# Latency and fault tolerance through parallelization of time, with applications to nano-materials simulation

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

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

## Abstract

*Emerging computing environments, such as the Grid, and also massively parallel processors, promise enormous raw computational power. However, effective use of such platforms is often difficult, because conventional spatial decomposition leads to fine granularity, resulting in high communication overhead. We recently introduced the concept of* guided simulations *[10] to parallelize along the time domain. Here, we use the fact that typically results of other simulations of closely related problems are available. In this approach, we automatically and dynamically determine a relationship between old simulations and the one being performed, and use this to parallelize along the time domain. Though specific techniques depend on the nature of the application, the general algorithm can be applied to a wide class of problems. In our earlier work, we demonstrated the validity of this approach by applying the technique for two simplified problems. In this paper, we consider a real application involving molecular dynamics simulation of nano-materials. Here we report on a specific scheme that we have developed for a computational nanotechnology application, and demonstrate its effectiveness. As it turns out, one of the key stumbling blocks to the ubiquitous use of molecular dynamics in nanotechnology is the extremely small time steps (*$10^{-15}$*sec) needed, which makes it difficult to access long time scales. The present work aims to alleviate that problem by enabling effective use of parallelism even for small physical systems. Further, we also show how this approach can lead to an inherently fault-tolerant algorithm – that is, the algorithm progresses despite process failures, without the need for checkpointing.*

## 1 Introduction

Emerging computing environments promise much raw computing power. For example, massively parallel machines with a hundred thousand processors are expected by the end of the decade, and ten-thousand processor systems are predicted to be widely available. The Blue Gene machine, with around 65,000 processors, is due for release in 2005. The Grid too promises abundant computing power. On the other hand, the need for such power is also increasing, as computing becomes an increasingly important component of science and engineering. However, most current algorithms do not scale well to large numbers of processors, often due to high communication overhead caused by fine granularity. While certain techniques, such as Monte Carlo, can be effective in such environments, as can applications where the loosely-coupled nature of the scientific phenomena involved leads to coarse granularity, a wide variety of applications do not fall in these classes.

Fault tolerance is also an important factor in such environments, since the likelihood of some processor failing during the course of a computation increases rapidly with the number of processors. Checkpointing-based solutions are popular, but are not expected to be feasible for large systems of the near future, because the time for checkpointing is expected be larger than the mean time between failures. Thus the algorithm should have some natural resilience, based on which it can progress sufficiently even if processors fail, without depending on checkpointing.

Our work addresses both these issues. The motivation for our work arises from our interest in computing nano-material properties, using Molecular Dynamics (MD)[1] simulations. MD based simulation, in possible

[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. The latter set of problems are usually formulated with periodic boundary conditions on all sides (microcanonical ensemble). Several independent simulations starting from different states are then performed, and their results averaged in some sense. This leads to a trivially parallelizable solution scheme. Such a level of homogeneity is not possible when mechanical loads are applied or when fracture can occur at any point depending on the local properties.

combination with other continuum based techniques (e.g. Finite element methods), is expected to play a very key role in the nanotechnology revolution. Computational simulations based on MD is the analytical and design tool of choice. However, its use severely restricted by the extremely small time increment (of the order of femto ($10^{-15}$) seconds) in each step of the computation, since this leads to a large number of iterations being required. For example, realistic design of devices should be orders of magnitude higher (days to a few years, $10^7$ sec). Any method that attempts to solve this limitation of time scale is expected to have tremendous impact, and this has been identified as an important challenge in nanoscale simulations and computational materials science [4, 9].

Multiscale techniques, where MD is combined with continuum based methods, can reach longer time scales. However, MD simulations used here still often fall short of the desired time scales by a few orders of magnitude. Note that the computational effort arises from the long time scales involved, rather than from the size of the physical systems. Massive parallelism can enable us to reach longer time scales, if the efficiency is high. However, conventional spatial decomposition is not effective since it leads to fine granularity. Our approach is to parallelize along the time domain. This leads to both, latency tolerance, and a naturally fault tolerant algorithm, as we shall show below.

There are two aspects to this work. (i) The general approach based on the concept of guided simulations, which we introduced recently. In this approach, a relationship is automatically and dynamically determined between a previously completed simulation and the simulation being performed, and this relationship is used to parallelize the current simulation. (ii) The specific techniques that are used to determine a relationship between different simulations. These are application dependent to a large extent. In our earlier work [10], we demonstrated the validity of our approach on some simplified toy problems, and also divulged its significance when applied to real problems. In the present work, we consider a real application. Techniques applicable to the simplified problems have limitations in a real application, and we report on a different technique we developed for this problem, and demonstrate its effectiveness.

Though the mathematical details of the technique varies from problem to problem, the conceptual framework remains unaltered, and is not restricted to the specific application being considered. So we first describe the general features of the class of relevant scientific problems in § 2. We describe our approach to time parallelization, using guided simulations, in § 3. This is followed by a discussion of our application in § 4, and the description of our new technique for predicting the relationship between simulations in § 5, followed by experimental results in § 6. We then discuss fault-tolerance in § 7, and finally summary our conclusions and future work in § 8.

## 2 The class of scientific applications considered

A large class of scientific application follows the general paradigm that one tracks the evolution of the state of a system with time, where the change in state is modeled through a differential equation, in addition to boundary/initial conditions. For example, we can describe the model generally as: $dS/dt = f(S,t)$, where $S$ is the state, $t$ is time, $f$ is some function of $S$ and $t$, and $dS/dt$ gives the rate of change of the state. The physical domain where the equation is applicable is prescribed, along with the conditions at the boundary, in addition to constraints specified at $t = t_0$. This continuous problem is discretized by computing the state at discrete points in time, and the state at some time $t_i$ is obtained as: $S_i = S_{i-1} + g(t_{i-1}, t_{i-2}, \cdots, S_{i-1}, S_{i-2}, \cdots)$, where $g$ is some function, possibly implicitly, of previously observed states and times (typically, of the time increments $t_i - t_{i-1}$). If the initial state $S_0$ is known at some time $t_0$, then the above recurrence can be used to iteratively determine the states at other points in time. In order to achieve the required accuracy and ensure stability of the solution, the time increments need to be sufficiently small (with the exact details being very much dependent on the numerical scheme used). In this paper, we discuss a computational scheme of a more restrictive form, which is commonly encountered, where the time points are all equally spaced, and also the next state depends only on the previous state, as shown in Eqn. 1. This is done just for convenience of presentation; our techniques are applicable to the more general situation too. We use another notational convenience in Eqn. 1 below. The time points $t_i$ will refer to times at which we are interested in observing the state. In order to go from $t_{i-1}$ to $t_i$, we may actually need to apply several time steps of a differential equation solver.

$$S_i = F(S_{i-1}). \quad (1)$$

The time taken per iteration clearly increases with increase in the the size of the state and with increase in the number of time steps for which the computation has to be performed. When the solution is computationally intensive, parallel computation can be used to reduce the time taken.

As an example, the above form of the differential equation can in fact describe Newton's laws of motion. MD simulations apply an iterative process that results from the numerical solution of the differential equation to simulate $N$ atoms, having 3 position and 3 momentum components each. The maximum incremental time step is dictated by the vibrational frequency of the atoms, which is in the range of femto seconds. In order to reach even a microsecond of simulation, which is still well below the realistic time periods of devices, about a billion iterations are required.

# 3 Guided simulations for time parallelization

## 3.1 Time parallelization

In a parallelization along the time domain, different processors compute the state of the physical system at different points in time. The difficulty here is that the computation of the states is defined iteratively, with the results for time $t_{i-1}$ being needed for computing the state at time $t_i$, as seen from Eqn. 1. So it is not clear how each processor can independently and simultaneously compute the state for a different point in time. Our approach is to attempt to dynamically determine a relationship between the current simulation and prior simulations 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[2]. We give a schematic of our approach in Fig. 1.

*Prediction:* The most important feature of our strategy is our ability to predict the state, which serves as the starting point for each processor. The predictor should be both, accurate most of the time, and much faster than the verifier. We exploit the fact that there exists (or will come to exist) a large data base of simulation results for the same class of problems with slightly different input conditions. Further, we postulate that there will be a tangible relationship between the results of these simulations that can be effectively put to use. In this paper, we will make use of a single prior simulation, which we call the *base* simulation. We will call the simulation being performed as the *current* simulation. We dynamically determine a relationship between the base and current simulations, and this relationship is used to predict the state of the current simulation at some future point in time.

If there was 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. This implies that automatically determining such a relationship is also not easy. So, we usually do not determine the state directly from the base, but track how changes in the state of the base relates to changes in the state of the current simulation. We then use this to predict the change in the current simulation's state over a fixed interval of time. Such a scheme requires us to be able to predict only over a shorter time interval. 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. We shall give specific details for a class of problems in § 5. (It is possible that such prediction will yield new insights into the relationships between different simulation parameters, eventually leading to new scientific understanding, though a discussion of that is beyond the scope of this paper.)

*Time parallelization:* Fig. 1 describes the time parallelization procedure. We will let $S_i$ denote the state at time $t_i$ resulting from an accurate computation that starts at the given start state $S_0$, $T_i$ a predicted state at time $t_i$, and $\hat{S}_i$ a state at time $t_i$ that results from an accurate computation that starts from a possibly inaccurate predicted state at time $t_{i-1}$. We choose the number of time points to be much larger than the number of processors. This is realistic, since we are dealing with long time simulations, and so a sufficient number of time points can be selected. Processor $i, i \in [1, P]$ predicts states $T_{i-1}$ and $T_i$ at times $t_{i-1}$ and $t_i$ respectively, with $T_0$ defined as $S_0$. It then performs an accurate simulation, starting from the predicted state at time $t_{i-1}$ up to time $t_i$, yielding $\hat{S}_i = F(T_{i-1})$. In order to verify the accuracy of the predicted state at time $t_i$, processor $i$ then compares $\hat{S}_i$ and $T_i$. If they differ sufficiently, then the start state for processor $i+1$ was incorrect and we say that processor $i+1$ *erred*, and its result $\hat{S}_{i+1}$ has to be discarded. In fact, the results for all processors $\geq i+1$ are discarded, since they have not been verified to be correct. For example, since $\hat{S}_{i+1}$ is incorrect, we cannot be sure that $T_{i+1}$ is correct even if $\hat{S}_{i+1}$ is sufficiently close to it. Consequently, we cannot be sure that $\hat{S}_{i+2}$ is correct, and so on. Note the following: (i) Processor 1 can never err, since its accurate computation, $\hat{S}_1$, started from the known initial state $S_0$. So the computation always progresses. (ii)

[2]We also wish to mention the parareal approach to time parallelization, which served as a motivation for our work. We have summarized that approach and discussed its limitations in [10], and shall not describe it further here. However, we wish to mention one limiting factor. It requires a sequential computation at each iteration of the process, and this eventually limits speedup and decreases efficiency. In fact, experimental results in [1, 7] showed poor efficiency (around 1% often).

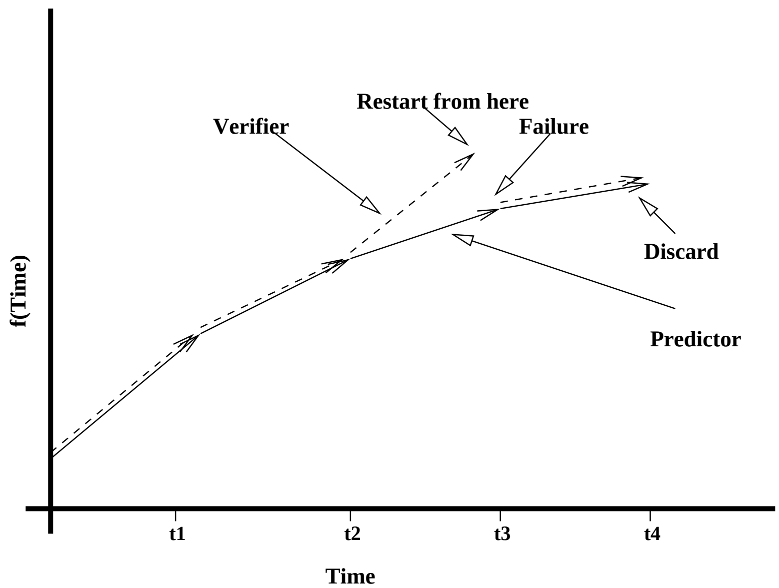


Figure 1: Schematic of parallelization of time. Processor $i, i \in [1, P = 4]$ somehow predicts the states at time $t_{i-1}$ and $t_i$ (with the state at time $t_0$ being a known initial state $S_0$), using the results of an old simulation, which we call the *base*, and its relationship to the current simulation. Then processor $i$ performs accurate 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, since the predicted state for $t_3$ was inaccurate, computations for subsequent points in time too have to be discarded, since they might have started from an incorrect start state. The next iteration 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.

Difference between $T_P$ and $\hat{S}_P$ does not lead to any computation being discarded, since $T_P$ is not used as the initial state of any accurate computation; the comparison between $\hat{S}_P$ and $T_P$ is performed just for the purpose of updating the predictor. (iii) All the processors must use the same predictor; otherwise verification of $T_i$ at processor $i$ does not imply that the prediction for $T_i$ at processor $i+1$ was correct.

Based on the difference between $T_i$ and $\hat{S}_i$, processor $i$ updates its prediction scheme (usually, some parameters to the prediction algorithm need to be updated). The processors perform an all-reduce to determine an index $k$, such that the smallest ranked processor that erred has rank $k+1$. If no processor erred, then $k$ is set to $P$. The prediction parameters on processor $k$ (the last processor with validated $\hat{S}$) are broadcast to all processors. We also broadcast the state $\hat{S}_k$ computed by this processor. The reason for this is that our predictor implementations actually predict a change in state, and add this to a given state $S$ to predict the new state, as shown in algorithm 3.1. In order for the predictions to be consistent across processors, the state $S$ too needs to be identical on all processors. The above procedure is then repeated, starting with the known, verified, state $\hat{S}_k$ at time $t_k$.

**Algorithm 3.1:** PREDICT(State $\hat{\mathrm{S}}_\mathrm{i}$, at time i, Desired time j)

**if** $j == i$
  **then** return $\hat{S}_i$

$$\textbf{else} \begin{cases} \Delta S \leftarrow \text{ a (possibly inaccurate) prediction} \\ \quad\text{for } \hat{S}_j - \hat{S}_i \\ \text{return } \hat{S}_i + \Delta S \end{cases}$$

Algorithm 3.2 shows the steps in this procedure. In its description, we assume the existence of the following functions. (i) *Predict* predicts the state of the system at a desired point in time, by first, somehow, predicting the change in state. The prediction mechanism is application dependent to a large extent, and we shall describe a specific predictor for our application in § 5. (ii) *AccurateComputation* performs an accurate simulation, starting at a given start state and applying the recursion defined by $F$ in Eqn. 1. (iii) Once we observe the error between the prediction and the accurate computation, we update the predictor by tuning its parameters. This clearly depends on both, the application and the predictor. (iv) *IsDifferenceTooLarge* determines whether the difference in states is too large for us to consider the prediction acceptable. This too is clearly application dependent. The above steps show that within the general paradigm of time parallelization through guided simulations, there are several application dependent algorithmic components that need to be developed. (v) *AllReduce* and *Broadcast* can be performed using MPI. We note that the reduction to find the minimum of $\text{Next}_j$ determines the rank of the largest indexed processor whose accurate computation has been verified as valid. We also note when $i$ is sufficiently close to $m$, we do not need all the processors to actually work in the *for* loops nested within the while loop. Strictly, those loop should go from 1 to $\min(P, m-i)$, but we do not specify this in the algorithm in order to maintain clarity.

*Numerical stability:* 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 accurate computation should not propagate it. If it does propagate errors, then the scheme would have a problem even without the prediction process, since it is not stable to errors inherent in the numerical process. Assuming a stable and accurate scheme, the new start states will be accurate, as will the results that are output, if we output states that result from the accurate verifier computation.

*Time complexity:* Let $S$ denote the speedup, $E$ the efficiency, $P$ the number of processors, $m$ the number of time intervals simulated, $n$ the number of iterations of the *while* loop in algorithm 3.2, $\tau_a$ the time for performing the accurate computation over one time interval, $\tau_p$ the time to perform two predictions, determine whether two states are sufficiently close, and update the predictor, $\tau_r$ the time to perform an all-reduce of an integer on $P$ processors, and $\tau_b$ the broadcast time of the state and predictor parameters on $P$ processors.

The sequential time is given by $m\tau_a$ and the time on $P$ processors by $n \times (\tau_a + \tau_p + \tau_b + \tau_r)$. The speedup is thus given by:

$$S = \frac{m\tau_a}{n \times (\tau_a + \tau_p + \tau_b + \tau_r)} = \frac{m}{n} \times \frac{1}{1 + \frac{\tau_p + \tau_b + \tau_r}{\tau_a}} \quad (2)$$

In the equation above, the first fraction $m/n$ varies between 1 and $P$. The former value is obtained if all the predictors fail. Since the computation always progresses, only one time interval is successfully simulated each time leading to $n = m$. The latter value is obtained if all the predictors are accurate (and if $P$ divides $n$). In that case, $n = m/P$. Thus this ratio depends on the accuracy of the predictor. If we assume that $m \gg P$ and that iterations of the while loop in algorithm 3.2 lead to successful completion of $\alpha P$ time intervals on the average, where $\alpha$ is between $1/P$ and 1, then $n = m/(P\alpha) \Rightarrow m/n = \alpha P$.

We note that the second fraction is close to 1 if $\tau_p + \tau_b + \tau_r$ is relatively small compared with $\tau_a$. We have flexibility with respect to choosing $\tau_a$ for the following reason. Each time interval usually involves several steps of an accurate differential equation solver. We can make the time interval larger, implying that more steps of the differential equation solver need to be performed, and thereby increase $\tau_a$ until it becomes sufficiently large. We cannot make $\tau_a$ arbitrarily large in practice, because it is often difficult to predict accurately over long time periods, and so increasing $\tau_a$ will, at some point, start reducing $\alpha$. Conversely, better predictors will enable us to increase $\tau_\alpha$, thereby increasing the efficiency, by improving both the fractions in Eqn. 2. If we are interested in simulating for a fixed period in time, then increasing $\tau_a$, by increasing the number of time steps of the accurate solver, will decrease $m$. This does not impose an important restriction to the increase, because the desired time scale is extremely large in applications involving long time scales. In fact, achievable time scales are several orders of magnitude smaller than that desired in a large class of important applications (which serves as a motivation for this work). The times $\tau_r$ and $\tau_b$ increase with the number of processors. Under many simple models, they would increase proportional to $\log P$. However, due to their dependence on hardware, it is often more useful to determine them from benchmark results. In the application for which we show results in this paper, the time for these operations is very small relative to $\tau_a$, and the fraction $(\tau_p + \tau_b + \tau_r)/\tau_a$ is smaller than $10^{-3}$. When this happens, and it is quite realistic, the speedup is given by

$$S \approx \frac{m}{n} \quad (3)$$

**Algorithm 3.2:** TIMEPARALLELIZATION(Initial State $S_0$, Number of processors P, Number of time steps m)

```
i ← 0
Ŝ_0 ← S_0
while i < m
  do {
    for each processor j ∈ [1, P], in parallel
      do {
        T_{i+j-1} ← Predict (Ŝ_i, i, i + j − 1)
        T_{i+j} ← Predict (Ŝ_i, i, i + j)
        Ŝ_{i+j} ← AccurateComputation(StartState = T_{i+j-1}, StartTime = i + j − 1, EndTime = i + j)
        UpdatePredictionParameters(CurrentParameters, Ŝ_{i+j}, T_{i+j})
        if IsDifferenceTooLarge(Ŝ_{i+j}, T_{i+j})
          then Next_j ← j
          else Next_j ← P
      }
    k ← AllReduce(Next, min)
    Processor k broadcasts Ŝ_{i+k} and its prediction parameters
    for each processor j ∈ [1, P]
      do in parallel i ← i + k
  }
```

If $m \gg P$ then we can give the speedup in terms of the empirically determined parameter $\alpha$ as

$$S \approx \alpha P \tag{4}$$

That is, the speedup is limited primarily by errors in the prediction. Note that efficiency in this case is given by $E \approx \alpha$, and varies between $1/P$ and 1. Note that the computation is always load balanced if $m \gg P$, since each processor performs an independent simulation of the same physical system.

*Spatial parallelization:* We wish to note, as an aside, that we can combine temporal and spatial parallelization; instead of one processor performing computations for the entire system at a given point in time, a group of processors will be responsible for this. For example, if the system can be parallelized efficiently through spatial decomposition on ten processors, and we have a thousand processors available, then groups of ten processors each can be assigned the task of computing for a specific time step. So time parallelization will lead to an improvement in addition to that obtained through purely spatial parallelization.

## 3.2 Prior results

We reported two different experiments in [10]. In one, a "toy" example, the situation simulated is similar to that shown in Fig. 2. However, the exact material behavior was specified by a formula, instead of being computed using molecular dynamics. Ten points were considered, and the material was assumed to be one-dimensional. A simple predictor was used to compare the base and the current simulation, which differed in their material properties. This indicated that prediction was feasible, if, indeed, there was a simple relationship between the material responses, even though this relationship was not known to the simulation code. Another experiment used results of actual MD computations on a CNT (which is three-dimensional). However, both the predicted and actual simulations were pre-computed, and the state was defined using only the positions of atoms. The predictor was used to check if it could perform an accurate prediction, but with a limited definition of the states. States were considered equivalent if the average of the difference in atom positions was below a certain, realistic, threshold. This demonstrated that prediction was possible even in an MD setting. However, the definition of state was not sufficient. While the predictor performed well, it had the following limitations. It required that the number of atoms in the base and the current simulations be identical, and it also required that the data for each atom in the base also to be stored, for all the time points that we needed. We would generally expect this scheme to be used when we have a database of several simulation results available. Storing the coordinates of each atom would involve much space. So we would like to use minimum information from other simulations that will permit efficient prediction. This minimization of information will also help if we want to dynamically compare a large number of simulations and determine a suitable base simulation to use. The new predictor, used with the real application, overcomes these limitations, and also performs better in terms of both accuracy and speed.

# 4 A nanomaterials application

## 4.1 Application

Development of nanomaterials require a clear understanding of the phenomena that take place at the scale of a few atoms. Though MD is recognized as a practical computational tool to simulate atomic behavior, there are two fundamental problems that limits its use – its inability to simulate (i) large space and (ii) long time scales. The former can be addressed, to a certain extent, through the use of multiscale models, though it is still an important research topic.

The more serious limitation of MD is its inability to model 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 a million time steps we can only simulate up to a nanoseconds, at which time most of the thermal, mechanical, physical, or magnetic events have not even started. This is a very critical issue that needs to be addressed if the computational component of nanotechnology research is to contribute to its progress. Thus we need methods to extend the simulation times.[3] Massive parallelism with the order of $10^3$ or more processors can yield a similar order of magnitude improvement in time scales attained, if reasonable efficiency can be maintained. The aim of this paper is to demonstrate a technique that lead to much higher efficiency than a purely spatial parallelization.

## 4.2 Specific problem

We consider a Carbon Nanotube (CNT) that has one end fixed, and the other end moved at a constant velocity $u$ as shown in Fig. 2. Different simulations use different velocities. This is a common type of simulation to test the strength and elastic modulus of a material [2]. We wish to use as small a nanotube as possible, since our aim is to achieve as large a time scale as possible. Our application (a multiscale model in which the CNT properties, describing the nanoscale behavior of the CNT, are input to an FEM simulation) needs at least 1000 to 3000 atoms for this purpose [3]. This provides an example of an important class of applications where a small system needs to be simulated to long time scales [8].

[3] Some time extension schemes, such as hyper MD [11] and Kinetic Monte Carlo [6], are topics of much interest. However, these techniques have limitations which make them not applicable to a large class of problems, such as the one mentioned above. Further discussion of the limitations of these techniques is given in [1, 4].

## 4.3 Equivalence of states

MD simulations bring an interesting issue – that of determining the equivalence of two dynamic states. This issue needs to be addressed since we need to determine if the predicted and accurate computations are sufficiently close. MD simulations track the vibrational motion of an atom around its mean position, as shown in Fig. 3. Even when nothing interesting is happening to the physical system, if we look at the states at two different points in time, it will be unlikely that the atoms will be in the same positions, since their vibrations cause them to behave more like a stochastic system. So we cannot expect the predicted state to have atoms in the same positions as the 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, e.g. microcanonical ensembles.

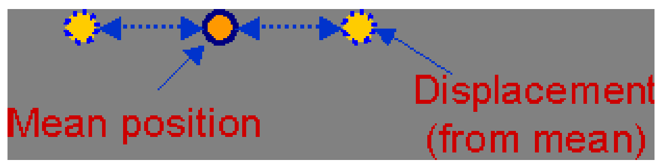


Figure 3: Atoms vibrate around some mean position even when nothing interesting is happening to the system.

We 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. (i) We will compute the distance between positions for each atom 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. (ii) 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

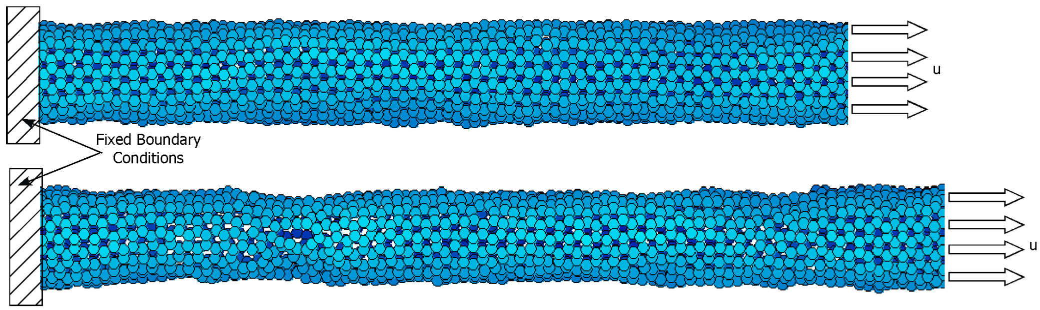


Figure 2: Schematic of the problem. A Carbon nanotube is held fixed at one end, and pulled at the other end at a constant velocity. The higher figure shows an initial configuration, and the lower figure the configuration at a different point in time.

maximum deviation. So we will consider other factors. There are two types of energies that are important. (iii) 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. (iv) 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 required to be below a threshold. These threshold are shown in the graphs in § 6.

# 5 A predictor for the CNT application

In this predictor, we predict the change in each coordinate 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. Let $x_t$ represent a coordinate at time $t$. Using two terms of the Taylor's series, we have

$$x_{t+\Delta t} = x_t + \dot{x}_{t+\Delta t}\Delta t, \tag{5}$$

where $\dot{x}_{t+\Delta t}$ is the actual slope $\delta x/\delta t$ 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 can experience.

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 $b_{i,t}$. We will assume that the values of $b_{i,t}$ have been recorded, and are available for the duration of the base simulation. The coefficients $a$, on the other hand, are computed and 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 the difference for 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 $a$ 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 $\beta$ is the weight assigned to the latest value. We note that the point in time $t$ mentioned here is the latest point at which the actual coefficients are available, and need not coincide with the $t$ in Eqn. 5. Of course, more sophisticated prediction can be made, using curve fits that use the last few estimates of the difference. 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 in time for each point, with the constant of proportionality being a function of its normalized coordinates for our experiments. This is sufficient for a moderate time interval, but fails after that. Of course, the prediction-verification process detects the errors and corrects for it, but with some loss in efficiency.

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.

# 6 Experimental results

## 6.1 Experimental parameters

*Experimental parameters:* Both the base and the actual simulation pull one end of a Carbon nanotube containing 1000 atoms at a constant velocity. 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 is more computationally intensive than many other potentials, but is known for its accuracy in Carbon nanotube calculations. One end of the tube is kept fixed by forcing around 170 atoms at that end to remain stationary. The tube is pulled at a fixed rate in the $z$ direction by forcing around 170 atoms at the other end to increase their $z$ coordinate value at a rate of $u$ Angstroms per time step, where $u$ is 0.00005 for the base simulation, 0.0000625 for the simulation being performed, and the time step is 0.5 femto seconds. The remaining around 660 atoms are simulated using MD, with the end atoms providing the boundary conditions. Note that with just 660 atoms, we cannot achieve much speedup through spatial parallelization, since the granularity will become very fine. Most published results on codes using the Brenner potential require around $1000 - 2000$ atoms per processor for good efficiency. Our spatial parallelization code permits smaller granularity than this, but even then does not yield good efficiency on more than 2 processors.

We first performed an accurate MD simulation for the base case (since it is needed for the prediction) at a temperature of around $10K$, and recorded the output of every 1000 time steps, over $200,000$ time steps. The new simulation was performed at a temperature of $300K$, thereby testing the effectiveness of prediction when temperatures are different. The predictor used $\beta = 0.5$, and the basis functions used were the first two terms of the polynomial basis, 1 and $x$.

A parallel run was simulated, with Matlab used as a front end to perform the analysis, and calling a Fortran code that performed the MD simulation. (Though parallelism was only simulated, performance results on the IBM SP3 clearly show that loss in efficiency is caused by errors in prediction, and not by issues such as load balance or even the communication cost. For example, the time taken to perform one iteration of the accurate simulation (that is, 1000 steps of the MD computation) is of the order of 10 seconds . In contrast, an all-reduce of an integer using MPI on an IBM SP3, even with 100 processors, takes just around 0.0005 seconds, while a broadcast takes around 0.005 seconds for the state size involved. While we can expect it to take longer on the grid, it will still be much smaller than the computation time. The predictor takes the order of $10^{-3}$ seconds. Thus the overheads are three orders of magnitude smaller than the basic accurate computation, and so we are justified in ignoring these.)

*Results:* We first show in Fig. 4 the difference between the predictor and the verifier for each of the four quantities that define the equivalence of two states. This simulation is performed on 50 processors. The results are always below the threshold, indicating that errors do not occur. However, we can see that the errors keep decreasing over time, indicating that the predictor learns better about the relationship between the two simulations.

In Fig. 5, we show the speedup, based on the results of a simulation of a parallel run. As mentioned earlier, the low overhead associated with this parallelization makes it clear that this will be the actual speedup obtained. The efficiency is almost 1 for up to 50 processors.

*Discussion:* With 100 and more processors, the speedup does not increase, partly due to the fact that the number of processors is close to the number of time steps available (when $P$ is close to $m$, many processors are idle when we are simulating time close to $m$). Longer simulations might lead to better speedup, and so we also plot the theoretical speedup that is yielded by the factor $\alpha$ of Eqn. 4. While the speedup obtained by this simulation is itself substantial (for example, if we combine it with spatial decomposition, we can get a speedup of around 85 on 100 processors , based on

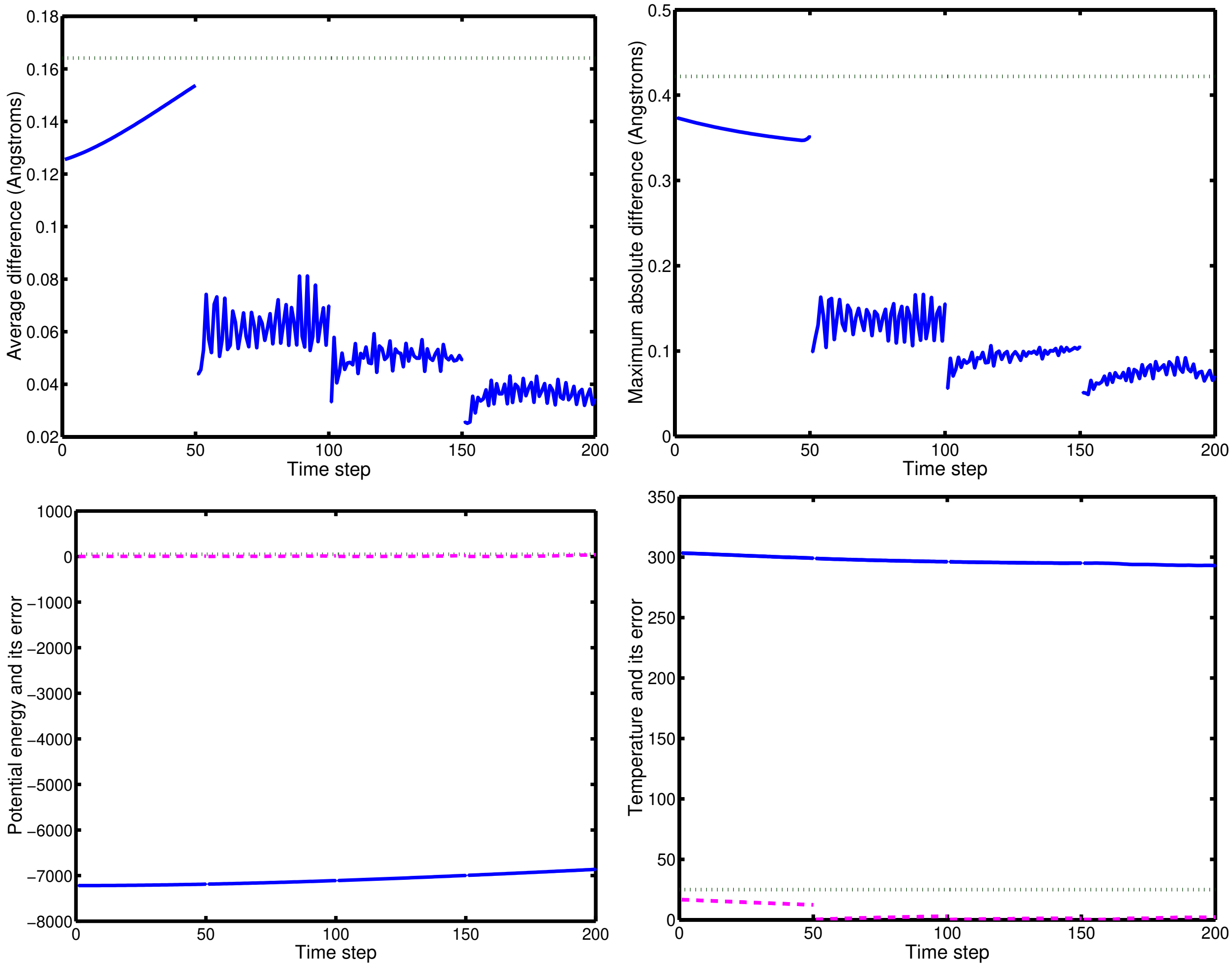


Figure 4: Error comparing the predicted state with the accurate verifier computation with $P = 50$. The green dotted lines indicate the error threshold. All the results are below this threshold. Each solid blue line and magenta dashed line represents one iteration of the while loop in algorithm 3.2. The blue lines in the bottom two figures represent the quantity mentioned, and the magenta lines the errors. The green line in the bottom left figure is barely visible, but is just above the magenta line. Note that the 200 time intervals reported here represent a total of $200,000$ steps of MD simulations, since each time interval involves 1000 MD steps.

the efficiency results on our spatially parallelized code), it does suggest that even greater speedup can be obtained. The major reason for the speedup being limited is the difficulty of predicted over long periods of time. The results on 50 processors indicate that we can predict over a period of $50,000$ time steps (50 time intervals). If we made each time interval smaller, say 100, then we could probably go up to 500 processors through purely time parallelization, with the communication and prediction overhead adding just around 2% overhead. With two processors simulating each time interval, we would be able to achieve about 80% efficiency on 1000 processors. In terms of the discussion of $\alpha$ earlier, we have chosen $\tau_a$ larger than it needs to be, causing $\alpha$ to increase.

*Limitations of the experiments:* One of the limitation of the experiment is that it is a simulation of a parallel run, rather than a real parallel run. However, since the computation cost per iteration is three orders of magnitude larger than the communication and prediction costs, it is clear that the speedup and efficiency would be maintained in a parallel run. The loss in efficiency primarily due to to errors in prediction.

# 7 Fault tolerance

Time parallelization can also be used to produce algorithms that are inherently fault tolerant, without the need for checkpointing. Instead of the failure being

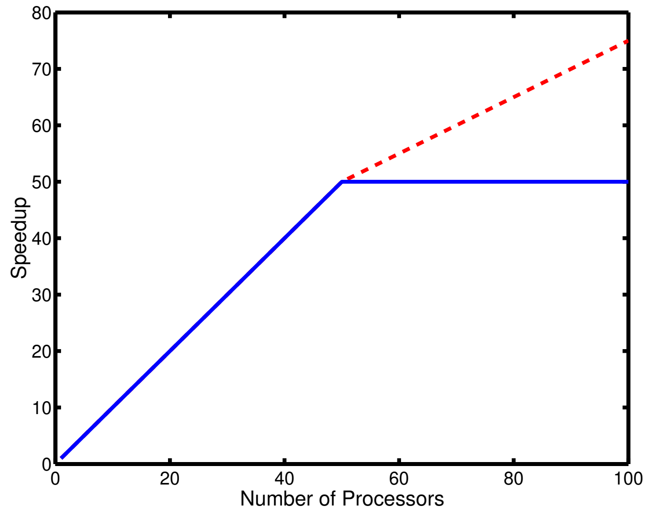


Figure 5: The solid line gives the speedup. The dashed line give the theoretical speedup based on Eqn. 4.

due just to an error in prediction, it will also be due to failure of processes or processors. However, in the case of these types of failures, we will not need to discard the result of subsequent time steps, since they have not really been invalidated. We just need to "fill in" the missing point in time too, to check the validity of future points in time already computed. So it does not lead to an additional loss in efficiency. (The speedup will, of course, diminish, since there are fewer processors available after failure.) This assumes some software that detects failure of nodes, and communicates that to the processors. For example, FT-MPI's [5] semantics permit such use. We perform a reduction at each iteration. If the reduction fails due to a processor failing, all other processors will know about this, since they are all involved in the reduction operation.

We can perform these computations also in a master-worker paradigm. Global communication is avoided, with communication being performed with the master, who assigns time intervals to be simulated to each worker. Since communication costs are small related to computation cost (which is performed only by workers), this is feasible. Process failures will need to be communicated only with the master, since it is the only one that communicates with a worker. This, of course, assumes that the master does not fail. In fact, our simulation of the parallel run can be considered a master-worker one, in which the Matlab front end served as the master, that assigned work to the Fortran MD code that performed the MD computations. The limitation was that the workers were run on a sequential machine.

# 8 Conclusions and future work

*Conclusions:* We have proposed a strategy which can enable high latency environments, such as the Grid and massively parallel processors, to be effective for problems where a typical spatial decomposition will lead to too fine a granularity. We have demonstrated its effectiveness on an important and real application of great technological significance.

This method can be combined with spatial parallelization for even greater effectiveness. Of course, further work needs to be performed to extend the applications for which this technique can be used. We outline some ideas below.

*Future Work:* In the short term, we intend implementing this scheme on a real parallel machine, combine it with spatial parallelization, and simulating more complex problems for longer time periods. These are fairly straight-forward extensions of this work. We give more important research issues on which we are working, with a longer time horizon in mind.

In the current experiments, the base and the current simulations were performed for the same times scales. In many current applications, we cannot reach the time scales desired, and so a base simulation reaching such a time scale is not available. We need to determine a suitable base simulations whose short-time behavior can be a reasonable predictor of the long-time behavior of a simulation of interest. 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. We can develop similar "surrogates" for time and perform "base" simulations. It will be acceptable for these simulations to be occasionally wrong (for example, missing an important physical phenomenon), since the verifier checks and corrects for this.

Coming up with better predictors is another aspect that we are investigating. For example, it will be useful to determine different basis functions for different types of physical phenomena. It is possible for different portions of the CNT to be experiencing different phenomena, and we can use clustering techniques to identify a region experiencing a certain phenomenon, and use appropriate basis functions for it.

Another goal is to determine suitable base simulations for the current simulation automatically from a large set of available results. It is possible that the appropriate base simulation will change during the course of the simulation, and identifying appropriate ones based on the pattern of behavior exhibited by the current simulation would be a useful tool.

Such techniques may also enable us to analyze a repository of simulation results and search for relationships between simulation parameters. Such relationships, if then proved through a more fundamental understanding of the science, will reduce the need for many other simulations.

# 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] N. Chandra, S. Namilae, and C. Shet. Local elastic properties of carbon nanotubes in the presence of stone-wales defects. *Physical Review B*, 69, 2004.

[3] 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 Methds in Industrial Forming Processes, Numiform 2004*, 2004.

[4] 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.

[5] E. Gabriel, G. E. Fagg, A. Bukovsky, T. Angskun, and J. J. Dongarra. A fault-tolerant communication library for grid environments. In *Proceedings of the 17th Annual ACM International Conference on Supercomputing (ICS'03), International Workshop on Grid Computing*, 2003.

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

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

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

[10] 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.

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