

Data-Driven Time-Parallelization in the AFM Simulation of Proteins

L. Ji¹, H. Nymeyer², A. Srinivasan¹, and Y. Yu¹

¹Florida State University
Dept. of Computer Science
Tallahassee, FL 32306 USA
{ji, asriniva, yu}@cs.fsu.edu

²Florida State University
Dept. of Chemistry and Biochemistry
Tallahassee, FL 32306 USA
hny Meyer@fsu.edu

Abstract

Molecular Dynamics is a popular technique to simulate the behavior of physical systems, with resolution at the atomic scale. One of its limitations is that an enormous computational effort is required to simulate to realistic time spans. Conventional parallelization strategies have limited effectiveness in dealing with this difficulty. We recently introduced a more scalable approach to parallelization, where data from prior, related, simulations are used to parallelize a simulation in the time domain. We demonstrated its effectiveness in nano-mechanics simulations. In this paper, we develop our approach so that it can be used in a soft-matter application involving the atomic force microscopy simulation of proteins. We obtain an order of magnitude improvement in performance when we combine time parallelization with conventional parallelization. The significance of this work lies in demonstrating the promise of data-driven time parallelization in soft-matter applications, which are more challenging than the hard-matter applications considered earlier.

1. Introduction

Molecular Dynamics (MD) finds widespread use in atomistic simulations in Chemistry, Materials, and Biology. For example, MD provides an excellent method to identify individual conformational states of proteins, and transitions between different conformational states. In MD, forces on atoms due to interactions with other atoms are computed using certain empirical force fields. Once forces are computed, Newton's laws of motion are used, almost always with an explicit time integration scheme, to determine the trajectory of the system. 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.

A limitation of MD is the short time span that can be simulated. Large-scale protein conformational changes, such as folding and allosteric transitions, typically occur in the millisecond time scale. MD, on the other hand, can access time scales of the order of only a microsecond, for the following reason. High frequency motions of atoms limit the time step size of the numerical differential equation solver to about a femto second (10^{-15} s). Consequently, 10^{12} time-steps are required to simulate to a millisecond of real time. This requires an enormous computational effort, as illustrated later. In fact, this limitation in the time scale accessible through MD has been identified as one of important challenges in computational biology [8] and computational nano-materials [11].

Massive parallelism can help deal with the high computational effort to a certain extent. The difficulty is in obtaining high efficiencies with current parallelization strategies, because they are more effective in dealing with large state space, than with long time scales. For example, conventional spatial decomposition methods (including atom and force decompositions) yield high efficiencies only when the time per iteration is of the order of ten milliseconds, as shown in fig. 1, on the most scalable codes. These do not scale to better than the order of a millisecond per iteration, even if we accept low efficiencies. At granularities of 1 – 10 milliseconds per iteration, it will take 30 – 300 years to simulate to a millisecond of real time, using a femto-second time step. This problem is magnified on codes that are less scalable.

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 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 de-

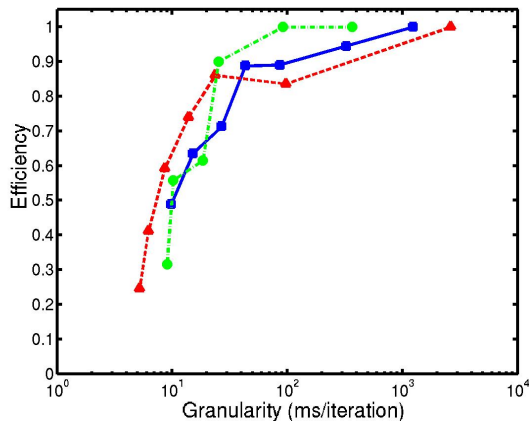


Figure 1. Scaling results on an IBM Blue Gene/L. The solid and dashed lines show results for NAMD, based on data from [7]. The solid line is for a 327,000 atom ATPase PME simulation, and the dashed line for the 92,000 atom ApoA1 PME simulation. The dash-dotted line shows results for IBM’s Blue Matter code on a 43,000 atom Rhodopsin system, based on data from [4].

composition is of limited use for these problems, because the state space is even smaller, leading to fine granularities on smaller numbers of processors.

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 paper, we demonstrate the feasibility of this approach on an important soft matter application in computational biology.

The basic idea behind our approach is to have each processor simulate a different time interval, as illustrated in fig. 2. The difficulty is that, in an initial value problem, a processor does not know the initial state for its 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 prior results 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 MD computation, to predict the starting state for each processor. The MD computations are then used, again, to verify if the predictions were correct, and to learn the relationship with prior simulations dynamically, 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. The load is also well balanced. This method can be combined with spatial decomposition to increase the scalability over existing methods. In fact, our results demonstrate an improvement by an order of magnitude through such an approach.

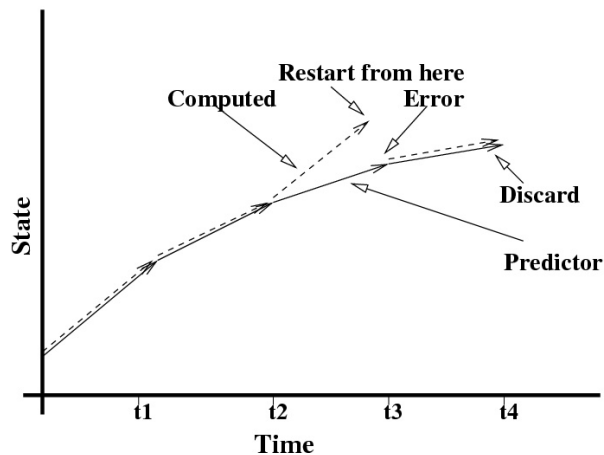


Figure 2. Schematic of time parallelization.

The outline of the rest of this paper is as follows. In § 2, we present background information on MD and on *Atomic Force Microscopy* (AFM) simulations in biology. We then explain our data-driven time parallelization approach, and the unique challenges of applying it to soft-matter applications, in § 3. We present experimental results in § 4. We summarize our conclusions in § 5.

2. Application

2.1. Molecular Dynamics

MD is a computational simulation method that determines the position and velocity of each atom that is contained in a computational cell subjected to external boundary conditions (force, pressure, temperature, or velocities). At any point in time, an empirical force field is used to compute the forces on all atoms. Then Newton’s laws of motion are used to determine the positions and velocities of all atoms at the next point in time, using an explicit numerical differential equation solver. This process is repeated. The time step size is required to be around a femto second (10^{-15} s) to ensure stability. Realistic simulations need to

be performed to microseconds or milliseconds time spans. Thus $10^9 - 10^{12}$ iterations are required, which needs enormous computations effort, even if the physical system is small.

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 further in section 3.3.

2.2. Atomic Force Microscopy on Titin

The biological system we consider is *Titin*. It 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. Therefore, its properties, such as muscle elasticity, can be determined by studying the properties of each individual domain using protein-unfolding experiments.

In Atomic Force Microscopy of Titin, mechanical force is applied to its two ends. This 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 force is applied, as shown later in fig. 8. AFM is becoming an important biophysical technique, to study protein unfolding events at atomic resolution.

AFM measurements are limited in their pulling speed, and the use of a finite pulling speed directly affects the mechanism of folding [12]. 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^{-5} m/s. Our aim is to use such high pulling rate results to time-parallelize lower pulling rate experiments. We use 10 m/s runs to time-parallelize 1 m/s computations. Simulating at even lower speeds requires more computational resources than we have available.

3. Data-Driven Time Parallelization

In this section, we first describe the data-driven time parallelization approach. Many strategies can be implemented within the framework of this approach. We describe the approach used in our earlier hard-matter applications. We then describe the challenges of soft-matter applications, and then present the specific strategy used in the AFM application.

3.1. The Approach

Figure 2 illustrates our approach. Let us call a few iterations, say 1,000 or 10,000 time steps of a differential equation 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 MD computations on processor 1 lead to accurate results on that processor. In fig. 2, 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 state computed by processor 3 using MD at time t_3 , and we compute states for times t_4 , t_5 , t_6 , and t_7 in parallel.

Note the following: (i) Processor 1’s MD result is always correct, since it always starts from a state known to be accurate. So the computation always progresses. (ii) All processors must use a consistent prediction mechanism, so that the predictions for time t_i are identical on processors i and $i + 1$. (iii) A global communication operation (AllReduce call in Algorithm 1) is used to detect the smallest ranked processor to err. (iv) The initial state for processor 1 needs to be sent by another processor, except during the first phase of the computation. (v) If a 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.

3.2. Prior Work

Our prior work was on the tensile test of a Carbon Nanotube (CNT), where a CNT is pulled at one end, keeping the other end fixed. Conventional parallelization of the application we considered scales to only 2 – 3 processors (at a granularity under 10 ms per iteration). Our computations scaled to two to three orders of magnitude greater number of processors, with efficiencies typically over 90%. One sim-

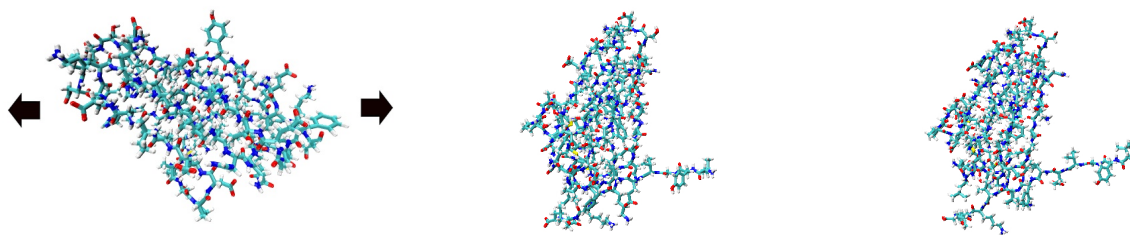


Figure 3. AFM pulling of TI 127 mutant. Left: Initial state. Center: State before a strand breaks apart. Right: State after a strand breaks apart.

ulation yielded a granularity of $13.5 \mu\text{s}$ per iteration, which we believe is the finest granularity achieved in classical MD computations. Prediction was based on first reducing the dimensionality by finding a low-dimensional subspace where changes happen slowly, making them easier to predict.

3.3. Challenges in Soft-Matter Applications

Soft-matter applications, as typically encountered in computational biology, are more difficult to time-parallelize than the hard-matter applications described above, for the following reasons. In hard-matter simulations, MD trajectories started under slightly different initial conditions (such as different velocities) or using different random number sequences, follow trajectories that are close to each other, at least for substantially long periods of time. In contrast, soft-matter simulations exhibit diffusive heterogeneous behavior. The trajectories are extremely sensitive to initial conditions. Two different simulations with slightly different initial conditions will diverge exponentially as time progresses. Similar divergence is caused by numerical errors in the differential equation solver. In fact, a single MD trajectory does not give much useful information in soft-matter simulations. Instead, one requires statistics from a large number of simulations.

3.4. Specific Approach in the Biological Problem

In this section, we present our implementation of the data-driven time-parallelization approach for AFM pulling of proteins (Algorithm 1). The primary difference from the CNT application is in prediction. Simple dimensionality reduction techniques are not effective for this application. Instead we use the actual data from previous simulations. As the simulation proceeds, we determine the prior simulation which the current simulation is behaving like, and use that prior simulation to predict the initial states.

During the verification phase, we have to choose some criteria to determine an acceptable error threshold. We have results of faster pulling runs, with different seeds to the random number generator used in the thermostat, and also different initial states (initial velocities of the atoms). We note the differences between simulations with different random number seeds. We set error thresholds for the difference between predicted state and computed state based on the above differences. That is, a prediction is considered sufficiently accurate if its difference with the computed state is similar to the difference observed if we had used a different random number sequence.

We use two metrics for the differences. One is the maximum difference in positions between the same atom in the two states, defined by $MAXD = \max_i d_i$, where d_i is the difference in positions of the i th atom in the two states. The other is the root mean square deviation, which is defined as $RMSD = \sqrt{\frac{1}{N} \sum_{i=1}^N d_i^2}$, where N is the number of atoms. In computing these differences, only the atoms in Titin were considered, and not the solvent atoms. Furthermore, before the states were compared, they were aligned to remove the effects of rigid body motion (rotation and translation), as is conventionally done with RMSD calculations.

Figure 4 shows the variation of RMSD and maximum difference with time, for two faster pulling runs using different seeds. Based on such results, we set the RMSD threshold to 0.2 nm and the maximum difference threshold to 1 nm in step 9 of algorithm 1.

3.5. Combining Space and Time Parallelization

Time and space parallelization can be combined, so that time parallelization is used to improve scalability over what is possible through just spatial parallelization. Instead of a single processor computing for one time interval, a group of processors computes for a time interval. We had mentioned the potential for such combination in earlier work. In the

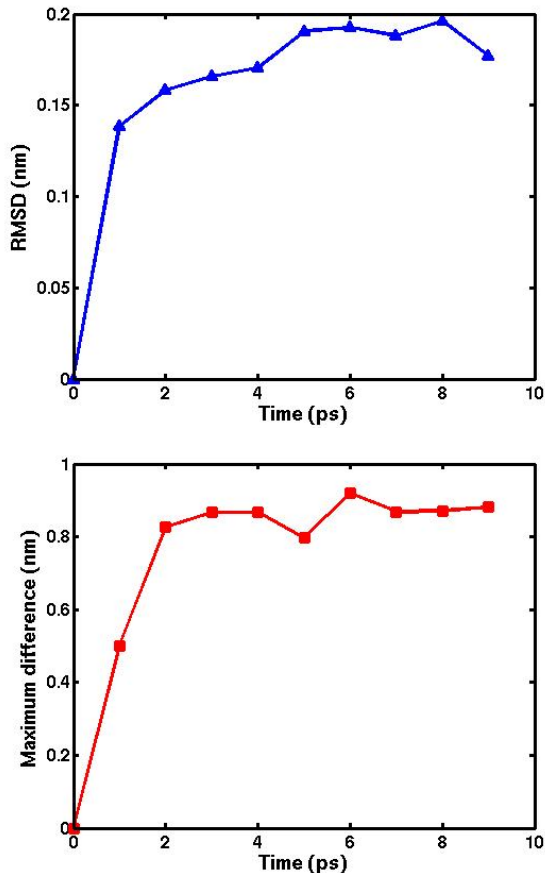


Figure 4. Difference between two trajectories that are identical, except that they use different seeds to the random number generator used in the thermostat.

current work, we actually implemented this feature. This required some changes to the GROMACS 3.3.1 [5] Molecular Dynamics code, which we used in our simulations. In particular, the code was not designed to be composed with itself or with other code. We modified it so that it can be composed with itself. We also need some additional tools provided by GROMACS, to remove solvent atoms, align Titin states through rigid body motion, and compute RMSD. It was difficult to compose these with the main program, and so we instead used the *system* call to call these programs (which are run sequentially and independently on one processor of each group). They interact with the main program through file I/O. While this is much slower than would be possible if they had been integrated with the main program, the overhead associated with this is relatively insignificant, because each time interval of 10,000 time steps requires several hours of computing time, even when spatially parallelized, because GROMACS does not scale well to very fine granularities.

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

```

1:  $t \leftarrow 0$ 
2: while  $t < m$  do
3:   for each processor  $i \in [1, \min(P, m - t)]$  do
4:     if  $i \neq 1$  then
5:        $S_{t+i-1} \leftarrow \text{Predict}(t + i - 1)$ 
6:     end if
7:      $S_{t+i} \leftarrow \text{Predict}(t + i)$ 
8:      $\hat{S}_{t+i} \leftarrow \text{AccurateMD}(\text{StartState} = S_{t+i-1}, \text{StartTime} = t + i - 1, \text{EndTime} = t + i)$ 
9:     if  $\text{RMSD}$  or  $\text{MAXD}$  too large( $\hat{S}_{t+i}, S_{t+i}$ ) then
10:       $\text{Next} \leftarrow i$ 
11:    else
12:       $\text{Next} \leftarrow P$ 
13:    end if
14:  end for
15:   $k \leftarrow \text{AllReduce}(\text{Next}, \text{min})$ 
16:   $S_{t+k}$  on processor 1  $\leftarrow \hat{S}_{t+k}$  computed by processor  $k$ .
17:  for each processor  $i \in [1, P]$  do
18:     $t \leftarrow t + k$ 
19:  end for
20: end while

```

4. Experimental Results

The aim of our experiments is to evaluate the potential of data-driven time parallelization in soft-matter MD simulations, in terms of scalability and accuracy.

Our primary computing platform is a 32-node dual-processor (but not dual-core) Opteron cluster with Gigabit Ethernet interconnect. The processors run at 2.0 GHz, have 2 GB ECC DDR SDRAM memory per node, 64 KB L1 cache, 1024 KB L2 cache, and run Red Hat Linux kernel 2.6.9. The MPI implementation used was LAM, and gcc was used for compilation.

Combined space-time parallelism was evaluated on the *Tungsten* Xeon cluster at NCSA. This cluster consists of Dell PowerEdge 1750 servers, with each node containing two Intel Xeon 3.2 GHz processors, 3 GB ECC DDR SDRAM memory, 512 KB L2 cache, 1 MB L3 cache, running Red Hat Linux. The file system used is Lustre. Myrinet 2000 and Gigabit Ethernet interconnects were available. We used the Myrinet interconnect. The ChaMPIon/Pro MPI implementation was used with the gcc compiler, and code compiled with '-O3' optimization flag set.

The simulation setup was as described below. We used the GROMACS software to perform the MD simulations. The input was a protein native structure *TI 127* (a 1TIT

Titin mutant with A-strand deleted) from the Protein Data Bank [9]. A total of 9,525 *TIP4* water molecules were added as solvent. Na^+ and Cl^- ions were added to maintain charge neutrality. The total number of atoms was around 40K. The simulations were performed under NVT conditions, with temperature kept at 400 K using a Langevin thermostat. Springs were attached to the first and last Carbon atoms under a force constant of $400kJ/(mol \times nm^2)$.

4.1. Speedup

Figure 5 shows that spatial parallelization does not scale beyond a small number of processors. The results show that the time parallel code can scale well to 16 processors. The prediction is usually sufficiently accurate, with minor errors, yielding efficiencies around 90% or higher on up to 10 processors. With 12 and 16 processors, there is a set of prediction errors before and after peaks on the force extension curve, which lowers the efficiency to the 75% – 80% range. The loss in efficiency is primarily due to prediction errors. The parallelization overheads are themselves relatively insignificant, because the time taken for each time interval is around 5 hours, while the overheads are in seconds.

Figure 6 compares spatial parallelization with combined time and spatial parallelization. The overheads of the time parallelization associated with prediction, verification, and I/O are insignificant, accounting for around 0.1% of the total time. Some of the loss in efficiency is due to prediction errors, as shown in Figure 5 (solid line). However, the main loss in efficiency is due to the spatially parallelized code itself having poor efficiency of 46.5% on 8 processors on the Xeon cluster.

A sequential run requires around 5000 hours to simulate to ten nanoseconds (we need to simulate to around 9 ns to be sure that the one of the peaks of the force-displacement code is observed). This would take close to a year of sequential computing time. In order to obtain speedup results faster on the spatially parallelized code, we simulated for a nanosecond to get the speedup results. We also performed complete spatially parallel runs on 16 processors each, which took a little less than a month of computing time per run. For the time parallel runs, a few complete runs were performed on 10 processors. Prediction errors typically occur closer to where the force-displacement curves peak, as expected. In order to determine the speedup, runs were started a little before the peak, and then simulated for 2 ns. This makes it feasible to compute the speedup on small numbers of processors too. However, the speedup over the whole run will likely be larger, because then we would include the initial region where prediction errors do not occur. The speedup

TIP4 is a water molecule model which has 2 Hydrogen atoms and 2 Oxygen atoms.

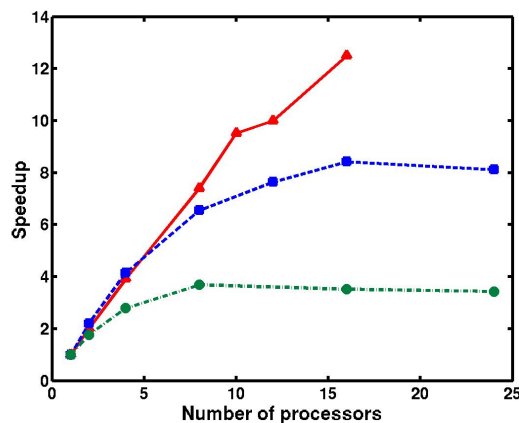


Figure 5. Speedup results. Time parallelization on the Opteron cluster (solid line), spatial parallelization on the Opteron cluster (dashed line), and spatial parallelization on the Xeon cluster (dash-dotted line).

for the combined space-time parallel code was computed as follows. From the purely time-parallel result described above, the number of iterations required for the loop in step 2 of algorithm 1 was computed, for each value of the number of processors, P . The combined parallel code used P groups of processors, where each group performed a spatially parallel computation on 8 processors. The simulation results (but not the timing results) are the same as a time parallel computation on P processors, but performed faster. We performed shorter runs to determine the time taken for one time interval of the combined time-space parallel code, and multiplied it by the number of iterations obtained through pure time parallelization, to determine the total time. As mentioned above, since the pure time parallelization results underestimate the speedup over the whole run, the combined results too probably underestimate the speedup over the entire run.

4.2. Validation

We next need to validate the correctness of the time parallelized runs. This is more difficult than for the hard matter computation, because a single MD trajectory is not meaningful. Instead, we are interested in the statistical average of a number of trajectories. We use the peak force, for the first peak, to validate the time-parallel results.

There are different theories to explain the variation of peak force with pulling speed, such as the following. The rupture force is predicted to grow proportionally with the logarithm of the pulling speed in [3]. Hummer and Szabo [6] predict that it will be proportional to $(\ln v)^{1/2}$, where v is the pulling speed. Dudko *et al.* [2] predict it

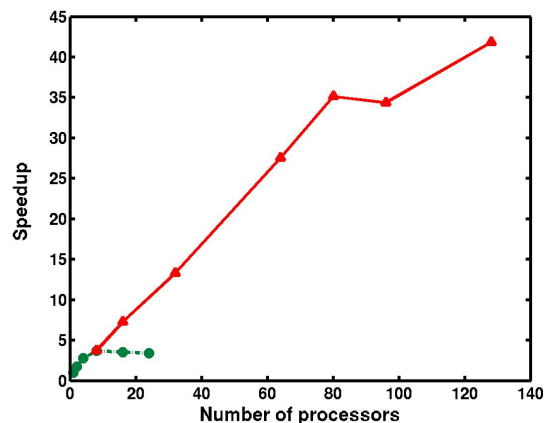


Figure 6. Speedup results. Combined time and spatial parallelization on the Xeon cluster, with each group being 8-way spatially parallelized (solid line) and spatial parallelization on the Xeon cluster (dash-dotted line).

to be proportional to $(\ln v)^{2/3}$. Peak rupture forces under different pulling rates yield two important constant properties of the protein: (i) the unfolding rate constant and (ii) the distance from folded state to transition state. These two properties provide a simplified interpretation of a particular protein’s energy landscape and allows one to compare mechanical strengths of different proteins.

We can validate the correctness of the time parallel runs by verifying that the output rupture forces have a range of values consistent with the spatially parallelized runs, and that they occur at correct points in time. Figure 7 shows that the rupture forces from our time parallelized code have a similar range as the spatially parallelized code, and that their averages too are close.

We also wished to validate the MD simulations, to see if their results are consistent with experiments, even though it is not directly related to time parallelization. It is difficult to do this directly, since the pulling speeds in the two cases differ by several orders of magnitude. Instead, we extrapolate experimental results, and see that the MD results are in the range predicted by a logarithmic variation with pulling speed. While the exact values appear a little above the extrapolation line, they are in an acceptable range.

We next show results that demonstrate that the time-parallel code shows transitions at similar points in time as the spatially parallelized code. Figure 8 compares the force extension profiles obtained from the GROMACS spatially parallelized code and our time parallelized code. Both codes show transitions (inferred from the peaks) at similar points in time. Figure 9 shows that the RMSD differences between a time-parallel run and the exact trajectory are sim-

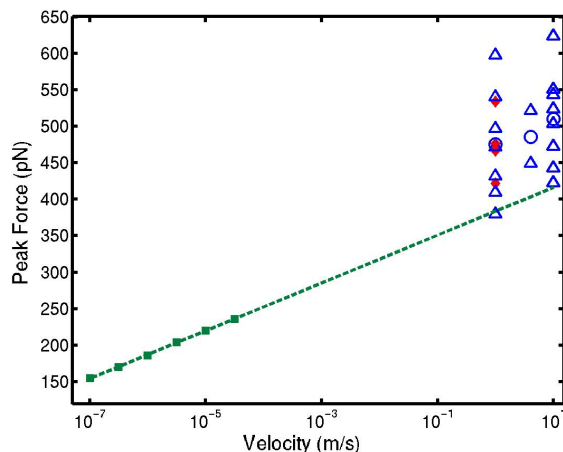


Figure 7. Squares are rupture forces obtained from experiments [1]. Triangles are rupture forces obtained from spatially parallelized runs. Diamonds are results of the time parallelized code (four data points at the slowest MD pulling speed). Circles are the means from spatially parallelized runs. The five-pointed star, almost coinciding with a circle and two diamonds, is the mean for the time-parallel run. The dashed line shows the linear least square best fit line of the experimental data.

ilar to the differences obtained using different random number sequences.

We now give possible reasons to explain the accurate statistics observed through time-parallelization, despite differences between predicted and computed states. Any numerical integration scheme for MD will not yield an exact trajectory, because small numerical errors will cause the simulated trajectory to diverge from the true one, as mentioned in § 3.3. Instead, we want the statistical properties for a collection of computed trajectories to be close to those of the exact trajectories. This is typically attempted using symplectic integrators (which are volume preserving in phase space). Such methods yield solutions that are close to the exact solution of an approximation to the force field used, over a time interval whose length is $O(1/\text{time step size})$ [10]. Note that being symplectic is a property of the time-integrator; if the time-integrator is symplectic, then this property is preserved in the time-parallel scheme.

The numerical integration scheme is also usually required to be time-reversible. A volume preserving and time-reversible integrator ensures detailed balance. Time-reversibility is not preserved in the time-parallel scheme. We now explain why this does not appear to affect our results. The set of conformations of the protein can be roughly

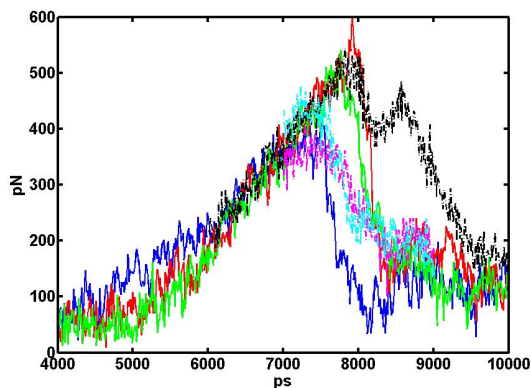


Figure 8. Plot of force versus time (or equivalently, extension). The solid lines are forces for three spatially parallelized runs, and the dashed lines are forces for three time parallelized runs.

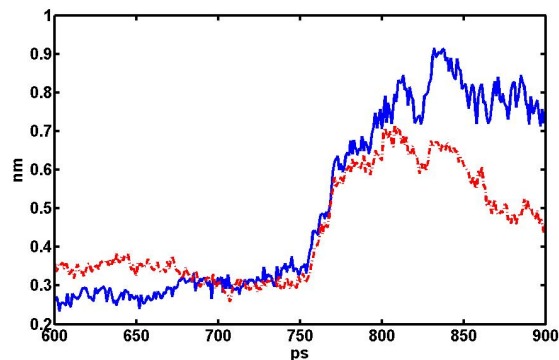


Figure 9. Plot of RMSD versus time. The solid line shows the RMSD between GROMACS spatially parallelized run and our time parallelized run. The dashed line shows the RMSD between two GROMACS runs with the same pulling rate, but with different seeds to the random number generator used in the thermostat.

partitioned into a number of basins. Major transitions happen when the protein moves from one basin to another, but most of the time is spent in a single basin. The error thresholds ensure that the predicted and computed states are considered equivalent only if they are close to each other in the same basin. The time-scales involved in going from the predicted state to the computed state (when the differences are below the thresholds) would be very short compared with the length of a time interval (which is 10 ps). This, perhaps, causes the results to be accurate.

5. Conclusions

We have showed the promise of data-driven time parallelization in a practical biological soft-matter application – AFM simulation of proteins. When combined with conventional parallelization, it extends the latter’s scalability by an order of magnitude.

Acknowledgments

We acknowledge NSF for providing funding under awards CMMI0403746, CNS0551472, and DMS0626180, and NCSA, ORNL, and NERSC, for providing computing time. A.S. expresses his gratitude to Sri Sathya Sai Baba for his help and inspiration in performing this work.

References

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

[2] O. K. Dudko, G. Hummer, and A. Szabo. Intrinsic rates and activation free energies from single-molecule pulling experiments. *Physical Review Letters*, 96:108101, 2006.

[3] E. Evans and K. Ritchie. Dynamic strength of molecular adhesion bonds. *Biophys*, 72:1541–1555, 1997.

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

[5] GROMACS: <http://www.gromacs.org>.

[6] G. Hummer and A. Szabo. Kinetics from nonequilibrium single-molecule pulling experiments. *Biophys*, 85:5–15, 2003.

[7] 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.

[8] Computational science: Ensuring America’s competitiveness, May 2005. Report of the President’s Information Technology Advisory Committee.

[9] Protein Data Bank: <http://www.rcsb.org/pdb>.

[10] R. D. Skeel. Symplectic integration with floating-point arithmetic and other approximations. *Applied Numerical Mathematics*, 29:3–18, 1999.

[11] 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.

[12] P. M. Williams, S. B. Fowler, R. B. Best, J. L. Toca-Herrera, K. A. Scott, A. Steward, and J. Clarke. Hidden complexity in the mechanical properties of Titin. *Nature*, 422:446–449, 2003.