Scalable Time-Parallelization of Molecular Dynamics Simulations in Nano Mechanics

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

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

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

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 small, requiring a large number of iterations to simulate realistic time spans. Conventional parallelization is not very effective for this. We recently introduced a new approach to parallelization, where data from related prior simulations are used to parallelize a new computation along the time domain. In our prior work, the size of the physical system in the current simulation needed to be identical to that of the prior simulations. The significance of this paper lies in demonstrating a strategy that enables this approach to be used even when the physical systems differ in size. Furthermore, this method scaled up to almost 1000 *processors with close to ideal speedup in one case, where conventional methods scale to only* $2-3$ *processors.*

1 Introduction

Nanotechnology impacts a range of fields that includes materials, electronics, pharmacy, and health care, among others. Molecular dynamics (MD) is widely used to simulate the behavior of physical systems in such applications, with resolution at the atomic scale. However, a serious limitation of MD is its inability to simulate phenomena that take long time, for reasons given below. MD computations involve the iterative solution of an initial value problem, with time steps of the order of a femto (10^{-15}) second. So, even after a few million iterations, 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 is not sufficient to get a realistic picture of the behavior of a physical 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, and in simulations of biological molecules.

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 some type of decomposition of the state space of a system. This 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 time 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 data-driven time parallelization approach, which we called *guided simulations*, where results from related simulations were 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 solve an initial value problem. We observe that, typically, the current simulation is not the first one that is being performed; *usually, the results of many related prior simulations are available. We use results from one such related simulation (which we call the*

[1]Some MD applications, involving computation of physical or thermodynamic properties, are trivially parallelizable, with the results of independent simulations being averaged. Such a scheme is not possible in general, such as in the problem we consider.

base simulation) to predict the state of the current simulation at the beginning of each time interval. 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 in § 3.

We summarize our prior work, as well as related work by others, in § 4. In contrast to our prior work [6], the prediction strategy in this paper does not require that the sizes of the physical systems in the base simulation and the current one be identical, but it uses knowledge of the physics of the problem, to a certain extent. The ability to simulate a different sized system is useful, because it permits a single run, with a smaller tube size (and smaller span of time), to enable a number of more realistic simulations that use larger physical systems (for longer spans of time [2]).

The effectiveness of our strategy is demonstrated in tensile tests of CNTs, where the length of the CNTs are 1, 1.2, 1.6, and 2 times that in the base simulation, and using different simulation parameters (pulling speed) than in the base. § 5 gives the details of these numerical experiments, along with other tests that validate their 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 span 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 is the largest attained in classical molecular dynamics 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. One important application of CNTs is in nano-composites, where CNTs are embedded in a polymer matrix. In such applications, it becomes important to determine the mechanical properties of the CNT. One important simulation/experiment is the *tensile test*, in which one end of the CNT is pulled at a constant velocity, while the other end is fixed. The response of the material is characterized by the *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. 4 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 a multi-scale finite element code to determine the effect of the polymer matrix on the CNT, and vice-versa. Another important property is the strain at which the CNT starts to break.

2.2 Molecular Dynamics

We describe MD in the context of the CNT application, though it is, of course, a more *general technique, which can be used for simulating the behavior of a set of atoms or molecules*. 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 based on the positions of the atoms. 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. Accuracy and stability considerations limit Δt to the order of a femto second (10^{-15}s), in MD.

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 four 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 parallelization, and so the time required, even in a parallel computation, is the order of 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 in the order of an hour. It is assumed that the stress-strain relationship determined at this higher strain rate is the same

[2]The time t required to simulate a CNT with N atoms when pulled at velocity v until it starts breaking is roughly proportional to N^2/v. The reason for this is that the CNT's length is proportional to N, and so the number of time steps required to reach a fixed value of strain (which is defined later) is proportional to N/v. Furthermore, the time required for each iteration is proportional to N.

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 [8]. On the other hand, if we were able to parallelize the computation efficiently on a large number of processors, then we could reach the desired time span with more realistic strain-rates too. We wish to use the existing high-velocity simulation results to perform relatively more realistic lower-velocity simulations on a large number of processors. MD simulations in nano-mechanics are often performed with simulation parameters that are more extreme than is realistic, due to the time-scale problem mentioned above. Consequently, similar prior results are often available, and so our approach can be extended to a larger class of applications.

3 Time Parallelization through Guided Simulations

We recently introduced the general idea of guided simulations to parallelize along the time domain [5, 6]. We describe it below, for completeness, but specialized to the CNT application. A more general description is provided in [5]. We then describe the specific prediction strategy used in this work.

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 use* 500 *or* 1000 *time steps per time interval in our experiments.*

Fig. 1 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.

In Fig. 1, processor i, for each $i \in \{1 \cdots 4\}$, predicts (as described later) 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. Note that processor 1's initial state is known to be accurate. So its computed results for time t_1 are accurate. In Fig. 1, since these results are close to the predicted state for time t_1, the predicted state for time t_1 too is accurate, which implies that the computed state on processor 2 for time t_2 too is accurate, because it started from an accurate initial state. The computed results in processor 2, in turn, are close to the predicted results at time t_2, implying that the computed results on processor 3 for time t_3 are accurate. 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 may have started from incorrect initial states. 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 prior ones. Note that the outputs of the simulation are always states computed using MD, and not predicted states.

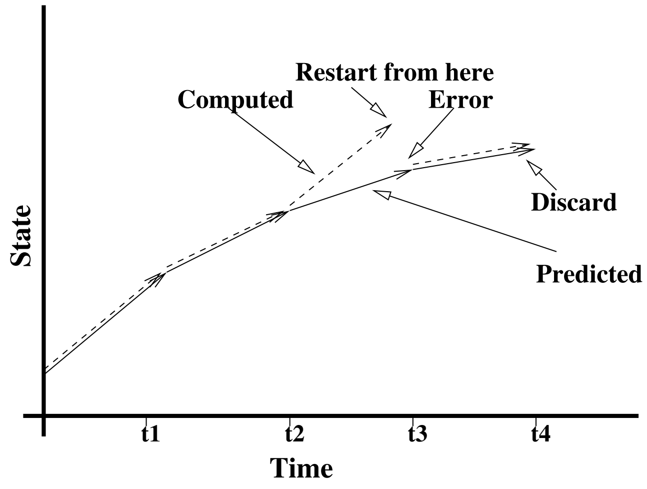

Figure 1. Schematic of parallelization of time.

Note the following: (i) Processor 1's accurate MD result is correct, since it always starts from a state known to be accurate. So the computation always progresses. (ii) All the processors must use the same predictor; otherwise verification of prediction at time t_i on processor i does not imply that the prediction for initial state at time t_i on processor $i+1$ was correct. (iii) The answers given will 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.

Algorithm 3.1: TIMEPARALLELIZE(Initial State 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 $i \leftarrow i + k$
```

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 time t in the current simulation is related to time $t/10$ in the base.

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

$$x_{t+\Delta t} = x_t + \dot{x}_{t+\Delta t}\Delta t, \quad (1)$$

where $\dot{x}_{t+\Delta t}$ is the actual slope dx/dt at some point in $[t, t+\Delta t]$. We do not know the value of $\dot{x}_{t+\Delta t}$, 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 $\dot{x}$ 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 the tensile test problem.

Let $\dot{x}_{t+\Delta t} \approx \sum_i a_{i,t+\Delta t}\phi_i(x_t)$. Once we have performed an accurate simulation for time t, we know the actual $\dot{x}_t$ 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 $b_{i,t}$. 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 $a_{i,t+\Delta t} \approx b_{i,t+\Delta t} + R_{i,t+\Delta t} = b_{i,t+\Delta t} + a_{i,t} - b_{i,t}$. 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. Note that if values of $b_{i,t}$ are not available, only the values of $a_{i,t}$ are used.

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 is 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[3].

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, and so if we look at the states of a physical system in equilibrium, at two different points in time, it is 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 in the accurate simulations either. Details on the physical justification for our verification procedure are given in [7]. Here, we give a high-level overview of our criteria. We determine the difference in positions of corresponding atoms in the predicted and computed states. If the average difference is below a threshold, defined by the difference expected for equivalent systems, then the difference is considered acceptable. Similar thresholds are set for the maximum difference between any two corresponding atoms, and for the potential and kinetic energies of the system.

3.4 Time Required

The overheads of parallelization, such as prediction, and communication, are small compared with that of the "useful" computations (as performed by the sequential algorithm). Each processor performs MD computations for its time interval. For example, consider a time interval of 1000 time steps and a 1000-atom CNT. Sequential computations for 1000 time steps require around 13 seconds on an Intel Xeon processor running at around 3.2 GHz and around 46 second on a 375 MHz IBM POWER 3 processor.

The parallel overheads are due to time loss in prediction, communication, and file I/O from disk. Each processor performs two predictions. This requires two file reads, and nine least squares computations (three coordinates each, for the base simulation 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 that erred. 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 operations is much smaller than the computation time and so the efficiency is very high, even on a large number of processors. For example, on an Intel Xeon cluster at NCSA, the least square and other prediction related computations takes $\approx 10^{-3}$s, file read $\approx 0.01 - 0.2$s seconds, 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 ($\approx$ 13s) for simulating a single time interval.

[3]MD does not automatically preserve temperature. So a process called "thermostating" is performed, which modifies velocities, using random numbers, to keep temperature constant. This occurs both in the sequential and in the parallel algorithms.

4 Related Work

4.1 Prior Work

We introduced the idea of guided simulations for time parallelization of scientific applications in [4, 5]. We also demonstrated the effectiveness of the technique in practice, using a CNT computation with tensile test as an example in [5]. The prediction strategy in the current work improves on that in [5]. This enables the computation to scale efficiently on up to 990 processors, in contrast to 50 processors in the prior work. Furthermore, in the current work, a 10m/s base simulation predicts a 1m/s simulation, whereas the two velocities were much closer in the previous work.

In [6], we showed how basis functions can be selected in a more mathematical manner. The basis functions there needed data from a CNT (or any other physical system being studied) of the same size as the current simulation. In contrast, the predictor in the current work does not assume that. The amount of data broadcast in this predictor is greater than that in [6]. However, the actual efficiency is slightly higher, even though the current experiments are run on a larger number of processors, perhaps because there is one extra send/receive in [6].

4.2 Other Approaches

Works on parallelization of MD calculations on CNTs, as well as several publications on parallelizing MD computations on other physical systems in general, are summarized in [7]. Good efficiency is typically obtained when the granularity is of the order of at last 10 ms per time step in a parallel run. This does not give much scope for conventional parallelization for the application we consider, which requires the order of 10 ms per time step even on a single processor.

Time parallelization using the Parareal approach [1] (which does not use prior data) has been proposed as an alternative to conventional parallelization. However, the speedup and efficiency obtained have been limited, even in the model problems considered. We described its limitations in detail in [5].

In the 1980s and 90s, time parallelization using waveform relaxation [2], and various variants of this, were well studied. However, these techniques, which are based on ODE theory and can be considered generalizations of Picard iterations, had limited impact due to their slow convergence. (The slow convergence is a feature of the sequential algorithm.)

5 Experimental results

The tensile tests in this paper are performed by keeping one end of the tube 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. The Tersoff-Brenner potential was used in the MD simulations, and a fourth order Nordsiek scheme for time integration.

We used results from an MD simulation of a CNT containing 1000 atoms, conducted at a temperature of $300K$, whose output had been recorded every 100 time steps, up to a total of $330,000$ time steps, as the base case. The predictor used $\beta = 0.5$. 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 started to break. Results after the CNT starts to break are not of use in our application, though it is still interesting to observe them.

5.1 Speedup Results

The timing results are based on wall clock time. In the report of the speedup results, 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.

Speedup results[4] for a 1000-atom CNT[5], using a time interval of 1000 time-steps, on the *Tungsten* Xeon cluster at NCSA are shown in Fig. 2. 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 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 computing nodes ran in dedicated mode for the timing results. We can see that speedup is almost the ideal linear curve up to 990 processors on the Xeon cluster. The processors never erred in the course of the simulation (up to the point where the CNT started to break), and so loss in speed was only due to the overheads of prediction, communication, and reading the base simulation results from disk, as discussed in § 3.4.

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 of the sequential code (consequently, counting only the "useful" floating point operations, and not those for prediction). 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 [3] is 1.4 MFlops/atom, where we use the data for the run on the largest number of processors reported. We are not aware of any classical molecular dynamics simulation attaining a greater flop rate per atom than ours in a real application, and have reason to believe that ours is the largest.

Similar simulations on up to 500 processors on the *Eagle* IBM SP3 machine at Oak Ridge National Lab yielded efficiencies 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 was 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.

The simulations of CNTs of 1200, 1600, and 2000 atoms were performed on the *Seaborg* IBM SP at NERSC, using

[4]The speedup results compare the parallel time with that for an inherently sequential code, which does not have any of the overheads of the parallel code. Comparing with the parallel code run on a single processor would yield marginally higher speedups.

[5]Even though the number of atoms are identical to that in the base simulation, the simulation parameters differ, and the prediction scheme does not use knowledge of fact that the lengths are equal.

a time interval of 500 time-steps[6]. This system consists of compute nodes with 16 375MHz IBM Power 3+ processors each. Fig. 3 shows the speedup results. In the 1200 and 2000-atom simulations, the prediction is always sufficiently accurate, but minor errors during initialization on 200 processors caused a slight drop in efficiency to around 95%. With the 1600-atom simulation, there was one set of prediction errors toward the middle of the simulation, which caused the efficiency to drop to around 79% on 200 processors. The efficiencies for the other cases were well over 90%. The scalability is less than that observed when the base and current simulations are on CNTs of identical length. However, the speedup is still substantial. Note that the sequential computation needs the order of a week of computing time on the 2000-atom simulation, and so the benefit obtained from time-parallelization is considerable.

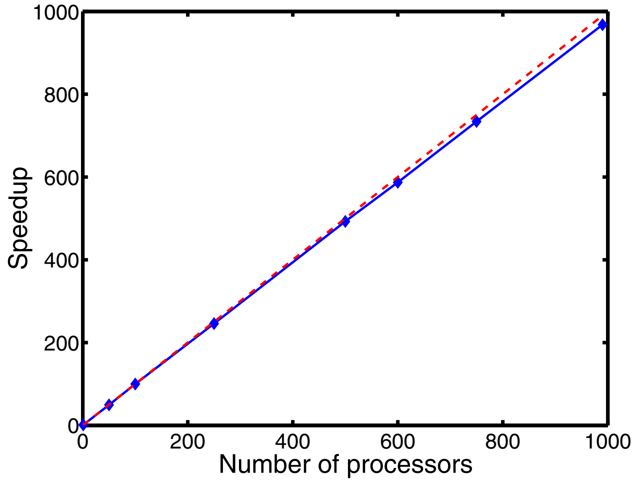

Figure 2. Speedup curve. The dashed line shows the ideal speedup, and the solid line shows the observed speedup. The flop rate is 420 **GFlops on** 990 **processors.**

5.2 Validation

To verify that the errors in our scheme did not propagate, we compared the stress-strain results from the parallel run with the exact sequential run. Fig. 4 shows the plot of stress versus strain, which is a material behavior of practical interest, for a 200 processor run on a 2000-atom CNT. The time-parallel results agree very well with the exact sequential results, even in the non-linear region, up to the point where the CNT starts to break. The time at which the CNT starts to break is also determined as with the sequential run, at a

[6]A smaller time interval size was required for initialization to be accurate on large numbers of processors.

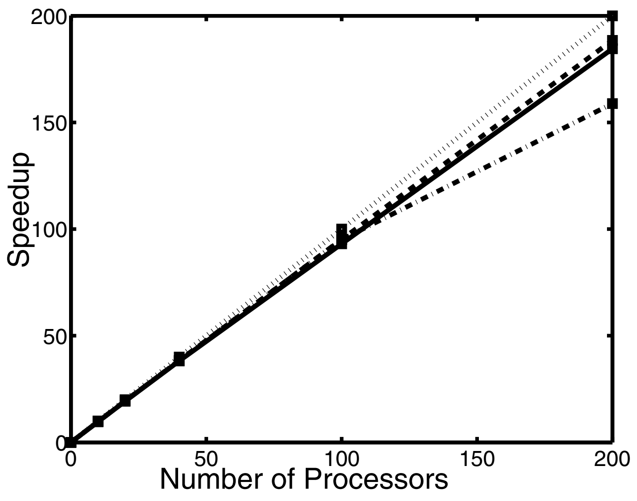

Figure 3. Speedup curve. The dotted line shows the ideal speedup. The dashed, dash-dotted, and solid lines show the speedups on 2000**,** 1600**, and** 1200 **atom simulations respectively, on the IBM SP at NERSC.**

strain of around 0.19 (0.1894 for sequential versus 0.1931 for parallel.)

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. Results beyond the point where the CNT starts to break are not of any practical use in our application.

The above results give empirical evidence for the stability of our method. Details on both numerical and physical reasons for this are given in [7].

6 Conclusions

We have demonstrated the effectiveness of a new approach to time parallelization, namely, using guided simulations, on a large number of processors in a practical application. Compared to our earlier work, this strategy can be used on physical systems of different sizes than the prior simulations. Furthermore, we scaled it to a larger number of processors, which is two orders of magnitude larger number of processors than with conventional parallelization. The flops per atom rate is also higher than that for any other MD simulation that we are aware of. Since MD is used in 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 can provide $10^4 - 10^5$ processors. If this approach

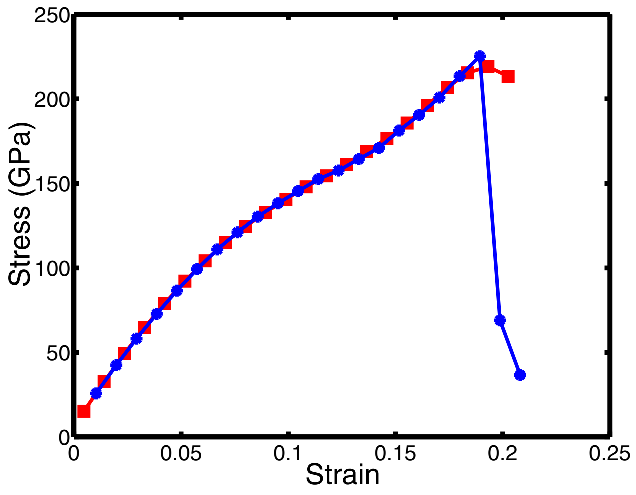

Figure 4. Stress-strain plot. The solid line denotes the *exact* **curve from sequential simulations on a** 2000**-atom CNT, with circles indicating the data points. The squares show the data from the parallel simulation on 200 processors, up to the point where the CNT starts to break (the stress decreases with increase in strain).**

can be scaled to such large systems, then it will be possible to perform MD simulations to several orders of magnitude longer time than currently feasible. We expect this approach to be useful in other applications involving hard matter, as is typically the case in nano-mechanics. However, with the soft-matter encountered, for example, in biological systems, more challenges have to be overcome. The reasons for this depends on physics issues that are outside the scope of this paper.

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 for realistic experimental conditions. We also wish to scale the computations to an order of magnitude larger number of processors. These will require better initialization, and also some method of validating the results, without using the sequential algorithm directly. Yet another future work is to perform predictions for more complex problems. Basis functions should be developed so that they correspond to different types of physical phenomena that may be experienced by the system. *Time parallelization can also be combined with spatial parallelization;* instead of one processor computing for one time interval, a group of processors, that distribute the atoms across the group, can be used to simulate each time interval. This will yield a code that improves on the best available conventional code.

Acknowledgments

This work was funded by NSF grant # CMS-0403746. We thank ORNL, NERSC 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 were first tested. A.S. gratefully acknowledges his debt to Sri S. S. Baba, but for whose help and inspiration, this work would neither have 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] C. W. Gear. Waveform methods for space and time parallelism. *Journal of Computational and Applied Mathematics*, 38:137–147, 1991.

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

[4] 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.

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

[6] A. Srinivasan, Y. Yu, and N. Chandra. Application of reduced order modeling to time parallelization. Proceedings of HiPC 2005. *Lecture Notes in Computer Science*, 3769:106–117, 2005.

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

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