consequently, spatial decomposition is of limited use for these problems, when using a fast code on a fast processor.

An important cause of parallel inefficiency is that the communication cost dominates over that for useful computations, as the number of processors increases.

1.4 Approach

Our approach is based on the observation that simulations typically occur in a context rich in data from other related simulations. We use such data to parallelize a simulation in the time domain. This leads to a more latency tolerant algorithm than with conventional parallelization. In prior work, we parallelized an important nano-materials application to over two orders of magnitude larger numbers of processors than feasible with conventional parallelization. The soft-matter computations typically encountered in computational biology are more challenging than the hard-matter simulations mentioned above. In fact, it was believed that this approach would not be feasible in such computations. However, in this thesis, we demonstrate the feasibility of this approach to an important soft matter application in computational biology.

We next summarize our approach to time parallelization of MD simulations. We have each processor simulate a different time interval. The difficulty is that, in an initial value problem, the processors will not know the initial state for their time interval until the previous processor has determined its final state. We deal with this as follows. We use MD simulations to determine a relationship between prior results and the current simulation. We then use the results of prior simulations, and their relationship to the current one, to predict the initial state for each time interval. Thus we use all the available knowledge about the physical system's behavior, including the current computation, to predict the starting state for each processor, and then perform accurate simulations in parallel. The MD computations can then be used, again, to verify if the predictions were correct, and to learn the relationship better from differences observed. This process continues. Each of these steps (prediction and verification) is performed in parallel. Some communication is inevitable; however, this overhead is small in practice, as shown later. With a suitable criterion for determining acceptable errors, the solution will always be accurate to the prescribed level; better predictions will enable better speedups. The load is also well balanced. This method can be combined with spatial decomposition to increase the scalability over existing methods.

1.5 Summary

The major problem in biological MD is the difficulty in performing long time-scale simulations. This thesis proposes to solve it using a data-driven time parallelization approach.

We demonstrated its effectiveness in a practical soft matter application. From our results, it appears we can scale the computations efficiently to at least ten processors. Since time-parallelization can be combined with spatial parallelization, we expect to get an additional one order of magnitude improvement in speed over purely spatial parallelization.

The outline of the rest of this thesis is as follows. In Chapter 2, we present background information on an important soft matter application in biology that will be used to demonstrate effectiveness of our time parallelization method. In Chapter 3, we summarize our prior work on a hard-matter application involving Carbon Nanotubes, and describe specific challenges in soft-matter applications. Next, we describe related works on parallelization, ODE-theory based solutions, and other's parallelization methods in Chapter 4. We show our result in terms of speedup and errors in Chapter 5. Finally, chapter 6 presents the conclusion, limitations of the current work, and directions for future research.

CHAPTER 2

BACKGROUND

2.1 Molecular Dynamics in Biology

Molecular Dynamics (MD) is a computational simulation method that determine the position r_i and velocity v_i of every atom $i = 1, ..., N$, that is contained in a computational cell subjected to external boundary conditions (force, pressure, temperature or velocities). A differential equation of motion that solves for these 6N (3 positions and 3 momentum components) variables is given by $\vec{F_i} = m_i \frac{\partial \vec{v_i}}{\partial t}$, $\vec{v_i} = \frac{\partial \vec{r_i}}{\partial t}$, where $\vec{F_i}$ is the force on an atom i having mass m_i. In a numerical scheme, ∂t is approximated by Δt. The problem is solved by iteratively computing the states at successive points in time, using a forward time numerical difference approximation. We shall refer to each iteration as a time step. A large computational effort can be involved when the state of the system is large, or when the number of time steps is large. In order to reduce the computation time, parallelization is often used, especially with large physical systems. Even when the state space is not large, the computational effort can be large if we need to compute for a large number of time steps. For example, hemoglobin has two energetic states: relaxed state and tense state. The transition from relaxed state to tense state takes more than tens of microseconds. It will take ten billion iterations to simulate this, using MD with a time step size of a femto second.

The difficulty in simulating to long time spans, using MD, has been identified as one of the important challenges in nanoscale simulations and computational materials science [9, 18]. Conventional parallelization is not effective in such problems, because the granularity becomes fine, limiting scalability. Furthermore, a single MD trajectory does not give useful information in soft-matter computations. Rather, we compute a number of trajectories, and perform some statistical averaging of quantities of interest. This will be explained in section 3.3.

MD requires good algorithms to integrate Newton's equation of motion. Newton's equations of motion are laws, which provide relationships between the forces acting on a body and the motion of the body. The Important criteria of a good algorithm are *energy conservation, time reversible, and symplectic*, that is, incompressible phase space flow. These properties are believed necessary in soft-matter simulations. There are two kinds of energy conservation, short time and long time. A good algorithm has no overall energy drifts for long time spans. Time reversible means future and past phase space coordinates play a symmetric role. If one were to reverse the momenta of all particles at a given instant, the system would track back its trajectory in phase space. These methods are believed to exhibit the shadowing property, which means that they follow the exact trajectory of an approximation to the force field used. Since the force fields used are themselves approximations, time reversible symplectic integrators are considered accurate. Modifications are made to the equations of motion to control temperature and pressure during the simulation. For example in *constant-NVT ensemble* simulations, the total number of particles, volume, and temperature are kept constant [19].

2.2 Force Spectroscopy/AFM

2.2.1 Titin

The biological system we consider is *Titin*. Titin is a giant multi-domain muscle protein forming a major component of vertebrate muscle. The properties of Titin are studied by characterization of its individual domains. Each Titin domain is identical - it is composed of many identical fibronectin type domains connected into a single unbranched polymer, which undergo very small change in the presence of neighboring domains. Therefore, its properties, such as muscle elasticity, can be determined by studying the properties of each individual domain using protein-unfolding experiments. Traditional unfolding experiments use chemical denaturants to cause unfolding of a protein structure [20].

2.2.2 Force Spectroscopy/Atomic Force Microscopy

In *Atomic Force Spectroscopy*, mechanical force is applied to two ends of a molecule. This applied force produces changes (such as unfolding) that are described by a force-extension profile – how much force is applied versus the relative separation of the points at which

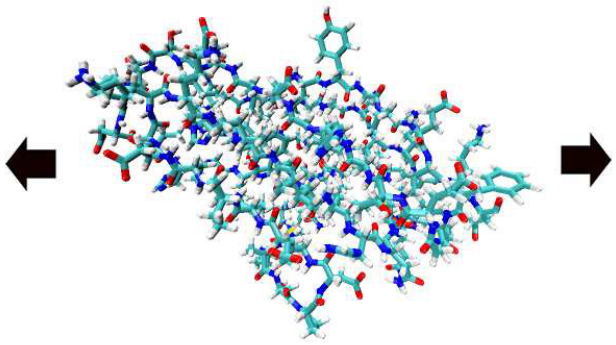

Figure 2.1: *AFM pulling of TI* 127 *mutant from two opposite directions.*

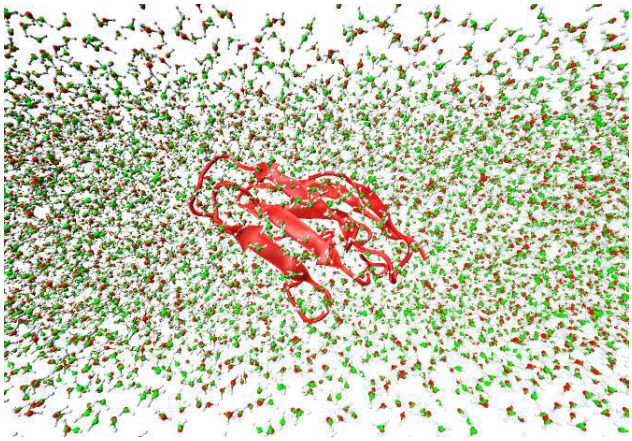

Figure 2.2: *Adding water molecules as a solvent before simulation starts. More than* $9,525$ *tip4p water molecules are added here.* 8 $Na+$ *and* 3 $Cl-$ *ions are also added to maintain charge neutrality.*

force is applied. *Atomic Force Microscopy*(AFM) is becoming an important biophysical technique, where force plays the role of chemical denaturant [21, 22, 23]. Force Spectroscopy has been shown to be relatively easily adapted to the study protein unfolding events in atomic resolution. In MD simulation, a protein native structure TI 127 (a Titin mutant with A-strand deleted [24]) is mechanically pulled apart from two attachment headpieces at constant pulling rate, as shown in fig. 2.1. The number of particles, volume, and temperature

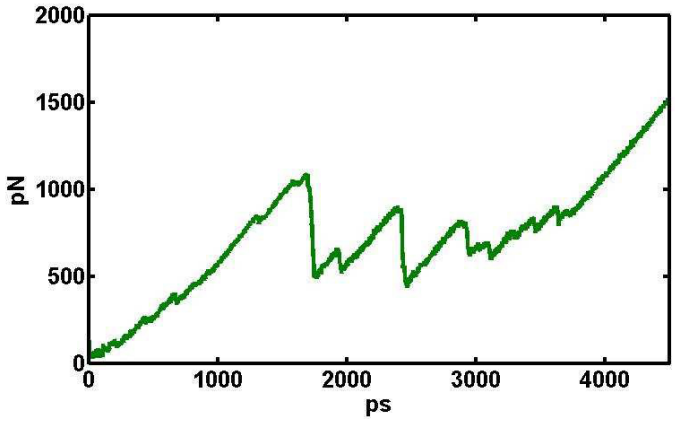

Figure 2.3: *A sample of Force extension profile generated by GROMACS AFM pulling of TI* 127 *at pulling rate of* 10 *m/s.*

are kept constant. Tip4p water molecules [1] are added as solvent, as shown in fig. 2.2. We also add *excess* ions to approximate the experimental salt concentration. Simulations have to be performed at constant high temperature (400 K) to induce the proteins to unfold on a computationally accessible timescale. We use Nośe Hoover Thermostat method to maintain constant temperature. This approach is based on the use of an extended Lagrangian that contains additional term of artificial coordinates and velocities adding to the equation of motion [25, 26, 27, 28]. Movement of the pulled ends causes an extension of the protein and the force exerted can be measured by multiplying the spring constant K by the extension of the spring: Δl, $F(t) = K\ \Delta l\ t$. The resulting forces are computed during each time step to generate a force extension profile, as shown in fig. 2.3. The force extension profile has a series of saw-tooth peaks corresponding to unfolding of protein domains, as shown in figures 2.4 2.5 2.6.

Unlike traditional bulk measurements of protein solutions, AFM is a single molecule technique able to directly probe the individual, heterogeneous response of single molecules [20]. The effect of force on the energy landscape can be studies in atomic level. AFM technique also allows us to study particular regions of the protein structure which was not assessable from conventional chemical denaturant method. Variation of AFM attachment points also allows one to measure multiple pair-wise separations in a single structure – essentially constituting

[1] *tip4p* is a GROMACS water molecule model which has 2 hydrogen atoms and 2 oxygen atoms.

N^2 different structural probes.

Despite the unique sensitivity and control of AFM, models are necessary to interpret what structural changes are producing the measured *force extension profile*. Furthermore, AFM measurements are limited in their pulling speed, and the use of a finite pulling speed directly affects the mechanism of folding [29, 30, 20, 22, 31]. Early biological AFM measurements observed relatively simple force extension profile. Recent tests of other proteins show that multiple intermediates and soft deformations are possible, making the pulling rate especially important for determining the mechanism. Unfortunately, traditional MD simulations are limited to rates of pulling that are several orders of magnitude faster than possible experimentally due to the lack of computational power. Current simulations are limited to reproducing pulling rates in the range of 1-10 m/s, compared with typical experimental rates of 10^{-7}–10^{-8} m/s.

A method to extend the time-scale of simulations of AFM is needed. The preservation of the natural kinetics is important for connecting to experiments, so potential energy surface transformation methods are inapplicable here. Conventional parallelization is effective but limited by granularity as discussed previously. Time parallelization is one method to efficiently extend the time-scale by utilizing massively parallel machines. It may still need to be combined with other techniques, such as Markov State Modeling, to reach realistic time scales.

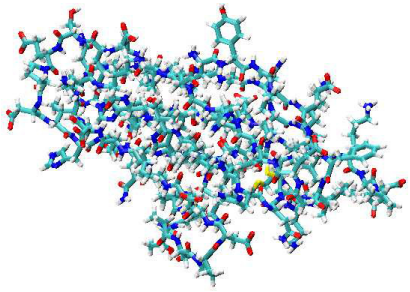

Figure 2.4: *Initial state before the AFM pulling starts.*

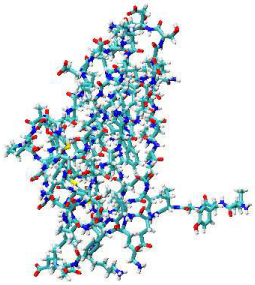

Figure 2.5: *State before a strand breaks apart.*

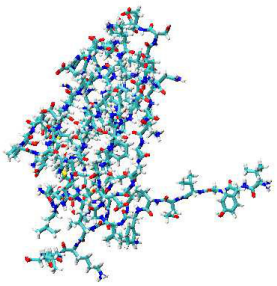

Figure 2.6: *State after a strand breaks apart.*

CHAPTER 3

DATA DRIVEN TIME PARALLELIZATION

3.1 Prior Work on Carbon Nanotube

In this section, we describe our prior work on carbon nanotube (CNT) based on the idea of parallelizing time using a data driven approach [32, 33, 34]. This work is a hard-matter computation in contrast to the soft-matter computation of the current work. Data-driven parallelization is feasible, because, usually, high pulling-speed results are available. We can use this high pulling-speed data to obtain slow (more realistic) pulling speed results. Figure 3.1 illustrates our approach. Let us call a few iterations, say $1,000$ time steps of an ODE solver, as a *time interval*. We divide the total number of time steps needed into a number of time intervals. Ideally, the number of intervals should be much greater than the number of processors. Let t_{i-1} denote the beginning of the i th interval. Each processor $i \in \{1 \cdots P\}$, somehow (to be described later) predicts the states at times t_{i-1} and t_i in parallel (except for the known state at t_0), using data from prior simulations. It then performs accurate MD computations, starting from the predicted state at time t_{i-1} up to time t_i, to *verify* if the prediction for t_i is close to the computed result. Both prediction and verification are done in parallel. If the predicted result is close to the computed one, then the initial state for processor $i+1$ was accurate, and so the computed result for processor $i+1$ too is accurate, provided the predicted state for time t_{i-1} was accurate. Note that processor 1 always starts from a state known to be accurate, and so the algorithm *always progresses* at least one time interval, since the accurate computations on processor 1 lead to accurate results on that processor. In fig. 3.1, the predicted state for t_3 was inaccurate, and we say that processor 3 *erred*. Computations for subsequent points in time too have to be discarded, since they might have started from incorrect start states. The next phase starts from time t_3, with the initial state being the correctly computed state at time t_3, and we

compute states for times t_4, t_5, t_6, and t_7 in parallel. The errors observed in the previous verification step can be used to improve the predictor by better determining the relationship between the current simulation and prior ones.

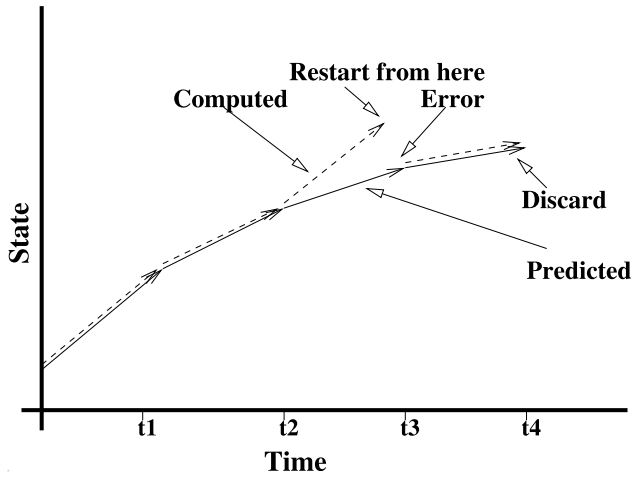

Figure 3.1: *Schematic of time parallelization.*

Note the following: (i) Processor 1's result is correct, since it always starts from a state know to be accurate. So the computation always progresses. All the processors must use the same predictor. Otherwise verification of prediction at time t_i on processor i does not imply that the prediction for initial state at time t_i on processor $i+1$ was correct. (ii) A global communication operation (AllReduce call in Algorithm 1) is used to detect the smallest ranked processor to err. (iii) The last correctly computed state (using MD, in the verification step), is sent to the first processor. (iv) If the time interval consists of a large number of time steps, then the communication and prediction overheads are relatively negligible, leading to a very latency tolerant algorithm.

Prediction We will describe a particular predictor used in our CNT application. The most important feature of our strategy is our ability to predict the state, which serves as the starting point for each processor, from its relationship with a base simulation. The predictor should be both accurate much of the time, and much faster than the verifier. Prediction over a long period of time is difficult. Instead, if $\hat{S}_i$ is the most recently computed state that is accurate, then we will predict the changes between $\hat{S}_i$ and the state at the times

Algorithm 1: Time Parallelize (Initial State S_0, Number of processors P, Number of time intervals m)

1: $i \leftarrow 0$
2: $\hat{S}_0 \leftarrow S_0$
3: **while** $i < m$ **do**
4: **for** *each processor* $j \in [1, min(P, m-i-1)]$ **do**
5: $T_{i+j-1} \leftarrow Predict(\hat{S}_i, i, i+j-1)$
6: $T_{i+j} \leftarrow Predict(\hat{S}_i, i, i+j)$
7: $\hat{S}_{i+j} \leftarrow Compute(StartState = T_{i+j-1}, StartTime = i+j-1, EndTime = i+j)$
8: $UpdatePredictionParameters(CurrentParameters, \hat{S}_{i+j}, T_{i+j})$
9: **if** $DifferenceTooLarge(\hat{S}_{i+j}, T_{i+j})$ **then**
10: $Next_j \leftarrow j$
11: **else**
12: $Next_j \leftarrow P$
13: **end if**
14: **end for**
15: $k \leftarrow AllReduce(Next, min)$
16: **if** $j = k$ **then**
17: $Broadcast(\hat{S}_{i+j}, PredictionParameters)$
18: **end if**
19: **for** *each processorj* $\in [1, P]$ **do**
20: $i \leftarrow i + k$
21: **end for**
22: **end while**

required, as shown in the call to *Predict* in algorithm 1. We accomplish this by predicting the change in each coordinate of the positions of the atoms independently. We normalize all the coordinates so that they are in $[0, 1]$, by letting the origin be 0 and then dividing by the length of the CNT along that coordinate direction. Similarly, we normalize the relative times in the base and in the current simulations by multiplying by the velocity with which one end of the tube is pulled, and dividing by the original length of the tube. For example, if the current simulation is pulled at one tenth the velocity as the base, then time t in the current simulation is related to time $t/10$ in the base. Let x_t represent a coordinate at time t. Then by two terms of taylor's series, $x_{t+\Delta t} = x_t + \dot{x}_{t+\Delta t}\Delta t$, where $\dot{x}_{t+\Delta t}$ is the actual slope $\partial x/\partial t$ at some point in $[t, t+\Delta t]$.

We do not know $\dot{x}_{t+\Delta t}$, and we will try to predict this value. Let $\dot{x}_{t+\Delta t} \approx \sum_i a_{i,t+\Delta t}\phi_i(x_i)$,

we can perform a least square fit to determine the coefficient $a_{i,t}$. We can determine coefficient of base, $b_{i,t}$. If the base simulation and the current simulation are almost identical, then we can say $a_{i,t} \approx b_{i,t}$. If they are different, we wish to correct by adding the difference between two simulations $R_{t+\Delta t} = a_{i,t+\Delta t} - b_{i,t+\Delta t}$, which is unknown. We can assume $a_{i,t+\Delta t} \approx b_{i,t+\Delta t} + a_{i,t} - b_{i,t}$. random fluctuations in MD simulation lead to poor results if we depend on only evaluation at one poin in time So we set $R_{t+\Delta t} = (1-\beta)R_t + \beta(a_{i,t} - bi, t)$, where β is the weight assigned to the latest value. Updating R_t each time step represents a simple form of learning. Since $a_{i,0}$ and $b_{i,0}$ are unknown, we assume a linear increase with time, in the values of the coordinates of atoms, in the direction in which CNT is pulled, with the constant of proportionality being a function of its normalized coordinates.

Verification The verification step consists of an accurate MD simulation, starting from a possibly inaccurate initial state. The computed state is then compared with the predicted state for the same point in time. We need to determine if the two state are sufficiently close. The detail steps are described in [33].

Validation Tensile test is an important simulation or experiment to determine the mechanical properties of a physical system. In a tensile test, the physical system is pulled at a constant velocity. The response of the material is characterized by stress for a given strain. Stress is the ratio of force to directed area, while strain is the ratio of deformation to the original dimensions of the material. The stress-strain relationship is an important material property, indicating the coarse-scale effect of the atomic motions. Figure 3.2 shows that the stress-strain relationship from the Carbon Nanotube (CNT) time parallel code is almost identical to the exact sequential results. Such a curve, for example, can be used by an FEM code to determine the effect of the polymer matrix on the CNT, and vice-versa. Another important property would be the strain at which the CNT starts to break.

Speedup Results: The speedup is almost linear, as shown earlier in fig. 3.3, with efficiencies typically well over 95%. Until the point that the CNT starts failing, the predictions are sufficiently accurate, and loss in speedup occurs only due to prediction and communications overheads, which are small relative to the computations required for a time interval. Note that the speedup is relative to an inherently sequentially algorithm; we are,

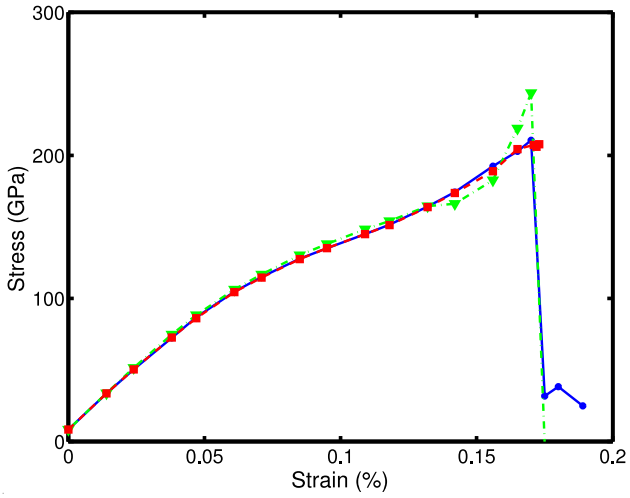

Figure 3.2: *Plot of stress versus strain at* $0.1m/s$. *The squares show the time parallel results on* 400 *processors. The solid line with circles (which are mostly obscured by the squares, since the time-parallel results almost coincide with the exact ones) represents the exact sequential MD results. The dash-dotted line with triangles represents the results of direct prediction.*

therefore, comparing against an optimized code that does not have any of the overheads of our approach. The sequential code required around a week of computing time. We give some timing data below to show that the parallelization overheads are small. On the Xeon cluster at NCSA, the prediction related computations take less than 10^{-4}s, the AllReduce $\approx 10^{-4} - 10^{-3}$s for 50-1000 processors, Broadcast $\approx 10^{-4}$s for 50-1000 processors, and the Send/Recv about 10^{-4}s. Load imbalance is not an issue, since each processor performs, essentially, the same amount of computation. All the overheads are insignificant, relative to the computation time ($\approx$ 13s) for simulating a single time interval.

3.2 Multiple Time Scales

The coarse scale results are what are important in many applications, such as in the CNT tensile test, where the stress-strain response is what is important. Figure 3.4 shows schematic of a simulation on multiple time scales. In our scheme, through the use of information from related simulations, we are able to get the coarse scale response without performing a costly sequential simulation. This helps with the accuracy of prediction, and with parallelizing the prediction phase.

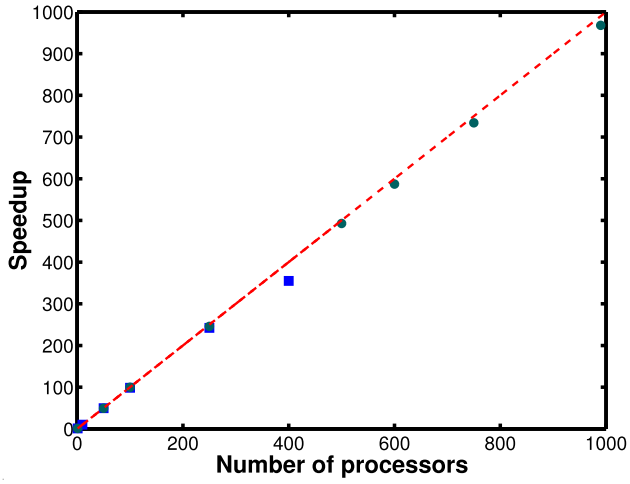

Figure 3.3: *Speedup curve on a nano-materials simulation. The dashed line shows ideal speedup. The squares and circles give the speedups with two different predictors, on a commodity Xeon cluster with Myrinet interconnect at NCSA. Conventional parallelization scales to only* $2-3$ *processors.*

3.3 Challenges in Soft-Matter Applications

We summarize the challenges in soft-matter applications as follows. The challenges in soft-matter applications are related to their complexity, and the underlying physics and chemistry of these processes are not yet completely understood. For example, folding and unfolding in proteins are dynamic processes with formation and loss the large number of weak non-covalent bonds. These bonds will fail under any level of pulling force if held for sufficient time. Bond strength is the force that produces the most frequent failure in repeated tests of breakage. Therefore, MD simulation is not to predict precisely what will happen in the experiment. However we are interested in statistical average behavior of a system with limited knowledge of initial state. Unlike hard-matter simulations which display mostly similar behaviors, soft-matters simulations usually exhibit diffusive heterogeneous behaviors. For all MD simulations, trajectory of the system depends on its initial conditions. Two different simulations with slightly different initial conditions will diverge exponentially as time progresses. Moreover, small integration errors can cause the simulated trajectory to diverge exponentially from the true one. This is also called Lyapunov instability [19]. During soft-matter simulations, local minimal energy is more difficult to predict. This is due to small

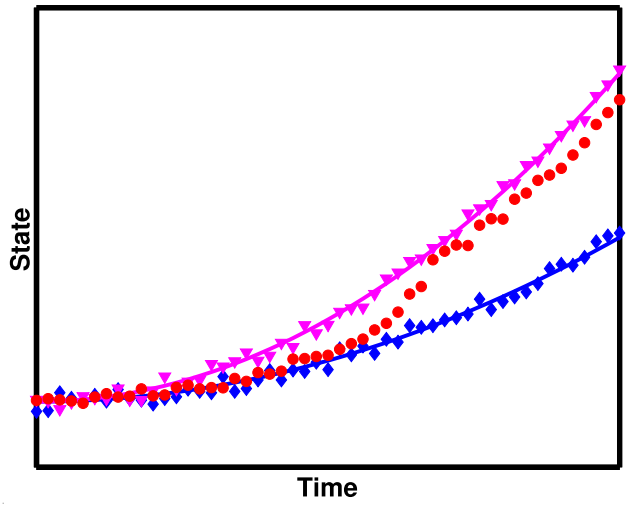

Figure 3.4: *Schematic of a simulation, showing multiple time scales. The states vibrate around the lines, showing a coarse scale response. The simulation marked by circles initially has a coarse-scale response similar to the simulation marked with diamonds. However, later, it changes to become similar to the simulations marked with triangles. This change is often caused by the vibrations. Thus the fine time-scale cannot be ignored in the actual simulation (in solving the ODE), even if we are interested only in observing the coarse scale response. Our prediction mechanism may err during the transition, but will then identify the new coarse scale behavior later.*

MD integrations time steps, limited by its high frequency motions. On the other hand, local minimal energy in hard-matter simulations tend to present harmonic behavior in contrast to those in soft-matter simulations.

3.4 Specific Approach in the Biological Problems

In this section, we present our implementation of the data-driven time-parallelization approach, for a class of biological MD simulations - AFM pulling of proteins (Algorithm 2). This algorithm is similar to that in the CNT application. The primarily difference is in prediction. The prediction process does not compute the differences in states. Instead, the prior data consists of a collection of results from prior simulations. At any point in time, we dynamically determine a prior simulation to which the current simulation's behavior is current similar. We use the actual states of that prior simulation to predict the states of the current simulation in the future. Furthermore, there are no prediction parameters that need to be broadcast to all processors.

During verification phase, we have to choose some criteria to determine what an acceptable error for each time interval is. We have results of faster pulling runs, which have different seeds to the random number generator used in the thermostat, and also different initial states (initial velocities of the atoms). The root mean square deviation (RMSD) for a collection of N values $\{x_1, x_2, x_3, \ldots, x_N\}$ is:

$$x_{rms} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2} = \sqrt{\frac{x_1^2 + x_2^2 + x_3^2 + \ldots x_N^2}{N}}, \tag{3.1}$$

where x_i are the differences in values of corresponding components of two vectors. The RMSD indicates how much the two structures differ. We will describe the procedures of selecting criteria as follows. First, we remove solvent water molecules from those results. Next, translation and rotation is added to the coordinates by GROMACS *dofit* function in order to remove rigid body motion of the system. Then, we calculate RMSD and maximum distance from a pair of systems. Figure 3.5 and fig. 3.6 show the root mean square deviation (RMSD) versus time and maximum distance versus time from two faster pulling runs *which differ only in the random number seed used*. Only 10 pico seconds of data are used in fig. 3.5 and fig. 3.6 because each time interval is 10 pico seconds in time parallelized simulation. From those figures, we set RMSD threshold to $2\mathring{A}$, and maximum distance to $10\mathring{A}$. Finally, we add RMSD and maximum error as thresholds in our time parallelized code. In algorithm 2, function *DifferenceTooLarge* checks the predicted state to see if its RMSD value and maximum error is larger than these thresholds. If it is the true, we say this processor has failed, and computations for subsequent points in time too have to be discarded, since they may have started from incorrect initial states.

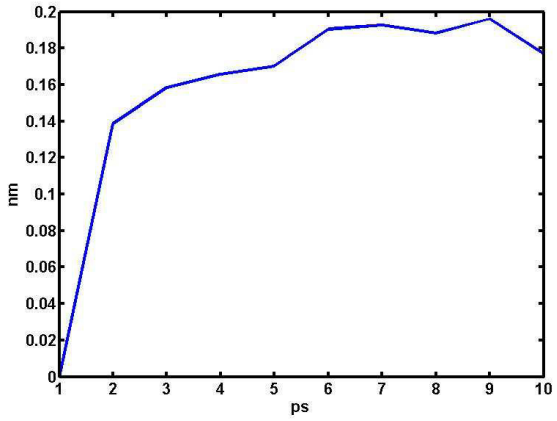

Figure 3.5: *RMSD versus time. Two runs have the same pulling rate but different seeds to the random number generator used in the thermostat.*

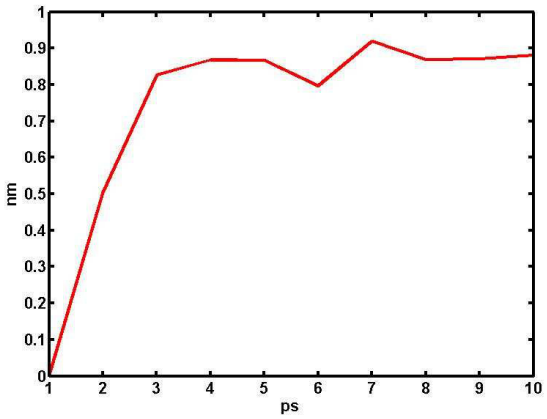

Figure 3.6: *Maximum distance versus time. Two runs have the same pulling rate but different seeds to the random number generator used in the thermostat.*

Algorithm 2: Time Parallelize (Initial State S_0, Number of processors P, Number of time intervals m)

1: $t \leftarrow 0$
2: $\hat{S}_0 \leftarrow S_0$
3: **while** $t < m$ **do**
4: **for** *each processor* $i \in [1, min(P, m-i-1)]$ **do**
5: $S_{t+i-1} \leftarrow Predict(t+i-1)$
6: $S_{t+i} \leftarrow Predict(t+i)$
7: $Accurate\ mdrun(StartState = S_{t+i-1}, StartTime = t+i-1, EndTime = t+i)$
8: $\hat{S}_{t+i} \leftarrow read\ state\ completed\ by\ mdrun\ for\ t+i$
9: **if** $RMSD\ or\ Max\ difference\ too\ large(\hat{S}_{t+i}, S_{t+i})$ **then**
10: $Next_j \leftarrow i$
11: **else**
12: $Next_j \leftarrow P$
13: **end if**
14: **end for**
15: $k \leftarrow AllReduce(Next, min)$
16: **for** *each processorj* $\in [1, P]$ **do**
17: $t \leftarrow t + k$
18: **end for**
19: **end while**

CHAPTER 4

RELATED WORK

4.1 Time Parallelization

In the 1980s and 90s, time parallelization using waveform relaxation [35, 36], and its various variants [37, 38], generically called "dynamic iterations", were well studied. To give a simple illustration of this method, consider the ODE $\dot{x} = f(x,t),\ x(t_0) = x_0$. An initial guess, $x^{(0)}(t)$, for the solution is taken for all $t \geq t_0$, subject to $x^{(0)}(t_0) = x_0$. Then an iteration of the form $\dot{x}^{(k+1)} = \hat{f}(x^{(k+1)},\ x^{(k)}, t)$ is used, with $x^{(k+1)}(t_0) = x_0$ and subject to $\hat{f}(u,u,t) = f(u,t)$ for any u, t. Conditions on $\hat{f}$ for convergence have been derived, and are fairly simple.

In the simplest case, if $\hat{f}(x^{(k+1)},\ x^{(k)}, t) = f(x^{(k)}, t)$, then we get the Picard iterations, which is illustrated in fig. 4.1. It is easy to parallelize this by having each processor integrate a different time interval, and then performing a parallel prefix operation. Parallel prefix is fairly fast in practice, and takes $O(\log P)$ time on P processors under, for instance, the task-channel model of computation. Unfortunately, the convergence rate of Picard iterations is slow.

Popular versions of waveform relaxation *split the system* into subsystems, and use analogues of Jacobi, Gauss-Seidel, or SOR (successive over relaxation) iterations. These can be explained in terms of how $\hat{f}$ is constructed, and details can be found in, for example, [36]. The Jacobi version of waveform relaxation is easy to parallelize in the time domain. Note that if we define $\hat{f}(x^{(k+1)},\ x^{(k)}, t) = f(x^{(k+1)}, t)$, then we get convergence in one iteration, but the problem is as difficult to solve as the original one.

While the asymptotic convergence rate form waveform relaxation is good, the convergence rate observed in practice is not good. One of the reasons for the slow convergence is that the effect of the initial condition takes long to propagate to the later values of time. So the basic sequential algorithm is often much slower than a conventional ODE solver, using

an equivalent integration scheme, and efficient parallelization cannot usually overcome this deficiency.

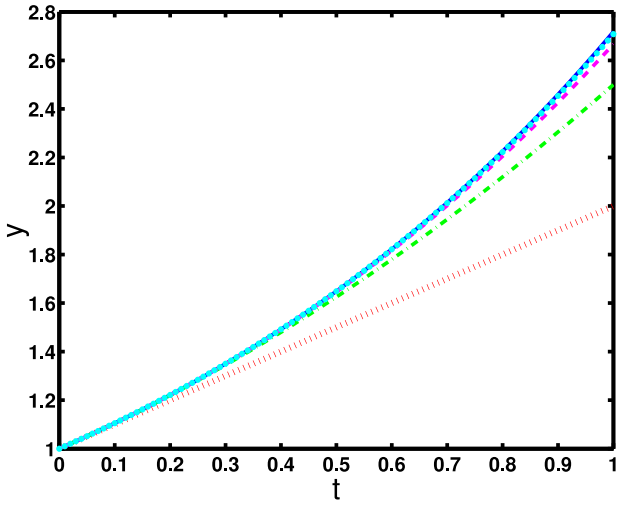

Figure 4.1: *Picard iterations to solve $y' = y$, $y(0) = 1$. The initial guess is along the x-axis. The dotted, dash-dotted, and dashed lines show the results of the first, second, and third iterations respectively, while the solid (top most) line shows the exact result, which visually coincides with the results of the fourth iteration, marked by a gray dashed line.*

Note that in our approach, in the worst case, the prediction fails at each step. The computation then reduces to a sequential one using a conventional ODE solver. An additional computational overheard of parallelization is incurred, but this is small. Furthermore, use of prior data attempts to propagate the effect of the initial condition faster in the following manner. The prior results have already accounted for the effects of their initial states. If the current simulation appears to be related to a combination of that of previous ones, then previous ones have, in some sense, performed the necessary computations, and we can "read" the effects of the current simulation's initial conditions from them. When prediction is invalid, the verification step detects the errors, and we lose speedup. This typically happens during transitions, such as bond breakage. Once a transition has been completed, we will again be able to detect some other combination of prior simulations to which the current simulation is now related. Note that our prediction strategy may enable us to combine our technique with waveform relation; if the predicted state is close to the true state, then the good asymptotic convergence rate of waveform relaxation may apply.

Parareal approach Other time parallelization approaches, related to shooting methods and also to ours, have attracted some attention in recent times. An example is the parareal approach [39, 40].

The parareal approach uses an iterative procedure, with each iteration consisting of a predict-verify-correct sequence. Predictor approximates solution from initial state. All starting states can be obtained by recursion. This is done in sequential. The verify step verifies if the predicted state are accurate in parallel. If the predictions are not correct, then a correction step is applied. The error is computed and this concludes an iteration of the predict-verify-correct sequence. The next iteration predicts the state by applying the inaccurate predictor, but then correcting by the error that was encountered the previous iteration. The verification and correction phase are applied as before, to complete this iteration. If the solutions have not converged, then the computation repeats until convergence.

This method requires a sequential computation for the prediction process, which eventually limits speedup and decreases efficiency. In fact, simulated results in [39, 40] showed limited speedup (8 – 130) and efficiency (25% – 1%), even ignoring communication cost (since they were simulations of parallel runs, rather than real ones), on some model problems. Communication costs increases linearly with the number of processors (due to the sequential computation of the predictor). The ratio of the time taken by the verifier to the time taken by the predictor is usually small. Even if convergence takes place in one iteration, the speedup is bound by this ratio. The number of iterations for convergence clearly affects the efficiency significantly. Note that as the iterations proceed, even though states for many of the points in time would have already been computed correctly, the processors responsible for those points are still involved in the computation. More importantly, the use of a coarser time grid is often not feasible. For example, in MD computations, taking time steps as large as 2 femto seconds can cause the solution to blow up. On the other hand, if the time step in the sequential phase is small, then this limits the parallel speedup. In our scheme, through the use of information from related simulations, we are able to get the coarse scale response without performing a sequential simulation with large time steps. This helps with the accuracy of prediction, and with parallelizing the prediction phase. Of course, the prediction and verification algorithms that we use are also very different from those used in the above approach.

4.2 Spatial Parallelization

Spatial parallelization[1] of MD is well studied [41, 42], including ours [43], which scale at granularities greater than a few hundreds of atoms per processor. Recent results on the IBM Blue Gene [17, 16] appear to scale to granularities of the order of one to ten atoms per processor. While these results are certainly impressive, a closer look only reinforces the limitations due to fine granularity, for the following reasons. When the computational cost is high, the communication cost is relatively low, leading to good speedups. While the exact sequential time is not reported in [17], from the results on smaller numbers of processors, it appears that the sequential time per time step per atom is slower than that of a fast code (GROMACS) on a fast processor (Xeon) by a factor of around 5. When the underlying (sequential) code is faster, the speedup results will not be as good, because the communication cost will be relatively high. Furthermore, while the above results show decreasing time with increasing number of processors, the improvement is marginal at low granularities. For example, increasing the number of nodes from around $8,000$ (at a granularity of 5.3 atoms per processor) to around $16,000$ (at a granularity of 2.6 atoms per processor), increases the speed by only around 10%. Basically, codes typically do not scale well at granularities finer than the order of tens of milliseconds per iteration in a parallel run, as shown in fig. 1.1.

The GROMACS code on Xeon appears to run around 10 times faster sequentially than NAMD on the Blue Gene. The parallel efficiency of GROMACS is lower than for NAMD. However, even if it were made to scale efficiently at granularities of 10 ms per time step, there is limited scope for efficient parallelization of systems with a few thousand atoms, if we consider that simulation of a $1,000$ atom system on Xeon takes around 5 ms using GROMACS. Thus, fast codes on a fast processors currently do not appear capable of scaling *efficiently* to granularities finer than a hundred atoms per processor.

4.3 Other Parallelization Methods

PES Smoothing A number of popular techniques for dealing with long time scales are based on modifications of the force field. These generally involve smoothing or flattening the

[1] We use this term for both atom-based decomposition (which assigns different atoms to different processors), and to force-decomposition (which assigns different force computations to different processors).

potential energy surface (the potential energy surface or PES is the potential energy written as a function of the phase space $\Re^{3N}$). Smoothing of the PES increases the probability of accessing high energy transition states that are rate limiting bottlenecks for conformational change.

A number of possibilities for PES smoothing exist. In the earliest methods [44, 45], smoothing transformations were combined with search algorithms in attempts to locate the global minimum of the PES. These methods do not generate correct statistical ensembles of conformations or dynamics.

More recent PES smoothing methods combine MC moves with additional control parameters that allow for the dynamic adjustment of the amount of smoothing. These methods [46, 47, 48, 49, 50, 51] are normally referred to as multi-ensemble methods because they generate many ensembles simultaneously, one for each value of the control parameters. Although these methods are guaranteed to produce a statistically correct ensemble of conformations, they do not preserve the kinetic behavior. These newer methods have been successfully applied to small proteins (fewer than 30 amino acid residues) designed to exhibit exceptionally fast dynamical transitions (under 10 microseconds) but have so far been limited by poor scaling with the size of the protein. Few PES smoothing methods exist that preserve the transition mechanisms and from which transition rates can be quantitatively computed. In the Hyper Molecular Dynamics method [52, 53, 54], PES smoothing occurs by *filling in* low energy wells. For systems with well-defined high-energy barriers, this method does not disrupt the relative probability of transitions out the current potential energy well [55].

Kinetic Monte Carlo In Kinetic Monte Carlo, local transition rates are pre-computed to determine the rate that different transitions would occur. Its advantages and limitations are well described in [56, 55].

Parallel Replica Method Molecular Dynamics simulation on small system (e.g., $< 10^3$ atoms) is inefficient because high communication cost. Parallel Replica method [57] exploits the properties of infrequent-event system as a way to develop an efficient parallel approach to MD simulations. A typical feature of infrequent-event system is the separation of two durations of time scales: a brief transition period and long waiting time between these periods. We can save time by not doing MD integration of the long waiting period, and

Transition-state theory (TST) can be used to compute the crossing events directly.

Parallel Replica method procedures are the followings: (1) The configurations are copied to M number of processors. (2) The minimization process to determine reference configuration. (3) On each processor a classic integrations is perform on uncorrelated replicas. (4) Each replica trajectory is monitored for transition event. When one processor i detects an event it notify all other processors and stop them. The total time is added by t_{sum}(the sum of time done by all M replicas since step (3)). (6) One processor, replica i is integrated forward for pre-chosen time during which new transitions may occur. (7) Replica i becomes the new configuration of the system. (8) Restart step (1).

The Parallel Replica Method is a rudimentary type of time-parallelization that can be used on systems with certain properties: correlation times should be short compared with the times for transitions to occur, and transition events should be well defined, brief events. The rate for transitions increases linearly with the number of simulated systems when correlation times within each basin are short compared with transition events. Applications of this method to soft systems such as proteins have been limited because of the difficulty in identifying transitions and the lack of large time-scale separation between relaxation in a well and inter-well transitions.

Markov State Modeling The use of grid computing, hundreds of thousands of computing nodes connected by internet, to study protein folding was pioneered by Pande's group. From $2000 - 2005$, his project Folding@Home [58] has already carried out a number of important and extremely difficult calculations. Since October 1, 2000, over $1,000,000$ CPUs throughout the world have donated time to the project. This has resulted hundreds to thousands times more computing power than what we typically find in a computing laboratory and enable studies at details we had not thought possible.

Pande has the following steps. First, Pande selects some starting points from previous data that is easily obtainable. Next, Pande starts tens of thousands of independent MD trajectories from those starting points. Then, Pande groups similar trajectories into discrete states by some arbitrary metric. Assuming the transitions between these states are Markovian, Pande estimates the transition between states by counting the number of times we see each transition. From this Markovian state model, as shown in fig. 4.2, it is possible to efficiently predict probability that a protein will fold. By constructing Markovian state

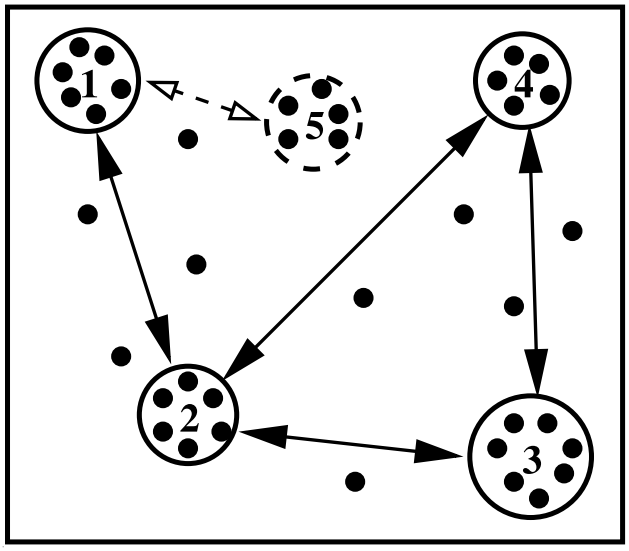

Figure 4.2: *A Markov chain describes at successive times the states of a system. At these times the system may have changed from the state it was in the moment before to another or stayed in the same state. The changes of state are called transitions. A Markov chain is a sequence of random variables X1, X2, X3, ... with Markov property, namely that, given the present state, the future and past states are independent. Markov chains are often described by a directed graph, where the edges are labeled by the probabilities of going from one state to the other states.*

model from those collected data, Pande is able to interpolate dynamics further than direct simulations [59, 60, 61].

However, it is shown if the conformations are grouped incorrectly, the state space is no longer Markovian, and any analysis that assumes a Markovian process may produce incorrect results. Even if the states are defined such that transitions between them are Markovian, the results could still be in error [62, 63].

Multiple Time-step Integration More conventional methods to increase the computation rate, such as multiple time-step integration methods, are frequently used. Different time-step sizes for different components of the computations are used in [64] to deal with small time-step sizes in atomistic simulations, and many multi-rate techniques [65] have been developed and analyzed. These give improvements of a factor of three or four. But it is difficult to achieve much more of an improvement using this approach, due to resonance, which limits how large the largest time-step can be [66]. There is much other research being

conducted on numerical methods for handling the time-scale problem [67] based on ODE theory. For example, equation-free schemes [68] attempt to determine coarse time-scale behavior from fine-time scale (MD) simulations, and use this in a solver that can use coarse-scale response. Coarse-scale states are "lifted" to the higher-dimensional fine-scale state, then fine-scale simulations are performed and a coarse-scale response obtained ("restricted"). A large step is then taken on the coarser-scale, to progress more rapidly in time. This process continues. Apart from parallelization, one important difference between our idea and the above one is that we use prior data to determine coarse-scale behavior. The above technique uses the fine-scale response of the current simulation itself, which limits how far in the future it can predict. However, in the absence of much prior data, the above scheme could be used for a non-data driven time-parallelization.

CHAPTER 5

RESULT

The aim of our experiments is to evaluate the potential of data-driven time parallelization in soft-matter MD simulations. Our computing platform is a cluster belonging to our research group, which consists of 32 nodes, with each node containing two AMD Opteron 2.0 GHz 64-bit processors, 2 GB ECC DDR SDRAM memory, 64 KB L1 cache, 1024 KB L2 cache, running Red Hat Linux, Kernel 2.6.9. The nodes are connected through Gigabit Ethernet. The MPI implementation LAM was used with gcc compilers for our C code. The simulation setups are briefly described as follows. GROMACS takes a PDB (protein data bank) file of TI 127 structure. A total of $9,525$ tip4p water molecules are added to $5X5X5$ box under NVT condition at temperature 400 K. The springs are attached the first and last carbon atom under force constant of $400kJ/(mol \times nm^2)$. The algorithm applied is discussed in section 3.4. The complete GROMACS settings are shown in appendix A and appendix B. We can see speedup is almost ideal up to 12 processors. The loss in speed was only due to the overheads of communication, and reading or writing the base simulation data to disk.

5.1 Rupture Force

Dudko *et al.* [69] analyzed the relationship between force and AFM pulling rate. There are three competing theories. The simplest one, the phenomenological theory [70] assumes the rupture forces scales to the exponential of applied force according to Bell's formula [71].

$$k(F) = k_0 e^{fx^{\ddagger}} \tag{5.1}$$

For constant pulling speed where $F(t) = KVt$ the rupture force is predicted to grow proportionally to the logarithm of the pulling speed [72], $\langle F \rangle \sim \ln V$. Hummer and Szabo [73] proposed idea of applying Kramers theory of diffusive barrier to predict $\langle F \rangle \sim (\ln V)^{1/2}$.

Dudko *et al.* [69] theory predicts

$$\langle F \rangle \sim (\ln V)^{2/3} . \tag{5.2}$$

Rupture forces under different pulling rates yields two important constant properties of the protein: k_u (unfolding rate constant) and x_u (distance from folded state to transition state). k_u and x_u provide a simplified interpretation of a particular protein's energy landscape and allows one to compare mechanical strength of different proteins.

We can validate the correctness of the time parallel runs by verifying that the output rupture forces have the expected range of values, and that they occur at correct points in time. Figure 5.1 shows the rupture forces from our time parallelized code are approximately the log of the their pulling rate.

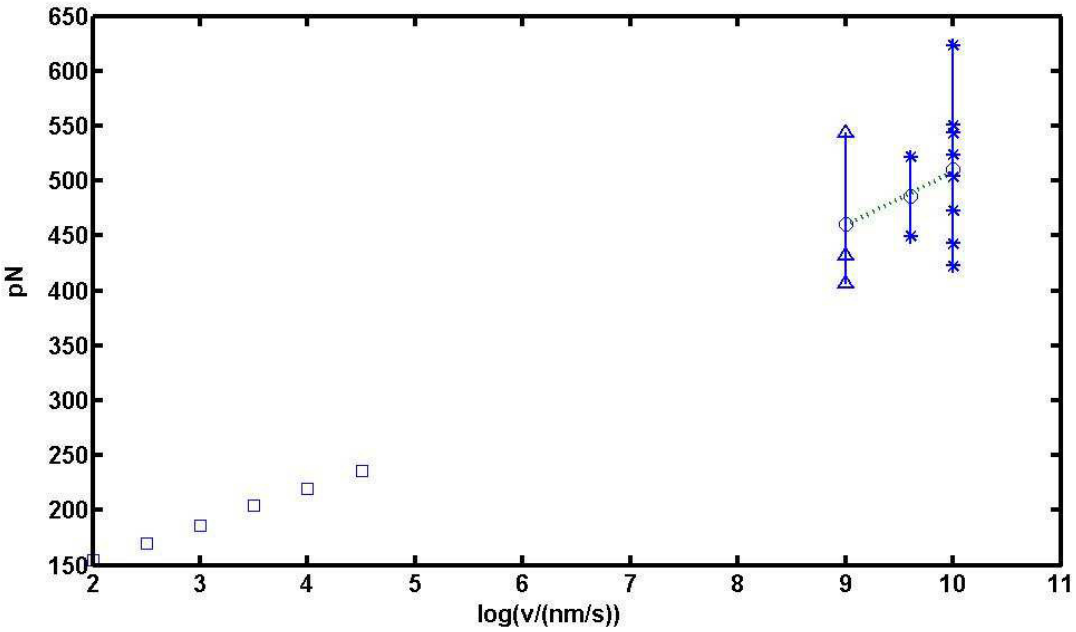

Figure 5.1: *Squares are rupture forces obtained from experiments [20]. Triangles are rupture forces obtained from our time parallelized code. Asterisks are rupture forces from spatially parallelized GROMACS code. All circles are the means. The dashed line shows the linear least square best fit line, based on data from three time-parallel and ten spatially-parallel runs.*

5.2 Validation

We next show results that validate the correctness of our approach in the problem considered. At first, we completed ten fast pulling runs at pulling rate of 10 m/s. Then, we used those fast pulling results to predict a slow pulling run at 1 m/s. Figure 5.2 compares the force extension profiles obtained from GROMACS spatially parallelized code and our time parallelized code. Both our curves shows a transition (inferred from the peak) at similar points in time. The solid line on Fig 5.3 shows the RMSD between GROMACS spatially parallelized run and our time parallelized run. The dashed line on fig. 5.3 shows the RMSD between two GROMACS runs of same AFM pulling rate but different seeds to the random number generator used in the thermostat. Those two lines are similar, and so the differences between a time-parallel run and the exact trajectory are similar to the difference obtained using a different random number sequence. Furthermore, fig. 5.1 shows that the statistics obtained are also similar to that with conventional code.

5.3 Speedup

Figure 5.4 shows the speedup results from the GROMACS spatially parallelized code. The maximum flop rate is computed to be 13.684 GFlops on 16 processors. The efficiency diminishes dramatically as the number of processors increases. In fact, spatially parallelized code on 24 processors performed worse than on 16 processors.

Next, Figure 5.5 shows the speedup results of our data-driven time parallelized code. The results show our time parallelized code can scale well at least on 10 processors. The prediction is always sufficiently accurate, but minor errors during 10 processors cause a drop in efficiency to around 96%. With 16 processors, there are a set of prediction errors in the middle of simulation before and after peaks on force extension profile. Figure 5.4 shows that spatial parallelization cannot scale beyond a small number of processors, and time parallelization is useful in extending the scalability by combining it with spatial parallelization. The benefit from time parallelization is substantial.

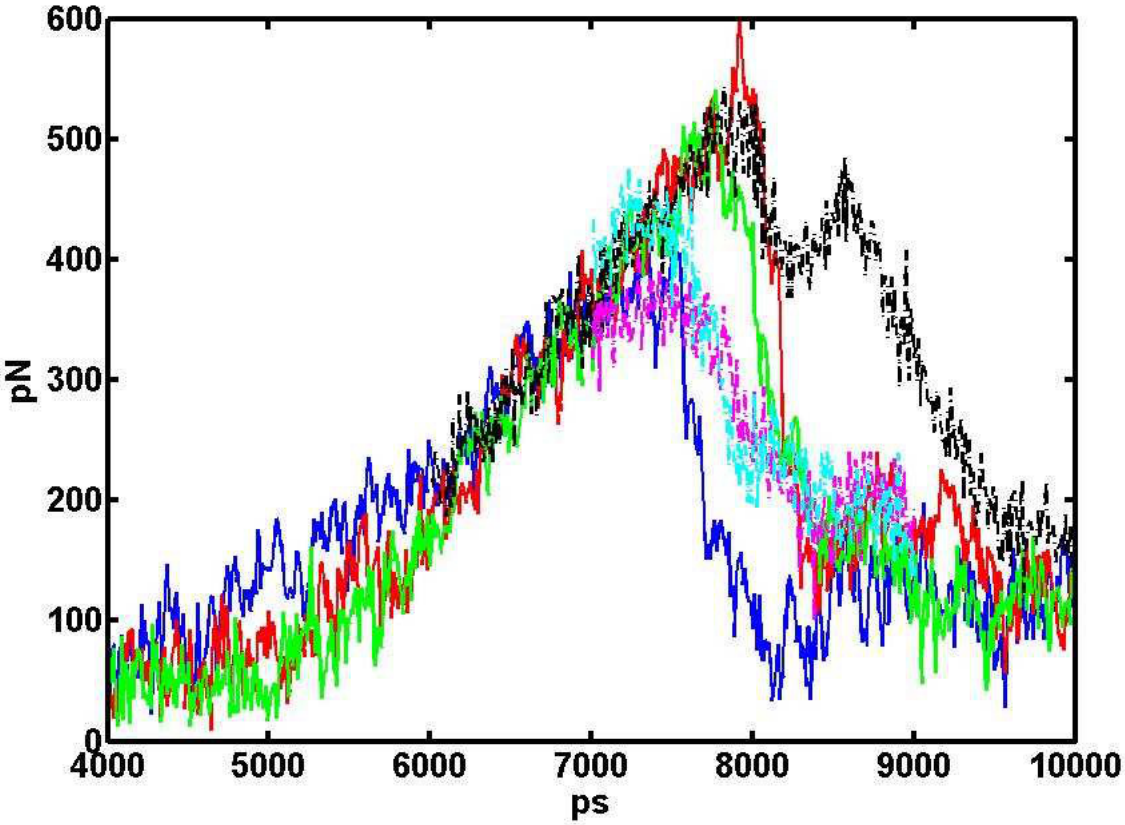

Figure 5.2: *The solid lines are the forces of the three spatial parallelized runs, and the dashed lines are the forces of the time parallelized runs.*

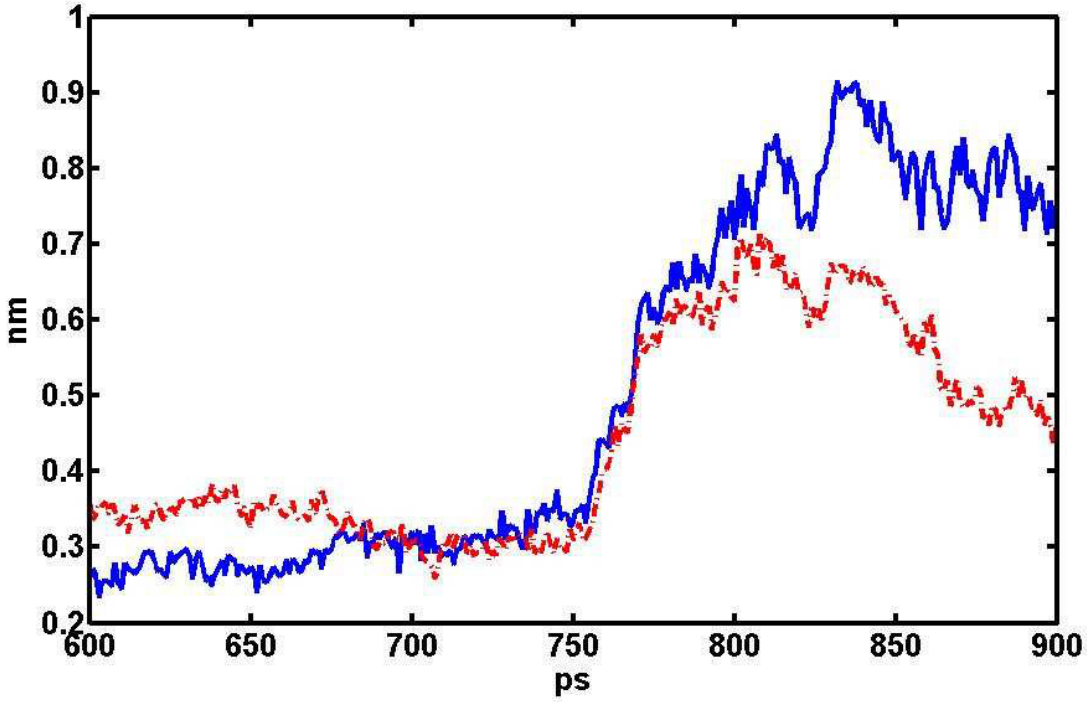

Figure 5.3: *The solid line shows the RMSD between GROMACS spatially parallelized run and our time parallelized run. The dashed line shows the RMSD between two GROMACS runs of same AFM pulling rate but different seeds to the random number generator used in the thermostat.*

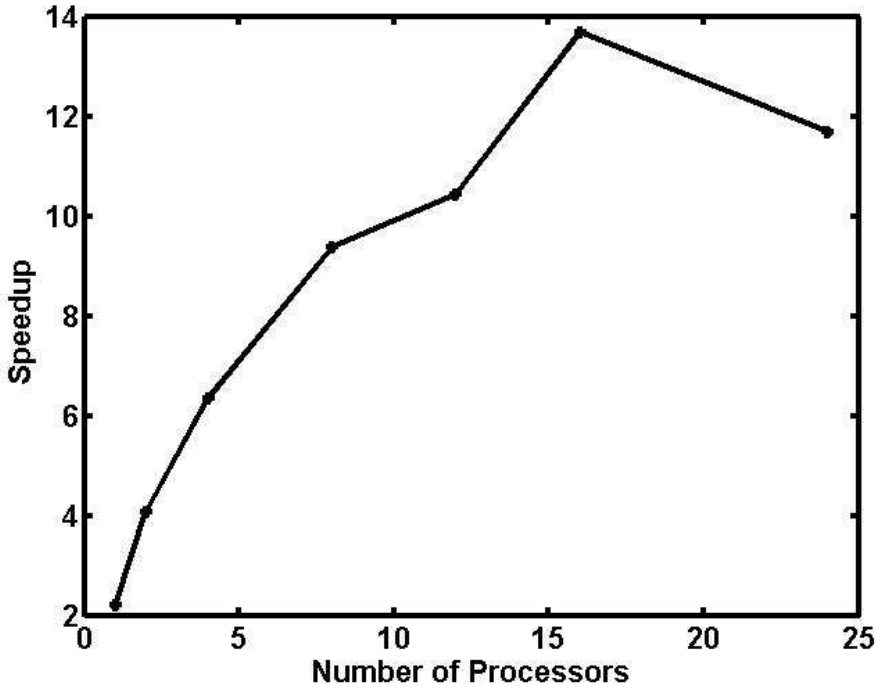

Figure 5.4: *Speedup curve. The line shows the speedup for spatially parallelized code. The flop rate is* 13.684 *GFlops on* 16 *processors, and* 11.686 *GFlops on* 24 *processors.*

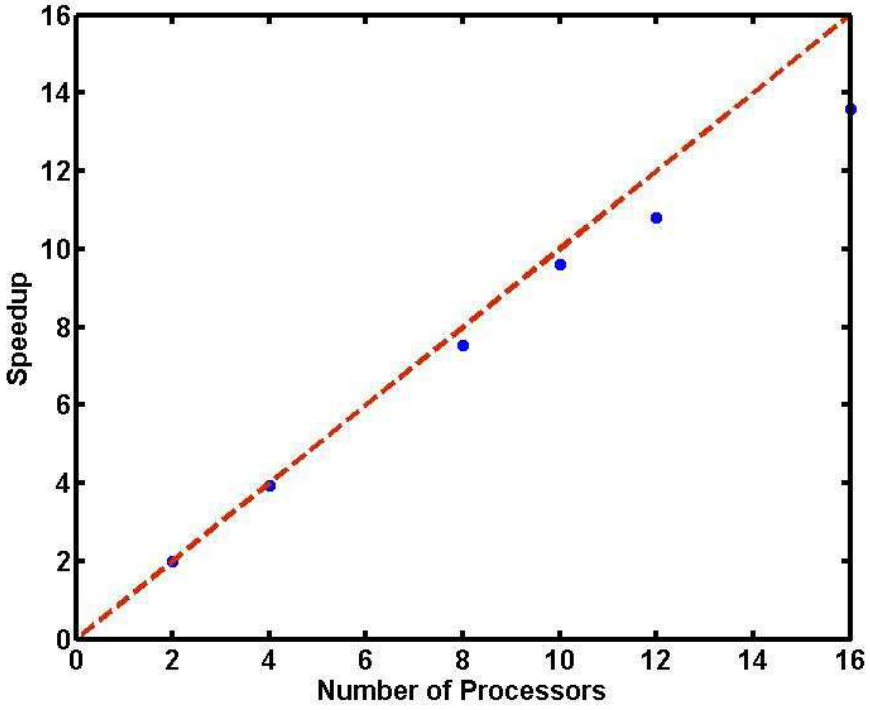

Figure 5.5: *Speedup curve. The dashed line shows the ideal speedup, and dots show the speedup for time parallelization.*

CHAPTER 6

CONCLUSION

We have showed the effectiveness of data-driven time parallelization in a practical biological soft-matter application – AFM pulling in protein folding. Since time parallelization works on the order of 10 processors, if we combine it with spatial parallelization, then we can extend the scalability of parallelization by an order of magnitude. Better prediction strategies might yield even better scalability. The MD applications considered in detail will, in themselves, have a major impact, by enabling better understanding of protein folding. Furthermore, the general methods developed can be used for a larger class of problems, since we address the common problem of scalability.

6.1 Limitations of Current Work

The work had the following limitations. (1) Somewhat ad-hoc criteria were used for the error thresholds. We explained in section 3.4 some conditions the prediction scheme should satisfy, in order to ensure that the errors are acceptable. Some other criteria such as calculating the number of H-bonds breaking apart might be more effective. (2) The most important limitation is that the scheme is not time-reversible and symplectic, which are believed necessary in soft-matter simulations. While comparisons with exact simulations showed that the results were acceptable, and this was sufficient to demonstrate the potential of our approach to soft-matter applications, if we are to use this technique in production runs, where exact results will not be available to compare against, then we need to ensure that the prediction mechanism does not violate time-reversibility and the symplectic property. (3) We would like to reduce the number of solvent water molecules in simulations. Section 1.3 mentioned implicit solvent method as good example to further improve overall performance.

6.2 Future work

Some of the future works as follows. Biological systems are known to have high dynamical Lyapunov exponents. So, initially close trajectories diverge rapidly, making determination of exact trajectories over long time impossible. Determining trajectories that are "correct" in a statistical sense is more important than long-term accuracy of a single trajectory. Statistical correctness can be produced by using so-called geometric integrators, which have the same global properties as the real dynamics, such as time-reversibility, energy conservation, and the symplectic property, that is, incompressible phase space flow. These methods are believed to exhibit the shadowing property, which means that they follow the exact trajectory of an approximation to the force field used. Since the force fields used are themselves approximations, time reversible symplectic integrators are considered accurate. We need to develop prediction schemes that do not violate the time reversibility and symplectic property of the integrator.

We also want to combine time parallelization with spatial parallelization – instead of one processor computing for one time interval, a group of processors, that distribute the atoms across the group, can be used to simulate each time interval. This will yield a code that improves on the best available conventional code. Lastly, we want to develop better criteria for verifying the accuracy of predictions.

APPENDIX A

Gromacs Options

*.mdp: allows the user to set up specific parameters for all the MD calculations that Gromacs performs.

A.1 MDP FILE

; VARIOUS PREPROCESSING OPTIONS
title = Short Dynamics
cpp = /lib/cpp
include =
define =
; RUN CONTROL PARAMETERS
integrator = md
; Start time and timestep in ps
tinit = 0
dt = 0.001
nsteps = 12000000
annealing_temp =
; GENERATE VELOCITIES FOR STARTUP RUN
gen-vel = yes
gen-temp = 300
gen-seed = 708582
; OPTIONS FOR BONDS
constraints = hbonds
; Type of constraint algorithm

constraint-algorithm = Lincs
; Do not constrain the start configuration
unconstrained-start = no
; Use successive overrelaxation to reduce the number of shake iterations
Shake-SOR = no
; Relative tolerance of shake
shake-tol = 1e-04
; Highest order in the expansion of the constraint coupling matrix
lincs-order = 2
; Number of iterations in the final step of LINCS. 1 is fine for
; normal simulations, but use 2 to conserve energy in NVE runs.
; For energy minimization with constraints it should be 4 to 8.
lincs-iter = 1
; Lincs will write a warning to the stderr if in one step a bond
; rotates over more degrees than
lincs-warnangle = 30
; Convert harmonic bonds to morse potentials
morse = no
; ENERGY GROUP EXCLUSIONS
; Pairs of energy groups for which all non-bonded interactions are excluded
energygrp_excl =
; NMR refinement stuff
; Distance restraints type: No, Simple or Ensemble
disre = No
; Force weighting of pairs in one distance restraint: Conservative or Equal
disre-weighting = Conservative
; Use sqrt of the time averaged times the instantaneous violation
disre-mixed = no
disre-fc = 1000
disre-tau = 0
; Output frequency for pair distances to energy file
nstdisreout = 100

; Orientation restraints: No or Yes
orire = no
; Orientation restraints force constant and tau for time averaging
orire-fc = 0
orire-tau = 0
orire-fitgrp =
; Output frequency for trace(SD) to energy file
nstorireout = 100
; Dihedral angle restraints: No, Simple or Ensemble
dihre = No
dihre-fc = 1000
dihre-tau = 0
; Output frequency for dihedral values to energy file
nstdihreout = 100
; Free energy control stuff
free-energy = no
init-lambda = 0
delta-lambda = 0
sc-alpha = 0
sc-sigma = 0.3
; Non-equilibrium MD stuff
acc-grps =
accelerate =
freezegrps =
freezedim =
cos-acceleration = 0
; Electric fields
; Format is number of terms (int) and for all terms an amplitude (real)
; and a phase angle (real)
E-x =
E-xt =
E-y =

E-yt =

E-z =

E-zt =

; User defined thingies

user1-grps =

user2-grps =

userint1 = 0

userint2 = 0

userint3 = 0

userint4 = 0

userreal1 = 0

userreal2 = 0

userreal3 = 0

userreal4 = 0

APPENDIX B

Gromacs Pulling options

*.ppa file: allows user to setup specific parameters for AFM pulling options.

B.1 PPA File

; GENERAL

verbose = yes

skip steps = 1000

; Runtype: afm, constraint, umbrella

runtype = afm

; Number of pull groups

ngroups = 2

; Groups to be pulled

group_1 = a_1

group_2 = a_1257

; The group for the reaction force

reference_group =

; Weights for all atoms in each group (default all 1)

weights_1 =

weights_2 =

reference weights =

; Ref. type: com, com_t0, dynamic, dynamic_t0

reftype = com

; Use running average for reflag steps for com calculation

reflag = 1

; Select components for the pull vector. default: Y Y Y

pull_dim = Y Y Y

; AFM OPTIONS

; Pull rates in nm/ps

afm_rate1 = 0.0005

afm_rate2 = -0.0005

; Force constants in $kJ/(mol * nm^2)$

afm_k1 = 400.0

afm_k2 = 400.0

; Directions

afm_dir1 = 1.0 0.0 0.0

afm_dir2 = 1.0 0.0 0.0

; Initial spring positions

afm_init1 = 7.361 3.020 2.036

afm_init2 = 9.651 1.551 2.529

REFERENCES

[1] R. H. Austin, K. W. Beeson, L. Eisenstein, H. Frauenfelder, and I. C. Gunsalus. Dynamics of ligand binding to myoglobin. *Biochemistry*, 14:5355–5373, 1975. 1.1

[2] H. Frauenfelder, G. A. Petsko, and D. Tsernoglou. Temperature-dependent X-ray diffraction as a probe of protein structural dynamics. *Nature*, 280:558–563, 1979. 1.1

[3] H. Frauenfelder, S. G. Sligar, and P. G. Wolynes. The energy landscapes and motions of proteins. *Science*, 254(5038):1598–1603, 1991. 1.1

[4] F. Parak. Physical aspects of protein dynamics. *Reports Prog. Phys.*, 66:103–129, 2003. 1.1

[5] L. Kullman, P. A. Gurnev, M. Winterhalter, and S. M. Bezrukov. Functional subconformations in protein folding: Evidence from single-channel experiments. *Phys. Rev. Lett.*, 96(3):038101, 2006. 1.1

[6] J. Brujic, R. I. Hermans, K. A. Walther, and J. M. Fernandez. Single-molecule force spectroscopy reveals signatures of glassy dynamics in the energy landscape of Ubiquitin. *Nature Phys.*, 2(4):282–286, 2006. 1.1

[7] Peter E. Wright and H. Jane Dyson. Intrinsically unstructured proteins: Re-assessing the protein structure-function paradigm. *J. Mol. Biol.*, 293:321–331, 1999. 1.1

[8] Peter Tompa. The interplay between structure and function in intrinsically unstructured proteins. *FEBS Lett.*, 579:3346–3354, 2005. 1.1

[9] Computational science: Ensuring America's competitiveness, May 2005. Report of the President's Information Technology Advisory Committee (available at: http://www.nitrd.gov/pitac/reports/20050609_computational/computational.pdf). 1.2, 2.1

[10] W. C. Still, A. Tempczyk, R. C. Hawley, and T. Hendrickson. Semianalytical treatment of solvation for molecular mechanics and dynamics. *J. Am. Chem. Soc.*, 112:6127–6129, 1990. 1.3

[11] M. Schaefer and C. Froemmel. A precise analytical method for calculating the electrostatic energy of macromolecules in aqueous-solution. *J. Mol. Biol.*, 216:1045–1066, 1990. 1.3

[12] G. D. Hawkins, C. J. Cramer, and D. G. Truhlar. Parameterized models of aqueous free energies of solvation based on pairwise descreening ofsolute atomic charges from a dielectric medium. *J. Phys. Chem.*, 100:19824–19839, 1996. 1.3

[13] A. Onufriev, D. Bashford, and D. A. Case. Modification of the generalized born model suitable for macromolecules. *J. Phys. Chem. B*, 104:3712–3720, 2000. 1.3

[14] M. S. Lee, M. Feig, F. R. Salsbury Jr., and C. L. Brooks III. New analytic approximation to the standard molecular volume definition and its application to generalized born calculations. *J. Comp. Chem.*, 24:1348–1356, 2003. 1.3

[15] E. Gallicchio and R. M. Levy. AGBNP: An analytic implicit solvent model suitable for molecular dynamics simulations and high-resolution modeling. *J. Comp. Chem.*, 25:479–499, 2004. 1.3

[16] S. Kumar, C. Huang, G. Almasi, and L. V. Kale. Achieving strong scaling with NAMD on Blue Gene/L. In *Proceedings of IPDPS*. IEEE, 2006. 1.1, 4.2

[17] B. G. Fitch et. al. Blue matter: Strong scaling of molecular dynamics on Blue Gene/L. Technical Report RC23688, IBM Research, 2005. 1.1, 4.2

[18] SCaLeS, the DOE office of science workshop on the science case for large-scale simulation (www.pnl.gov/scales), 2003. 2.1

[19] Daan Frenkel and Berend Smit. *Understanding Molecular Simulation.* Academic Press, 2002. 2.1, 3.3

[20] R. B. Best and J. Clarke. What can atomic force microscopy tell us about protein folding? *Chem. Commun.*, pages 183–192, 2002. 2.2.1, 2.2.2, 5.1

[21] Robert B. Best, David J. Brockwell, Jose L. Toca-Herrera, Anthony W. Blake, D. Alastair Smith, Sheena E. Radford, and Jane Clarke. Force mode atomic force microscopy as a tool for protein folding studies. *Analytica Chimica Acta*, 479:87–105, 2002. 2.2.2

[22] R. Rounsevell, J. R. Forman, and J. Clarke. Atomic force microscopy: Mechanical unfolding of proteins. *Methods*, 34:100–111, 2004. 2.2.2

[23] Jane Clarke and Gideon Schreiber. Folding and binding - new technologies and new perspectives. *Current Opinion in Structural Biology 2003*, 13:71–74, 2003. 2.2.2

[24] Susan B. Fowler, Robert B. Best, Jose L. Toca-Herrera, Trevor J. Rutherford, Annette Steward, Emanuele Paci, Martin Karplus, and Jane Clarke. Mechanical unfolding of a Titin IG domain: Structure of unfolding intermediate revealed by combining AFM, molecular dynamics simulations, NMR and protein engineering. *J. Mol. Biol.*, 322:841–849, 2002. 2.2.2

[25] S. Nosé. A unified formulation of the constant temperature molecular dynamics methods. *J. Chem. Phys.*, 81(1):511–519, 1984. 2.2.2

[26] William G. Hoover. Canonical dynmaics: Equilibrium phse-space distributions. *Phys. Rev. A*, 31(3):1695–1697, 1984. 2.2.2

[27] S. Nosé. Contant temperature molecular dynamics methods. *Prog. Theor. Phys. Supp.*, 103:1–46, 1991. 2.2.2

[28] Hans C. Andersen. Molecular dynamics simulations at constant pressure and/or temperature. *J. Chem. Phys.*, 72(4):2384–2393, 1980. 2.2.2

[29] N. D. Socci, J. N. Onuchic, and P. G. Wolynes. Stretching Lattice models of protein folding. *Proc. Nat'l Acad. Sci. USA*, 96(5):2031–2035, 1999. 2.2.2

[30] D. J. Lacks. Energy landscape distortions and the mechanical unfolding of proteins. *Biophys. J.*, 88:3494–3501, 2005. 2.2.2

[31] P. M. Williams, S. B. Fowler, R. B. Best, J. L. Toca-Herrera, K. A. Scott, A. Steward, and J. Clarke. Hidden complexicty in the machanical properties of Titin. *Nature*, 422(6930):446–449, 2003. 2.2.2

[32] A. Srinivasan and N. Chandra. Latency tolerance through parallelization of time in scientific applications. *Parallel Computing*, 31:777–796, 2005. 3.1

[33] A. Srinivasan, Y. Yu, and N. Chandra. Scalable parallelization of molecular dynamics simulations in nano mechanics, through time parallelization. Technical Report TR-050426, Department of Computer Science, Florida State University, 2005. 3.1, 3.1

[34] A. Srinivasan, Y. Yu, and N. Chandra. Application of reduced order modeling to time parallelization. In *Proceedings of HiPC 2005, Lecture Notes in Computer Science*, volume 3769, pages 106–117. Springer-Verlag, 2005. 3.1

[35] E. Lelarasmee, A.E. Ruehli, and A.L. Sangiovanni-Vincentelli. The waveform relaxation method for time-domain analysis of large scale integrated circuits. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 1:131–145, 1982. 4.1

[36] C. W. Gear. Waveform methods for space and time parallelism. *Journal of Computational and Applied Mathematics*, 38:137–147, 1991. 4.1

[37] B. Leimkuhler. Timestep acceleration of waveform relaxation. *SIAM Journal on Numerical Analysis*, 35:31–50, 1998. 4.1

[38] R. Jeltsch and B. Pohl. Waveform relaxation with overlapping splittings. *SIAM Journal on Scientific Computing*, 16:40–49, 1995. 4.1

[39] L. Baffico, S. Bernard, Y. Maday, G. Turinici, and G. Zerah. Parallel-in-time molecular-dynamics simulations. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 66:57701–57704, 2002. 4.1

[40] Y. Maday and G. Turinici. Parallel in time algorithms for quantum control: Parareal time discretization scheme. *International Journal of Quantum Chemistry*, 93:223–238, 2003. 4.1

[41] J. C. Phillips, G. Zheng, S. Kumar, and L. V. Kale. NAMD: Biomolecular simulations on thousands of processors. In *Proceedings of SC2002*. IEEE, 2002. 4.2

[42] D. Srivastava and S. T. Bernard. Molecular dynamics simulation of large-scale Carbon nanotubes on a shared-memory architecture. In *Proceedings of the IEEE/ACM SC1997 Conference*. IEEE Computer Society, 1997. 4.2

[43] J. Kolhe, U. Chandra, S. Namilae, A. Srinivasan, and N. Chandra. Parallel simulation of Carbon nanotube based composites. In L. Bougé and V. K. Prasanna, editors, *Proceedings of the 11 th International Conference on High Performance Computing (HiPC), Lecture Notes in Computer Science – 3296*, pages 211–221. Springer-Verlag, 2004. 4.2

[44] G. M. Crippen and H. A. Scheraga. Minimization of polypeptide energy, VIII. Application of the deflation technique to a dipeptide. *Proc. Nat'l Acad. Sci. USA*, 64:42–49, 1969. 4.3

[45] L. Piela, J. Kostrowicki, and H. A. Scheraga. The multiple-minima problem in the conformational analysis of molecules. Deformation of the potential energy hypersurface by the diffusion equation method. *J. Phys. Chem.*, 93:3339–3346, 1989. 4.3

[46] R. Swendsen and J. Wang. *Phys. Rev. Lett.*, 57:2607–2609, 1986. 4.3

[47] E. Marinari and G. Parisi. *Europhys. Lett.*, 19:451–458, 1992. 4.3

[48] C. Geyer and E. Thompson. *J. Am. Stat. Assoc.*, 90:909–920, 1995. 4.3

[49] K. Hukushima and K. Nemoto. *J. Phys. Soc. Japan*, 65:1604–1608, 1996. 4.3

[50] B. A. Berg and T. Neuhaus. The multicanonical ensemble: A new approach to simulate first-order phase transitions. *Phys. Rev. Lett.*, 68:9–12, 1992. 4.3

[51] U. H. E. Hansmann. Parallel tempering algorithm for conformational studies of biological molecules. *Chem. Phys. Lett.*, 281:140–150, 1997. 4.3

[52] M. R. Sorensen and A. F. Voter. Temperature-accelerated dynamics dynamics for the simulation of infrequent events. *J. Chem. Phys.*, 112:9599, 2000. 4.3

[53] A. F. Voter. A method for accelerating the molecular dynamics simulation of infrequent events. *J. Chem. Phys.*, 106:4665–4676, 1997. 4.3

[54] A. F. Voter, F. Montalenti, and T. C. Germann. Extending the time scale in atomistic simulation of materials. *Annual Review of Materials Research*, 22:321–346, 2002. 4.3

[55] F. Starrost and E. A. Carter. Modeling the full monty: Baring the nature of surfaces across time and space. *Surface Science*, 500:323–346, 2002. 4.3, 4.3

[56] Theory and modeling in nanoscience, May 2002. Report of the May 10-11, 2002 Workshop conducted by the basic energy sciences and advanced scientific computing advisory committees to the Office of Science, Department of Energy. 4.3

[57] A. F. Voter. Parallel replica method for dynamics of infrequent events. *Phys. Rev. B*, 57(22):R13985–R13988, 1998. 4.3

[58] Vijay S. Pande. Folding@home. http://folding.stanford.edu/, 2000. 4.3

[59] G. Jayachandran, V. Vishal, and V. S. Pande. Using massively parallel simulation and Markovian models to study protein folding: Examining the dynamics of the villin headpiece. *J. Chem. Phys.*, 124:164902, 2006. 4.3

[60] S. P. Elmer, S. Park, and V. S. Pande. Foldamer dynamics expressed via Markov state models. i. explicit solvent molecular-dynamics simulations in Acetonitirle, Chloroform, Methanol, and Water. *J. Chem. Phys.*, 123(11):114902, 2005. 4.3

[61] S. P. Elmer, S. Park, and V. S. Pande. Foldamer dynamics expressed via Markov state models. ii. State Space Decomposition. *J. Chem. Phys.*, 123(11):114903, 2005. 4.3

[62] W. C. Swope, J. D. Pitera, and F. Suits. Describing protein folding kinetics by molecular dynamics simulations. 1. theory. *J. Phys. Chem. B*, 108:6571–6581, 2004. 4.3

[63] W. C. Swope, J. D. Pitera, F. Suits, M. Pitman, M. Eleftheriou, B. G. Fitch, R. S. Germain, A. Rayshubski, T. J. C. Ward, Y. Zhestkov, and R. Zhou. Describing protein folding kinetics by molecular dynamics simulations. 2. example applications to Alanine Dipeptide and a β-Hairpin Peptide. *J. Phys. Chem. B*, 108:6582–6594, 2004. 4.3

[64] A. Nakano, P. Vashishta, and R. K. Kalia. Parallel multiple-time-step molecular dynamics with three-body interaction. *Comput. Phys. Commun.*, 77:303–312, 1993. 4.3

[65] C. W. Gear and D. R. Wells. Multirate linear multistep methods. *BIT*, 24:484–502, 1984. 4.3

[66] T.C. Bishop, R.D. Skeel, and K. Schulten. Difficulties with with multiple time stepping and fast multipole algorithm in molecular dynamics. *Journal of Computational Chemistry*, 18:1785–1791, 1997. 4.3

[67] L. Chen, P. G. Debenedetti, C. W. Gear, and I. G. Kevrekidis. From molecular dynamics to coarse self-similar solutions: a simple example using equation-free computations. *Journal of Non-Newtonian Fluid Mechanics*, 120:215–223, 2004. 4.3

[68] I. G. Kevrekidis, C. W. Gear, and G. Hummer. Equation-free: The computer-assisted analysis of complex multiscale systems. *AIChE Journal*, 507:1346–1355, 2004. 4.3

[69] Olga K. Dudko, Gerhard Hummer, and Attila Szabo. Intrinsic rates and activation free energies from single-molecule pulling experiments. *Physical Review Letters*, 96:108101, 2006. 5.1, 5.1

[70] Evan Evans, D. Berk, and A. Leung. Detachment of Agglutinin-bonded red blood cells. i. forces to rupture molecular-point attachments. *Biophys*, 59:838, 1991. 5.1

[71] George I. Bell. Models for the specific adhesion of cells to cells. *Science*, 200:618–627, 1978. 5.1

[72] Evan Evans and Ken Ritchie. Dynamic strength of molecular adhesion bonds. *Biophys*, 72:1541–1555, 1997. 5.1

[73] Gerhard Hummer and Attila Szabo. Kinetics from nonequilibrium single-molecule pulling experiments. *Biophys*, 85:5, 2003. 5.1

BIOGRAPHICAL SKETCH

Lei Ji

Lei "Leonardo" Ji was born in Nanjing, China. Lei is a naturalized US citizen. In the spring 2003 he completed his Bachelors degree in Computer Science at Louisiana Tech University. Under the advisement of Professor Srinivasan and Professor Nymeyer, he obtained his Master degree in summer of 2006 from Department of Computer Science at Florida State University. Lei's research interests include parallel computing, scientific visualization, computer graphics, numerical algorithms, and computational biology. Lei lives in Tallahassee, Florida.