Scalable Parallelization of Molecular Dynamics Simulations in Nano Mechanics, through Time Parallelization

Ashok Srinivasan
Florida State University
Dept. of Computer Science
Tallahassee FL 32306, USA
asriniva@cs.fsu.edu

Yanan Yu
Florida State University
Dept. of Computer Science
Tallahassee FL 32306, USA
yu@cs.fsu.edu

Namas Chandra
Dept. of Mechanical Engineering
Florida State University
Tallahassee FL 32310, USA
chandra@eng.fsu.edu

April 26, 2005

Abstract

Molecular Dynamics (MD) is an important atomistic simulation technique, with widespread use in computational chemistry, biology, and materials. An important limitation of MD is that the time step size is typically restricted to the order of femto (10^{-15}) seconds. Therefore a large number of iterations are required to reach realistic time scales. In fact, this is acknowledged as the major bottleneck in MD. While parallelization has been effective for dealing with the computational effort that arises in simulating large systems (that is, having a large number of atoms), conventional parallelization is not effective in simulating small physical systems to long time scales. Such simulations are oftentimes required for evaluating the physical, mechanical, and chemical properties at micro and nano scales. We recently introduced the idea of time parallelization through guided simulations, *where data from related simulations are used to parallelize a computation along the time domain. In this work, we use time parallelization to simulate a Carbon Nanotube (CNT), consisting of just around* 1000 *atoms, on up to* 990 *processors with close to ideal speedup. In contrast, spatial decomposition is not effective on more than 2-3 processors. The significance of this work is that it shows an approach that can deal with long time simulation of small to moderate sized physical systems, in realistic applications, where conventional parallelization is not effective. This novel approach further demonstrates that time can be treated, akin to space, as a parallelizable parameter, which is significant in scientific computing.*

1 Introduction

Nanotechnology can impact a range of fields, that include materials, electronics, pharmacy, and health care, among others. At the scale of a nano-meter, one cannot consider matter to be a continuum, and must understand physical phenomena occurring at the scale of atoms, or even smaller. Since mechanics of continuous media are not applicable when dealing with atomic scale, methods developed based on partial differential equations are not directly applicable. Due to size limitations in this scale, controlled experiments are very difficult, if not impossible, at this scale, and so computations play an even more important role in simulating the physics. Ab-initio quantum mechanics and Molecular Dynamics (MD) calculations have led the way in predicting the behavior of nano-elements. Ab-initio computations are very expensive computationally, and so MD is a popular alternative that is widely used.

Though MD is recognized as the practical computational tool to simulate nanoscale behavior, there are two fundamental problems that limits its use – its inability to simulate (i) large space and (ii) large time scales, due to computational constraints. For example, a simple device of size 1 cubic micro meter comprises of about 10^{12} atoms. Computations involving such a large number of atoms are all but impossible to handle directly. However, there has been a significant amount of research directed to solving the space limitation problem, using massively parallel machines and also multiscale modeling methods. As an example of the latter, Finite Element Method (FEM) may be combined with MD. In one example of this, MD is used near a region of interest (a crack tip), while FEM is used in regions far away from it [9]. Another example is the study of nano-composites, where a nano-material is dispersed in a polymer. MD can be used to determine the response of the nano-material, which is then used by the FEM at the continuum scale [4, 10]. The last example serves as a motivation for our application, namely, determining the realistic material response. The difficulty encountered is explained below.

A limitation of MD, more serious than the spatial scale, is its inability to simulate phenomena that take long time. The inherent time scale in MD for most materials is dictated by the vibrational modes of the atoms that restricts the time steps to the order of femto (10^{-15}) seconds. So even after the a few million time steps, which requires the order of a day of computational effort even for a small system with 1000 atoms, we can only simulate up to the order of nanoseconds. This may not be sufficient to get a realistic picture of the behavior of any material system. Any method that addresses this temporal scale limitation is expected to have tremendous impact, and this has been identified as an important challenge in nanoscale simulations and computational materials science [5, 11].

In this paper, we consider a single walled Carbon Nanotube (CNT) as an example physical system, and seek to determine its mechanical properties under tension (that is, when it is pulled). Details regarding this application, on MD, and on the importance of the time scale, are given in § 2. Since MD is used in a variety of other applications too, we can expect the impact of this work to be much broader than this specific application.

The usual approach to dealing with computational effort that arises from large physical systems is to parallelize it[1]. However, conventional parallelization is through spatial decomposition[2], and is not effective with small physical systems, since fine granularity leads to communication costs dominating the computational cost. When the computational effort arises from the long times required, parallelization of the time domain appears to be a natural possibility. However, time is not a quantity that is easily parallelized. We recently proposed a time parallelization approach through *guided simulations*, where results from related simulations are used to parallelize along the time domain.

The basic idea is to have each processor simulate a different interval of time. The problem is that each processor needs the state of the system at the beginning of the time interval it simulates, since we have an initial value problem. Typically, the current simulation is not the first one that is being performed; *usually, the results of many prior related simulations are available. We use results from one such related simulation (which we call the base simulation) to predict the state of the current simulation at a given point in time.* The relationship between the base simulation and the current simulation is updated dynamically as the simulation proceeds, to come up with increasingly better prediction. The predicted states are verified in parallel through exact MD computations, to ensure accuracy of the results. We explain this approach in greater detail, for the CNT application in particular, in § 3. We then briefly mention other approaches, for parallelization and to deal with the time-scale issue, in § 4.

We then present experimental results on CNT tensile tests. The material response determined is the stress-strain relationship, and the strain at which deformation mechanisms, which lead to the breakage of the CNT, start. The computation scales on up to 990 processors of a commodity cluster, with over 97% efficiency. § 5 gives the details of these numerical experiments, along with other tests that validate its accuracy. We also suggest the use of *flops per atom* as a measure of the ability to reach long time scales, since the larger it is, the longer is the time period that can be simulated in a fixed period of time. We achieve 420 MFlops per atom, for a total flop rate of 420 GFlops. We believe that our *flops per atom* rate may be the largest attained in MD computations of real applications so far.

We finally summarize our conclusions, and present directions for future work in § 6.

2 Carbon Nanotube Application

2.1 Tensile Test on CNT

The physical system we consider is a CNT. A CNT consists of Carbon atoms that are bonded to each other to form a molecular "tube". They possess a combination of properties never seen before; they have electrical conductivity higher than Copper, thermal conductivity greater than Diamond, stiffness much greater than steel, strength larger than Titanium, and weight lighter than feather. They are, therefore, an important topic of study in many potential applications of nanotechnology. One important application of CNTs is in nano-composites, where CNTs are embedded in a polymer

[1]The specific class of problems addressed here, that of evaluating mechanical properties, should be clearly distinguished from a host of other MD problems, where physical and thermodynamic properties, such as entropy and specific heat, are evaluated. In the latter set of problems, several independent simulations, starting from different initial states, are performed, and their results averaged in some sense. This leads to a trivially parallelizable solution scheme. Such a scheme is not possible when mechanical loads are applied or when fracture can occur at any point depending on the local properties.

[2]Apart from spatial decomposition, force decomposition techniques are also used in MD [12]. However, it is conceptually close to spatial decomposition, and so we do not address these separately. Time parallelization, in contrast, is both, conceptually quite different from spatial decomposition, and potentially more scalable.

matrix. It is hoped that the CNT will impart much of its strength to the nano-composite.

In such applications, it becomes important to determine the mechanical properties of the CNT. One important test is the tensile test, in which the CNT is pulled at a constant velocity, as shown in Fig. 1. The response of the material is characterized by stress (force required to pull the tube, divided by it cross-sectional area) for a given strain (the elongation of the nanotube, relative to its original length). A stress-strain curve, as shown in Fig. 6 later, describes the behavior of the material when it is pulled at the specified velocity (more formally, strain-rate). Such a curve, for example, could 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.

2.2 Molecular Dynamics

The state S_t of the system at any time t is defined by the position and velocity vectors, at time t, of the Carbon atoms that form the CNT. If there are N atoms in the systems, then there are $6N$ quantities (three position coordinates and the three velocity coordinates per atom) that define the state. The properties of the CNT at time t can be determined from these. Given S_t, we can compute $S_{t+\Delta t}$, at the next time step $t + \Delta t$, as follows, thus tracking the time evolution of the state, and consequently, the CNT properties. The forces on each atom are computed. This is the computational bottleneck. The forces depend on the positions of atoms that are "close by". The exact computation of the forces, and which atoms are considered close by, vary, depending on a "potential function" that is used in the calculation, whose choice is dictated by the type of bonding (for example, covalent, ionic, or van der Waals). These energy potentials establish a force-distance relationship between a group of atoms in the presence of all other atoms. These potentials are empirical in nature and are formulated to fit experimental observations. It is critical to select the right potential to use, since certain key atomistic processes relevant to the problem may not be captured by all potentials. For example the popular and simple Lennard-Jones potential is incapable of capturing the bond type (SP_2 to SP_3 changes) in Carbon, and the more complex and accurate, but computationally intensive Tersoff-Brenner [2] potential needs to be used for our problem.

Once the forces on the atoms are computed, the new positions of the atoms can be calculated using Newton's laws of motion. A numerical time integration scheme is used for this. We use a fourth order Nordsiek scheme. The time step Δt has to be chosen to ensure that the numerical scheme is stable, and also accurate. If Δt is large, then fewer time steps are required to simulate to a given point in time; however, the numerical scheme may not be stable for large Δt. For MD computations, the time step is typically required to be less than a femto second (10^{-15}s).

2.3 Time Scales

The small step size mentioned above proves to be an impediment to effective MD computation. To illustrate this, let us consider a CNT with 1000 atoms, having initial length 10 nm (nano meters), and let Δt be 0.5 femto seconds. Let us pull the CNT fast enough so that it elongates by around 10% in a μs (micro second). This is a large strain rate, and the velocity at which one end is pulled is then 0.001 m/s. The CNT breaks at around 20% strain, and so to simulate up to that point, we would require two billion time steps – that is, over a year of sequential computational effort. For lower strain rates, the time required is correspondingly higher. Furthermore, this computation will not parallelize efficiently on more than 2-3 processors using conventional spatial decomposition, and so the time required, even in a parallel computation, is around half a year. As an alternative, researchers typically simulate at a faster rate, typically 10 m/s for a CNT this size, in which case the same strain is reached with around a day's computational effort. It is assumed that the stress-strain relationship determined at this higher strain rate is the same as that which would be obtained under a lower strain rate. However, it is known that such an assumption is not accurate when the strain rates vary by several orders of magnitude [18, 17]. On the other hand, if we were able to parallelize the computation efficiently on a large number of processors, then we could reach the the desired time scale with more realistic strain-rates too.

3 Time Parallelization through Guided Simulations

We recently [13, 14] introduced the idea of guided simulations to parallelize along the time domain. We describe it below, for completeness, but specialized to the CNT application. A more general description is provided in [14].

3.1 Time Parallelization

Let us divide the time period for which the computation has to be performed into a number of time intervals, such that the number of time intervals is much greater than the number of processors. In this section, we will let t_i denote the beginning of the i th time interval. *Each time interval may require several steps of the time integration algorithm. In fact, we will use around* 1000 *time steps per time interval in our experiments.* In a

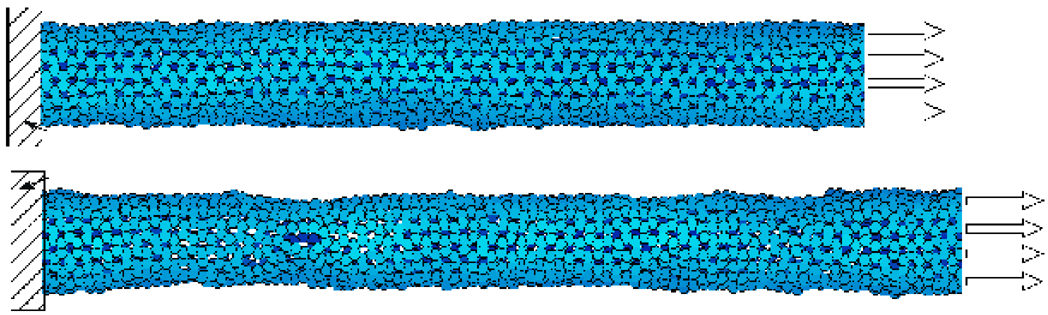

Figure 1: Schematic of the problem. A Carbon nanotube is held fixed at one end, and pulled at the other end at a constant velocity. The higher figure shows an initial configuration, and the lower figure the configuration at a different point in time.

parallelization along the time domain, different processors compute the state of the CNT at different time intervals. The difficulty here is that the computation of the states is defined iteratively, with the result for the previous point in time being needed for computing the state at any time. So it is not clear how each processor can independently and simultaneously compute the state for a different time interval. Our approach is to dynamically determine a relationship between the current simulation and a prior *base* simulation, whose results have been recorded, enabling us to possibly predict the state at different points in time, in parallel. We then verify if the predictions are accurate, again in parallel. Further details on prediction and verification are given later. Fig. 2 shows a schematic of the time parallelization idea, while algorithm 3.1 describes it formally. In the algorithm, the function $Predict(\hat{S}_i, i, l)$ predicts the state at time $l \geq i$, given a state $\hat{S}_i$ at time i, using certain prediction parameters, which are explained later for the CNT application. The function $UpdatePredictionParameters$ learns to predict better, from the difference between the prediction and the verification states.

Note the following:

1. Processor 1's accurate MD result is correct, since it started from the known initial state S_0. So the computation always progresses.

2. If there are P processes, then process P's prediction for time t_P is used only for updating the predictor. Difference between the predicted and computed states at time t_P does not cause any computation to be discarded, since the predicted state at time t_P is not used as the initial state of any processor.

3. All the processors must use the same predictor; otherwise verification of prediction at time t_i at processor i does not imply that the prediction for initial state at time t_i on processor $i+1$ was correct.

4. The answers given should be accurate, if our definition of the predicted and verified states being "sufficiently close" is correct. A good predictor enables greater speedup, while a poor one leads to it becoming a sequential computation.

3.2 Prediction

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. So we will not try to predict the state at some arbitrary point in time directly. 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 states at the times required, as shown in the calls to *Predict* in algorithm 3.1. We accomplish this by predicting the change in each coordinate (of the positions of the atoms) independently. In the description here, 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. It is easy to change between the actual and normalized coordinates. Using the normalized coordinates is advantageous, because it enables us to use base and current simulations that use CNTs of different sizes. Similarly, for prediction purposes alone, the relative times in the base and in the current simulations are normalized 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 actual time t will be related to actual time $t/10$ in the base.

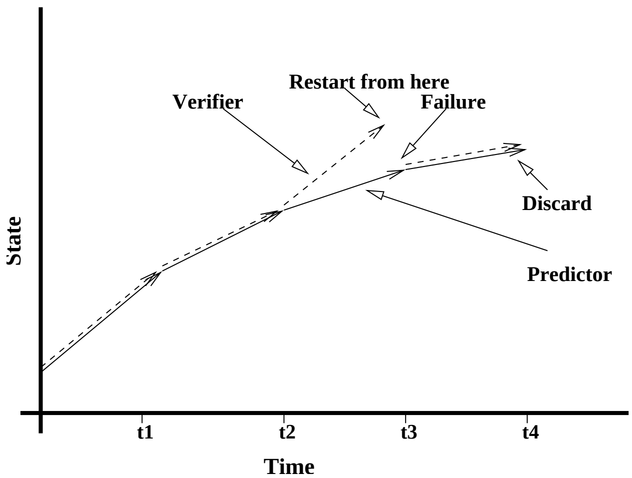

Figure 2: Schematic of parallelization of time. Processor i, for each $i \in [1, P = 4]$, somehow predicts the states at time t_{i-1} and t_i (with the state at time t_0 being a known initial state S_0), using the results of the base simulation, and its relationship to the current simulation. Then each processor i performs accurate MD simulations starting from the predicted state for time t_{i-1} up to time t_i, and verifies if the prediction for t_i was accurate. Both prediction and verification are performed in parallel. In the figure above, the predicted state for t_3 was inaccurate, and we say the process 3 *erred*. Computations for subsequent points in time too have to be discarded, since they might have started from an incorrect start state. The next phase starts from time t_3 (since the verification step actually computed the correct state for t_3), and computes states for times t_4, t_5, t_6, and t_7. 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 old simulations.

Let x_t represent a (normalized) coordinate at time t. Using two terms of the Taylor's series, we have

$$x_{t+\Delta t} = x_t + \dot{\mathrm{x}}_{t+\Delta t}\Delta t, \tag{1}$$

where $\dot{\mathrm{x}}_{t+\Delta t}$ is the actual slope dx/dt at some point in $[t, t + \Delta t]$. We do not know the value of $\dot{\mathrm{x}}_{t+\Delta t}$, but will try to predict it.

We will consider a finite set of basis functions, $\phi_0, \phi_1, ..., \phi_k$, which are functions of the coordinates of the atom positions, and express $\dot{\mathrm{x}}$ in terms of it. For example, we can take a polynomial basis $1, x, x^2$. These basis functions should ideally be chosen so that they represent the types of changes that can occur under physical phenomena that the CNT might experience. For example, for the tensile test, we use only the x term for the coordinates orthogonal to the direction in which the CNT is pulled, and 1 and x for the direction in which it is pulled. This makes it suitable for stretching in the

Algorithm 3.1: $\textsc{TimeParallelize}$(Initial State $\mathrm{S_0}$, Number of processors P, Number of time intervals m)

$i \leftarrow 0$
$\hat{S}_0 \leftarrow S_0$
while $i < m$
do
  for each processor $j \in [1, \min(P, m - i - 1)]$
  do
    $T_{i+j-1} \leftarrow$ Predict $(\hat{S}_i, i, i + j - 1)$
    $T_{i+j} \leftarrow$ Predict $(\hat{S}_i, i, i + j)$
    $\hat{S}_{i+j} \leftarrow$ AccurateComputation(
      StartState $= T_{i+j-1}$,
      StartTime $= i + j - 1$,
      EndTime $= i + j$)
    UpdatePredictionParameters(
      CurrentParameters, $\hat{S}_{i+j}, T_{i+j}$)
    if IsDifferenceTooLarge($\hat{S}_{i+j}, T_{i+j}$)
      then $\mathrm{Next}_j \leftarrow j$
      else $\mathrm{Next}_j \leftarrow P$
  $k \leftarrow$ AllReduce($Next, min$)
  if $j = k$
    then Broadcast $\hat{S}_{i+j}$, prediction parameters
  for each processor $j \in [1, P]$
    do in parallel $i \leftarrow i + k$

direction in which it is pulled.

Let $\dot{\mathrm{x}}_{t+\Delta t} \approx \sum_i a_{i,t+\Delta t}\phi_i(x_t)$. Once we have performed an accurate simulation for time t, we know the actual $\dot{\mathrm{x}}_t$ for each atom, and can perform a least squares fit to determine the coefficients $a_{i,t}$. We can express changes in the base similarly, and determine its coefficients, say $b_{i,t}$. The values of $b_{i,t}$ are available for the entire duration of the base simulation. The coefficients $a_{i,t}$, on the other hand, are only available until the time step simulated so far. If the base simulation and the current simulation are almost identical, then we can approximate $a_{i,t+\Delta t}$ by $b_{i,t+\Delta t}$. However, the simulations will typically differ, and so we wish to correct by adding the difference between the two simulations $R_{t+\Delta t} = a_{i,t+\Delta t} - b_{i,t+\Delta t}$, which is unknown. As a first approximation, we can assume that $R_{t+\Delta t} = a_{i,t} - b_{i,t}$ to yield the approximation $a_{i,t+\Delta t} \approx b_{i,t+\Delta t} + a_{i,t} - b_{i,t}$. On one hand, using the latest $a_{i,t}$ available might give the best estimate. On the other hand, random fluctuations in the MD simulation lead to somewhat poor results if we depend on only evaluation at one point in time. So we set $R_{t+\Delta t} = (1 - \beta)R_t + \beta(a_{i,t} - b_{i,t})$, where β is the weight assigned to the latest value. The term R_t represents the relationship between the base and current simulations, and updating it at each time step represents a simple form of learning.

Since $a_{i,0}$ and $b_{i,0}$ are unknown, we need to chose a suitable method of starting this process. We assume a linear increase with time, in the values of the coordinates of atoms in the direction in which the CNT is pulled, with the constant of proportionality being a function of its normalized coordinates. This is not a very good initial choice when the interval of time is very large. But after the first phase of computations in the while loop in algorithm 3.1, the error due to this choice is reduced rapidly.

The velocity distribution of the atoms was predicted to be the distribution at the previous point in time for the current simulation. Since the numerical simulations were carried out at constant temperature, this was sufficient. (MD does not automatically preserve temperature. So a process called "thermostating" is performed, which modifies velocities to keep temperature constant. This occurs both in the sequential and in the parallel algorithms.)

3.3 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 states are sufficiently close.

MD simulations bring an interesting issue – that of determining the equivalence of two dynamic states. In nature, atoms vibrate around their mean position, even when nothing interesting is happening to the physical system. MD simulations track these vibrations, as shown in Fig. 3. So if we look at the states of a physical system in equilibrium, at two different points in time, it will be unlikely that the atoms will be in the same positions, even though they represent the same system. So we cannot expect the predicted state to have atoms in the same positions as the in accurate simulations either. Instead, we observe that the specific state observed in the MD simulation can be considered a sample from an infinite number of possible states. We need to verify if the predicted and the accurate states could be samples from the same distribution. In the parlance of MD, the two states should be samples from the same ensemble. Different types of ensembles are defined in statistical mechanics, based on certain overall properties of the system (such as number of atoms, volume, and temperature) being identical. We use a stricter definition than those, since significant differences in micro-states (that is, states of portions of the CNT) may affect our results.

We give an informal description of our choice for defining the equivalence of states below. Let us consider two states having the same number of atoms, and a known mapping of atoms on one state to the atoms in the other state. This occurs in the predicted and accurate simulations, since they represent the same physical system but just assign different positions and velocities to each atom. We consider the two states to be equivalent for our purposes if the following hold.

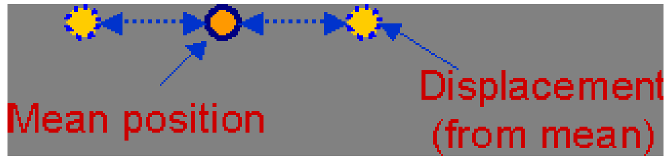

Figure 3: Atoms vibrate around some mean position even when nothing interesting is happening to the system.

1. We will compute the distance between the positions of corresponding atoms in the two states. We will check if the largest distance is greater than a threshold. This threshold is determined by performing MD simulations and observing the range of movement of an atom, at the temperature of the simulation of interest, without the CNT being pulled. Based on this we choose the maximum permissible distance between two equivalent states as the largest range of movement.

2. Similarly, we also verify if the average distance between atoms in the predicted and accurate computations is within a threshold. This threshold too, was determined as above, but considering the average distance an atom moves from its mean. It is possible for two states that are actually not identical to have the same mean and maximum deviation. So we will consider other factors.

3. There are two types of energies that are important. The potential energy is a function of the set of positions of the atoms. However, it is quite sensitive to the positions, and two states with acceptable mean and maximum deviations can have very different potential energies. The reason for this is that the potential energy function that we use (Tersoff-Brenner potential) depends on not just the distances between atoms, but also on their orientations and dihedral angles. It is also possible for the potential energies of two states to be similar, but for the maximum deviation to be high, since one atom may not affect the potential energy much in a system with many atoms. However, a single atom moving far away is an important event to be tracked, since it may represent the material starting to break, which we wish to observe. So all the above criteria are used.

4. In addition, the kinetic energy (or, equivalently, the temperature, which are both measured by the velocities of the atoms) is also measured. Thresholds for both the energies too were determined based on other MD simulations, and differences were required to be below the thresholds.

3.4 Time Required

Each processor performs MD computations for their time interval. We choose the time interval to be 1000 time steps, which requires the order of 10 seconds on an Intel Xeon processor running at around 3 GHz. The remaining computational overhead, which is relatively small, is mentioned below.

Each processor performs two predictions. This requires two file reads, and nine least squares computations (three coordinates each, for the base at two time points, and for the current simulation after the verification step). The least squares computation takes time linear in the size of the system, but quadratic in the number of coefficients (the latter is a small constant). An *AllReduce* on one integer (a process rank) is performed to determine the smallest indexed processor to err. A broadcast of the entire state of this processor, and the coefficients of its predictor basis functions, is performed, so that all processors will have the same coefficients. The overhead for all these operation is fairly small, related to computation for 1000 time steps, and so the efficiency is very high, even on up to 990 processors.

3.5 Error Propagation

We need to consider the numerical stability of the predictor. That is, can repeated small errors from the prediction lead to the final result being incorrect after a long period of time? We note that the new start states of each iteration of the simulation are the results of the accurate verifier computations that start from initial states that might have small errors. Numerical schemes are normally analyzed for their numerical stability. Informally, we can describe a numerical scheme as stable if small errors resulting from the numerical approximations do not propagate or magnify with time. In our case, the errors arise from the prediction, and if they are small, the numerical scheme used in the time integration in MD should not propagate it. From a physical point of view, small changes in an atom's position would cause inter-atom forces on it to push it back to the right position in our system. However, if the error is very large, then this may not happen. We expect the verifier to detect such large errors. In order to validate our results, we compared the stress-strain results from the parallel run with the exact sequential run, to verify that the errors did not propagate. This gives empirical evidence for the stability of our method for this particular application, with the particular error criterion used.

4 Related Work

4.1 Prior Work

We introduced the concept of time parallelization using guided simulations for scientific applications in [13, 14], and gave its general description. We also demonstrated the effectiveness of the technique in practice, using a CNT computation with tensile test as an example in [14]. The results here improve on those of the previous work in the following manner. In [14], the base simulation (pulled at 12.5 m/s) and the current simulation (pulled at 10 m/s) had similar time scales – that is, they displayed similar behavior at similar points in time. The great promise of time parallelization is in being able to simulate to much longer time scales than is currently possible. In such situations, there will be no base simulation that goes to as long a time period as the current simulation. So it is important to be able to use a base simulation, whose short-time behavior is related to the long-time behavior of the current simulation. Then a shorter base run can be used to predict the long time behavior of the current simulation. In this paper, we consider a current simulation that has an order of magnitude longer time scale than the base simulation. In the current simulation, the CNT is pulled at 1 m/s, in contrast to 10 m/s in the base simulation. A consequence of the longer time scale is, for example, that the CNT breaks at a much later point in time than with the base simulation.

Yet another achievement in this paper is that we parallelize the computation efficiently on up to 990 processors, in contrast to 50 processors in the previous work. In order to achieve this, it was also necessary to modify the basis functions used, compared with those used in the earlier work. Furthermore, some improvements in the implementation too were performed, such as the use of asynchronous I/O to read the base simulation files.

In this work, we also simulate to much longer times (around 3.5 million time steps, versus 200,000 time steps earlier), and simulate the CNT until after it starts breaking. So we also establish the effectiveness in prediction in the region where the material behavior is nonlinear.

We had mentioned the important issue of error propagation earlier. In this work, we compare the material response from time-parallelized simulation to the exact sequential simulation. This gives empirical evidence to show that the error does not propagate, at least until the CNT starts to break. This is another new result in this work.

4.2 Other Approaches

In view of the importance of MD calculations on CNTs using Brenner's potential, there have been many works reporting their parallelization. Notably, [15] parallelized it in a shared memory paradigm, and obtained good efficiency ($\geq 75\%$) at a granularity of around 2000 atoms per processors. In [7], we reported spatial parallelization results with good efficiency at granularity of around 500 atoms per processor, using distributed memory parallelization. In the same paper, we also showed why the nature of Brenner's potential makes a difference in the effectiveness of some popular parallelization strategies.

There have been several publications on parallelizing MD computations in general, using other potentials and physical systems. For example, LAMMPS [8] has been used to perform a 10 billion atom MD simulation recently, using the Lennard-Jones potential, on over 32,000 processors of the IBM Blue Gene/Light at Lawrence Livermore National Lab. They estimated the efficiency, based on reasonable assumptions, as around 90%. The granularity of this computation is a little over $300,000$ atoms per processor. The aggregate flop rate is $2.3TFlops$. Note that good efficiency is obtained due to the extremely large physical system, and high flop rate due to the large number of processors and the high efficiency. The number of time steps simulated was 100. So this is an example of a large physical system simulated to a short time scale. In contrast, we wish to simulate a small physical system to a long time scale.

Another popular MD software is NAMD, which uses spatial and force decompositions. It has scaled to 3000 processors using 327,000 atoms [12], with an efficiency of around 32%, with a flop rate of 467 GFlops. The granularity of the computation is around 100 atoms per processor. With a higher granularity, for example, 641 atoms per processor on 500 processors, the efficiency is higher, at around 87%, for a flop rate of around 200 GFlops. These results [12] were obtained on an Alpha cluster with Quadrics interconnect, using the Elan communication library. The scalability of this code is very good for conventional parallelization.

The point of mentioning the above results is to indicate the need for sufficiently coarse granularity in conventional parallelization. For example, if we were to simulate a system as small as ours (where only 800 of the 1000 atoms is actually moved using MD) on 990 processors, then conventional parallelization will lead to a granularity of less than one per processor, which is not feasible. The above results are not meant to directly compare the scalability of the different software packages, since they simulate different physics on different types of machines. Consequently, the sequential times differ considerably – [8] takes about 9μs per atom per iteration on the Blue Gene, while [12] takes about 86μs per atom per iteration on a 1 GHz Alpha processor (our code takes around 13μs per iteration on a 3.2GHz Intel Xeon processor).

Time parallelization using the Parareal approach [1] is a promising alternative to conventional parallelization. However, the speedup and efficiency obtained have been limited. We described the limitations in detail in [14]. We just mention some salient features here. The Parareal approach involves a sequential phase for prediction, which limits scalability. Consequently, the speedup and efficiency have been rather modest, even when used on some "toy" model problems. Speedups on simulated experiments (ignoring communication costs – the experiments were not on actual parallel machines) ranged between 8 to 130, with efficiency between 25% and 1.3% repectively on the model problems.

Some other approaches to dealing with long time scales, without recourse to parallel computation, are time extension schemes such as hyper MD [16] and Kinetic Monte Carlo [6]. However, these techniques have many limitations, described further in [1, 5]. They have not been used with CNTs, and it is not clear that they can be used here either.

4.3 When is Time Parallelization Appropriate?

Time parallelization requires the existence of a relationship between simulations performed with different parameters (such as velocity in our case). In order to simulate for long time scales, we will, in particular, need the feature that long term behavior under certain simulation parameters should be related to short time behavior under a different parameters. A simulation with short time scale parameters, from an existing database of simulation results, can then be used to predict the long time behavior under a different set of parameters. (Note that some short time scale data themselves may be results of time parallel runs.)

Such linking of time scales is often used in engineering design. For example, material characterization and property prediction are typically needed for engineering products that far exceed the test periods. Understanding the long-term behavior of materials, in the order of 30 years, is required before permitting their use in many applications, such as in the nuclear industry. It was quickly recognized that long-time room temperature behavior can be related to short-time high temperature behavior. Thus a relation between time and temperature was developed to predict the behavior of materials in 30 years by studying them at elevated temperature in less than a month. Similar relation between time and increased stress was used in creep studies where the long-time, low-stress, high-temperature behavior can be simulated from short-time, high-stress, high-temperature behavior. In these examples, the time extensions were achieved by exploiting the relation between short-time behavior to long-time behavior with appropriate parameters.

If there were a known relationship between the base and current simulations, then there is clearly no need to perform the current one, since its results can be inferred from that of the base. However, such relationships are often not known, and may only hold for a certain period in time. The relationship obtained is updated on observing the error in prediction after each verification step, as we "learn" more about the relationship between the simulation being performed and the base.

Given the above requirements for success in time parallelization, we may wonder if other techniques can exploit the same features. For example, given a database of high dimensional data, reduced order modeling techniques create a lower dimensional model which approximates the solution well. Simulations with the lower dimensional model can be performed much faster than with the higher dimensional model. A variety of reduced order modeling techniques have been successfully applied to PDEs, and have proven effective in many situations that involved interpolation between the parameters of existing data [3]. We believe this will be the situation with CNTs too. For example, if we had data available for CNTs pulled with velocities 10 m/s, 8 m/s, 6 m/s, 4 m/s, and 2 m/s, then we might be able to use a reduced order model for performing a simulation at 5 m/s. On the other hand, if we wished to simulate at 0.1 m/s (extrapolating from the existing parameters), then the reduced order solution might be erroneous. A time parallel run, on the other hand, performs verification steps. Poor predictions slow down the computation, but the answers will still be accurate, provided a correct error criterion is chosen. Furthermore, as the simulation proceeds, the predictor learns the relationship better and so improves.

In general, we envisage a situation where time parallelization is initially used to simulate to long time scales, and these results used to populate a database with sufficient simulation results that reduced order models can be constructed. Once this is accomplished, reduced order models would be used for parameters that involve interpolation. When one wishes to extend the range of parameters, then time parallelization would be used.

We also note that it is possible to combine spatial and time parallelization. Instead of a single processor being responsible for computing a single time interval, a group of processors, that distribute the atoms across the group, can be used to simulate each time interval. Thus spatial and time parallelization can complement each other.

5 Experimental results

Both the base and the current simulation pull one end of a Carbon nanotube containing 1000 atoms at constant velocities, but with different velocities for the base and the current simulations. The predictor, of course, did not make use of the fact the the same number of atoms were simulated. The Tersoff-Brenner potential is used in the MD simulations. Use of this potential requires more computational effort than many other potentials, but it is known for its accuracy in CNT calculations. In fact, communication cost with conventional parallelization too is higher.

One end of the tube is kept fixed by forcing around 100 atoms at that end to remain stationary. The tube is pulled at a fixed rate in the z direction by forcing around 100 atoms at the other end to increase their z coordinate values at a rate of u m/s. We used $u = 10$ m/s for the base simulation, and 1 m/s for the current simulation. The time step in the MD simulations was 0.5 femto seconds, and a time interval was 1000 time steps. Around 800 atoms are simulated using MD, and the end atoms moved deterministically as mentioned above.

We used results from an MD simulation conducted at a temperature of $300K$, whose output had been recorded every 100 time steps, over total $700,000$ time steps, as the base case. This is well beyond the point where the CNT starts to break (around $350,000$ time steps – this is the point in time from when the stress decreases with increase in strain). The predictor used $\beta = 0.5$. We had earlier [14] observed that the results are not very sensitive to the value of β, as long as β is not close to 0 or 1. The new simulations were performed until the CNT starts to break, at around 3.5 million time steps. Results after the CNT starts to break are not of value in our application, though it is still interesting to observe them.

5.1 Speedup Results

Speedup results are reported 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 in Lustre. Myrinet 2000 and Gigabit Ethernet interconnects are available. We used the Myrinet interconnect. The ChaMPIon/Pro MPI implementation was used with gcc/g77 compilers for our mixed C/Fortran code, compiled with '-O3' optimization flags set. The MPI calls were purely in the C code. The timing results are based on wall clock time when run in dedicated mode.

Fig. 4 shows the speedup results for the 1000 atom simulation. We ignore the initialization time for our code, which is small ($\approx$ 1% of the time of one time interval). We also ignore the time required by the system to start the processes on the machines and to call MPI_Init. The latter two operations consume a significant portion of the time, but are independent of the algorithm, and are also one time costs. We can see that speedup is almost the ideal linear curve up to 990 processors on the Xeon cluster. The processors never err in the course of the simulation (up to the point where the CNT starts to break), and so loss in speed is only due to the overheads of prediction, communication, and reading the base simulation results from disk. These overheads were small, compared with the computation time. For example, the least square and other prediction related computations takes $\approx 10^{-3}$s, file read $\approx 0.01 - 0.2$s seconds, with asynchronous file I/O even smaller, the AllReduce $\approx 10^{-4} - 10^{-3}$s, and Broadcast $\approx 10^{-1} - 10^{-2}$s between 50-1000 processors. Load imbalance is not an issue, since each processor performs, essentially, the same amount of computation. All the overheads are insignificant (total ≤ 0.4s), relative to the computation time (≈ 13s) for simulating a single time interval.

We performed similar simulations on up to 500 processors on the *Eagle* IBM SP3 machine at Oak Ridge National Lab. The efficiency was over 90%, but still a little lower than that on the Xeon cluster. The probable cause was that we performed the computations in non-dedicated mode. The sequential speed is also lower on the IBM machine, as was the largest number of processors that we used, and so the highest flop rate reached was 101 GFlops.

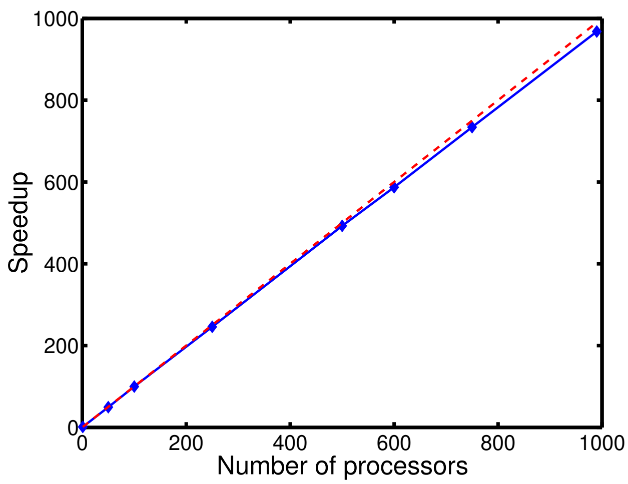

Figure 4: Speedup curve. The red dashed line shows the ideal speedup, and the solid blue line shows the observed speedup on NCSA Xeon cluster. The flop rate on the Xeon cluster is 420 GFlops on 990 processors.

The flop rate on the 990 processor run on the Xeon cluster was computed to be around 420 GFlops, as follows. First, the *number of floating point operations* per time step was determined using *tpmcount* on the ORNL SP3 machine, over 10,000 time steps. Then, from the time per time step on the 990 processor run, the flop rate was determined. The flop rate per atom gives an indication of how long we can simulate in time for a given physical system and potential function. The flop rate per atom for our simulation on 990 processors of the NCSA Xeon cluster is 420 MFlops/atom. The flop rate per atom in [12] is 1.4 MFlops/atom, where we use the data for the run on the largest number of processors reported. (The flop rate per atom in [8] is very low due to the large number of atoms.) We are not aware of any MD simulation attaining a greater flop rate per atom than ours in a real application, and believe that ours might be the largest.

We next wish to see how the predictor learns about the simulation. Fig. 5 shows the average absolute difference in positions of corresponding atoms of the predictor and the verifier, for a simulation on 500 processors. The error is initially high, since we do not know the behavior of the current simulation or its relationship to the base then. Subsequent phases learn about the relation between the base simulation and the current simulation and predict more accurately, though the error increases slightly just before time interval 3000. This time represents a strain of a little lower than 0.15, where, as can be seen in Fig. 6, the slope of the curve increases, indicating different material behavior. But just after time interval 3000, the error starts to decrease again, indicating learning.

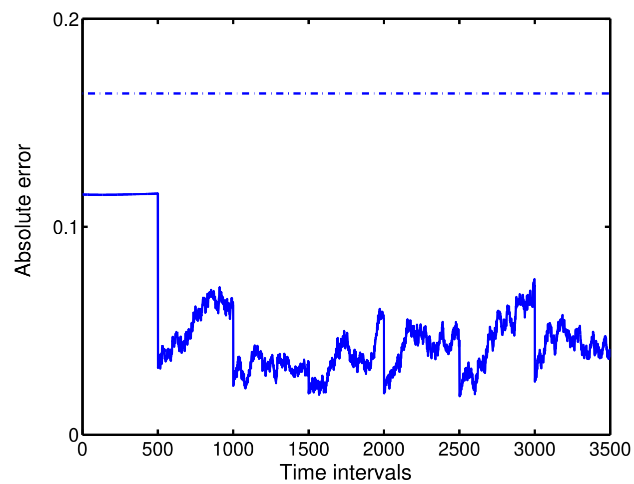

Figure 5: Plot of average absolute difference between positions of corresponding atoms of the predictor and verifier states, for a 500 processor simulation. The solid blue line shows the error, and the dash-dotted line the error threshold.

5.2 Validation

We had discussed the issue of error propagation in § 3.5. We gave reasons why the error will not propagate, if each error is sufficiently small. We also described the reasons behind choosing our particular error criteria in § 3.3. However, since time parallelization is still a new idea, it is important to validate the results with the exact answer. In fact, this desire to validate the results was the reason for choosing the velocity with which the CNT is pulled as 1 m/s. Using 990 processors, we could alternatively have performed an MD simulation that was three orders of magnitude smaller than the base of 10 m/s. However, it would not then be feasible to compare with the exact sequential results. We therefore chose a higher velocity.

Fig. 6 shows the plot of stress versus strain, which is a material behavior of practical interest. The time-parallel results agree very well with the exact sequential results, up to the point where the CNT starts to break. Fig. 6 shows this for a 500 processor run. The time at which the CNT starts to break is also determined as with the sequential run, at a strain of around 0.17.

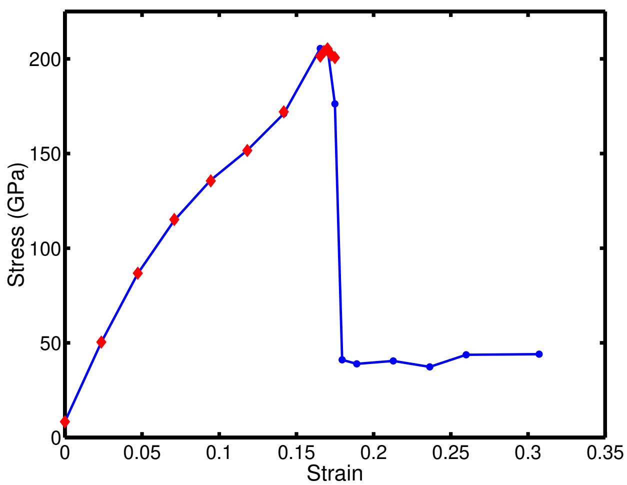

Figure 6: Stress-strain plot. The solid blue line denotes the exact curve from sequential simulations, with the blue dots indicating the data points. The red diamonds show the data from the parallel simulation on 500 processors, up to the point where the CNT starts to break (the stress decreases with increase in strain). The first six data points for the sequential and the parallel run are so close that the blue dots for the sequential data are not visible. Note that the stress starts with a positive value even with no strain, since the CNT is conceptually formed by rolling a graphite sheet, which introduces stress.

After the point of breakage of the CNT, the parallel and sequential run don't agree very closely. The reason for this is that our error criterion used the behavior of an intact CNT to decide if two states are equivalent. This is not a good enough criterion for a CNT that has started breaking. Furthermore, after the CNT starts to break, processes do err, since the basis functions where chosen for their suitability to stretching, and not for breaking. Results beyond the point where the CNT starts to break are not of any practical use in our application. However, from a computational perspective, it will still be interesting to see if the suitable error conditions and basis functions can be developed to predict behavior after the CNT starts to break too. Despite the fact that the simulations show different results, we observed that a similar type of failure of the CNT is revealed in both the parallel and in the sequential simulations.

6 Conclusions

We have demonstrated the effectiveness of a new approach to parallelization, namely, time parallelization using guided simulations, in a practical application. We have scaled it to an order of magnitude more processors than was possible with the same approach earlier. Furthermore, we have demonstrated its effectiveness in predicting for a simulation, using a base simulation that has an order of magnitude smaller time scale. Furthermore, it scales to more than two orders of magnitude larger number of processors than conventional parallelization for the same physical system. The flops per atom rate is also higher than that for any other MD simulations that we are aware of. Since MD is used a wide variety of applications, these results suggest a promising approach to dealing with the difficulty of performing MD simulations to long time scales. Large computational systems, consisting of well over 10,000 processors, are now available. If this approach can be scaled to such large systems, then it will be possible to perform MD simulations to four orders of magnitude longer time than currently feasible.

Some of the future work is as follows. We wish to perform MD simulations that reach even longer time scales, so that they can provide results under realistic experimental conditions that are encountered. We also wish to scale the computations to an order of magnitude larger number of processors. This will involve better initialization, and also some method of validating the results for their correctness, without using the sequential algorithm directly. Prediction is currently made using a base simulation that differs in one parameter – the velocity. This should be extended so that a database of base simulations, differing from the current one in perhaps several parameters, are used to predict the behavior of the base. Yet another future work is to perform predictions for more complex experimental conditions. The basis functions should, then be developed so that they correspond to different types of physical phenomena that may be experienced by the system. We will also integrate out time parallel code with our existing FEM code that uses MD results [4, 10].

Acknowledgments

This work was funded by NSF grant # CMS-0403746. We thank ORNL and NCSA for providing computer time, and Dave McWilliams of NCSA, in particular, for help in getting the large jobs to work. Computer time at ORNL was provided by CNMS/NTI grant #CNMS2004-028. Computer time at NCSA was provided under proposal #ASC050004. We also thank Xin Yuan at Florida State University for permitting use of his Linux cluster, where our codes are first tested. A.S. gratefully acknowledges his debt to Sri S. S. Baba, but for whose help and inspiration, this work would not have neither been started nor been accomplished.

References

[1] 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.

[2] D. W. Brenner. Empirical potential for hydrocarbon for use in simulating the chemical vapor deposition of diamond films. *Physical Review B*, 42:9458–9471, 1990.

[3] J. Burkardt, Q. Du, M. Gunzburger, and H. Lee. Reduced order modeling of complex systems. In D. F. Griffiths and G. A. Watson, editors, *Proceedings of the 20 th biennial conference on Numerical Analysis*, pages 29–38, Dundee, Scotland, U.K., 2003. University of Dundee.

[4] N. Chandra, S. Namilae, and A. Srinivasan. Linking atomistic and continuum mechanics using multi-scale models. In *Proceedings of the 8th International Conference on Numerical Methods in Industrial Forming Processes, Numiform 2004*, 2004.

[5] 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.

[6] G. Henkelman and H. Jonsson. Long time scale kinetic Monte Carlo simulations without lattice approximation and predefined event table. *J. Chem. Phys.*, 115:9657–9666, 2001.

[7] 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.

[8] LAMMPS molecular dynamics simulator, 2005. URL: www.cs.sandia.gov/~sjplimp/lammps.html.

[9] E. Lidorikis, M. E. Bachlechner, R. K. Kalia, A. Nakano, P. Vashishta, and G. J. Voyiadjis. Coupling length scales for multiscale atomistics-continuum simulations: Atomistically induced stress distributions in Si/Si3N4 nanopixels. *Physical Review letters*, 87:086104, 2001.

[10] S. Namilae, N. Chandra, and C. Shet. Mechanical behavior of functionalized nanotubes. *Chemical Physical Letters*, 387:247–252, 2004.

[11] NITRD/CRA Workshop on the road map for the revitalization of high end computing (www.cra.org/Activities/workshops/nitrd), 2003.

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

[13] A. Srinivasan and N. Chandra. Latency tolerance through parallelization of time in scientific applications. In *Proceedings of the 18 th International Parallel and Distributed Processing Symposium, Heterogeneous Computing Workshop*. IEEE, 2004.

[14] A. Srinivasan and N. Chandra. Latency tolerance through parallelization of time in scientific applications. *Parallel Computing*, (to appear – see www.cs.fsu.edu/~asriniva/papers/parcorev.ps meanwhile), 2005.

[15] 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.

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

[17] C. Y. Wei, K. J. Cho, and D. Srivastava. Tensile strength of Carbon nanotubes under realistic temperature and strain rate. *Physical Review B*, 67:115407, 2003.

[18] B. I. Yakobson, M. P. Campbell MP, and C. J. Brabec. High strain rate fracture and C-chain unraveling in Carbon nanotubes. *Computational Materials Science*, 8:341–348, 1997.