Latency tolerance through parallelization of time in scientific applications

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

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

Abstract

Distributed computing environments, such as the Grid, promise enormous raw computational power, but involve high communication overheads. It is therefore believed that they are primarily suited for "embarrassingly parallel" applications, such as Monte Carlo, and for certain applications where the loosely-coupled nature of the science involved in the simulations leads to a coarse grained computation. In a typical application, this is not feasible. We discuss our solution strategy, based on scalable functional decomposition*, which can be used to keep the computation coarse grained, even on a large number of processors. Such a decomposition can be attempted through a variety of means. We will discuss the use of time parallelization to achieve this. We demonstrate results with a model problem, and then discuss its implementation for an important problem in nanomaterials simulation. We also show that this technique can be extended to make it inherently fault-tolerant.*

1 Introduction

Motivation: Computation is becoming an increasingly important component of science. Due to increases in computational power, sufficiently realistic solutions can be obtained for an increasing class of problems of practical and commercial importance. However, for realistic solution of a wider class of problems, much greater computational power and better modeling techniques are required. An example of such a class of applications, which serves as a motivation for our work, is the development of nanomaterials and products, based on a fundamental understanding of the physical phenomena that take place at the scale of a nanometer. While the computational power required for such applications is enormous, emerging computing environments, such as the Grid and also massive parallelism with the order of a hundred thousand processors, promise sufficient raw computational power in the near future. The problem here is that it is difficult to make effective use of such systems. Thus new computational paradigms too are required to make effective use of such environments, and that is the focus of this paper.

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. 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. 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: $S_{t_i} = S_{t_{i-1}} + g(t_{i-1}, t_{i-2}, \cdots, S_{t_{i-1}}, S_{t_{i-2}}, \cdots)$, 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_{t_0} is known at some time t_0, then the above equation can be used to iteratively determine the states at other points in time. In order for accuracy and stability of the solution, the time increments need to be sufficiently small (with the exact details being very much problem dependent). 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; our techniques are applicable to the more general situation too. We use another notational convenience in Eqn. 1. 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.

$$S_{t_i} = F(S_{t_{i-1}}). \tag{1}$$

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.

Conventional parallelization: We next provide a simplistic overview of parallelization, and present the limitations of conventional techniques. Conventional parallelization is by domain decomposition (which we will also refer to as spatial parallelization), where the physical system is distributed across the processors, and each processor is responsible for computing the next state for a portion of the system. However, this computation normally requires access to portions of the system residing on other processors, and so communication is required. Communication costs typically increase proportional to the surface area of the portion of the system on each processor, while the computation cost increases proportional to the volume. Since the surface to volume ratio decreases with increase in volume, the communication overhead becomes relatively small for large systems, and such parallelization is effective. Since parallelization is often required for large system, such a technique is often effective. However, it is not effective when the communication latencies are high compared to the computation cost, such as in the following situations: (i) when we want to compute a small system for a very long time period, (ii) when massive parallelism causes the volume per processor to become very small, or (iii) in a grid like environment, where the communication cost is high. This has been recognized as an important challenge to be addressed, and this paper proposes a solution strategy to this issue, using an important application in computational nanotechnology as an example. Our solution strategy can also introduce "natural" fault tolerance at an algorithmic level (without the need for checkpointing), which is another important issue in the emerging computing environments.

Scalable functional decomposition: Functional decomposition, where the data is replicated, but each processor performs a different, independent, action on the data, makes the computation more coarse grained, and consequently reduces the communication overhead. It is thus attractive for the three cases mentions above (small physical systems, massive parallelism, and high communication-cost environments). It is, however, difficult to find many such independent operations that can be simultaneously performed on the system. Usually only a small constant number of such operations can be obtained, and this does not scale with the number of processors.

Our solution strategy is based on the notion of scalable functional decomposition. Here, we develop techniques that enable increasing number of independent operations, as the size of the parallel system increases. It is possible to accomplish these through a variety of means, such as appropriate time stepping. For example, extrapolation methods for solving differential equations involve solving over a time interval with different step sizes. Simulations with different step sizes can be performed independently on a different processors, leading to a functional decomposition.

Parallelization of time: In this paper, we discuss the use of time parallelization to achieve such a scalable functional decomposition. Here, different processors compute the state of the physical system at different points in time. Since each of these computations is an independent set of operations on the same physical system, we have a functional decomposition that scales with the number of processors. 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. The parareal idea introduced by Baffico et. al. [1] proposed an approach to deal with this problem. However, the speedup and efficiency obtained were limited, for reasons that we explain later. In this paper, we present our approach to dealing with the bottlenecks faced by Baffico's approach. It is based on the idea of *guided simulations*, where the results of old simulations can be used to speed up the computation of future simulations.

Outline: The outline of the rest of the paper is as follows. We first present the parareal approach and bottlenecks faced in its use in § 2. We then present our approach using guided simulations in § 3. Our approach is very general, though the particular application that we are interested in uses molecular dynamics (MD). The use of our technique with MD is complicated by certain "randomness" in the MD solutions. So, before discussing the solution with MD, we present results on a "toy" model problem without randomness in § 4, both the illustrate the concept and also to demonstrate its promise in applications without randomness. This is followed by a discussion of our application, and application of our strategy to MD simulations that arise here, in § 5. Experimental results are then presented in § 6, followed by a summary of our conclusions and future work in § 7.

2 Time parallelization through the parareal approach

The parareal approach: Time parallelization in this strategy is accomplished through a predict-verify-correct approach, as illustrated in Fig. 1.

Let us suppose that we want to compute the state for P time steps. The computation will involve P processors with ranks in $[1, P]$. Each processor wishes to start simulating with the state of the system at some time t_{i-1} and evolve it to the correct state at some time t_i, by performing the computation specified by F of Eqn. 1. If they can do this, then each processor would have performed simulation for a different time interval, and would incur no communication overhead. The difficulty is that normally obtaining the

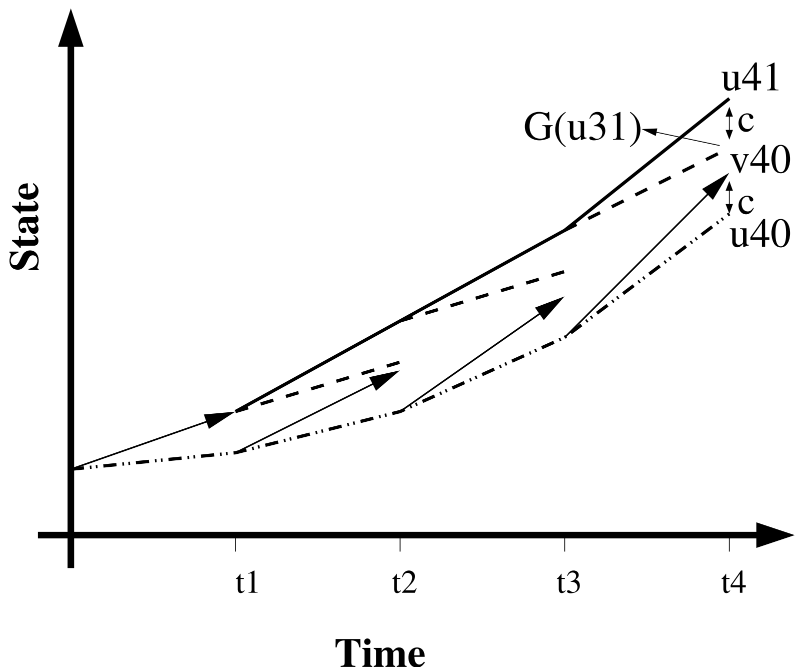

Figure 1. Schematic of parallelization of time using the parareal approach, when the state is a single scalar. The dash-dotted line shows the first sequential step, where a "predictor" is used to predict the states at each point in time. The solid lines with arrows denote an accurate verification step, which is performed simultaneously, and independently, on each processor. If these lines had fallen along the same curve as the predictor, then all the predicted states would be correct, and the computation can be terminated. Otherwise, the difference between the verifier and the predictor is computed. We have denoted it in this figure, for time t_4, by c. The predictor is again applied sequentially (shown by the dashed line), but it is corrected by the difference computed in the previous step, before it is applied iteratively again for the next time step. The solid line shows the resulting prediction. These steps are repeated until convergence.

correct start state at time t_{i-1} will require the results from processor $i-1$, thereby making the computation sequential. Instead, one can use a predict-verify-correct sequence (several iteration of this sequence may be required).

Iteration 0 is started by first computing an approximate solution to Eqn. 1 (we will later describe some methods for such prediction). We refer to this as the predictor. We define the process more formally below. If the known initial state at time t_0 is denoted by u_0, then we define $u(0, j) = u_0$ for all j. The predicted state at time t_i can be obtained by the recursion $u(i, 0) = G(u(i-1, 0)), i \in [1, P]$ (the second index 0 denotes iteration 0 of the predict-verify-correct sequence). Here, G is an approximation to F of Eqn. 1, such that it can be computed fast. This phase of the computation is sequential.

The algorithm next verifies if the predicted states are accurate, in parallel. Processor i computes the state at time t_i, by starting from the predicted state at time t_{i-1} and performing an accurate computation as described by Eqn. 1. The resulting states are denoted by $\hat{u}(i, 0) = F(u(i-1, 0)), i \in [1, P]$. If $\hat{u}(i, 0) = u(i, 0)$ for all i, then each processor started from the correct state during its verification step, and so the final results too are accurate, and the computation can terminate. If not, then the accurate computations do not yield a correct solution, since some of their start states were incorrect.

If the predictions were not accurate, then a correction step is applied. Processor i computes the error $c(i, 0) = \hat{u}(i, 0) - u(i, 0)$. This concludes one iteration of the predict-verify-correct sequence. Note that the state $\hat{u}(1, 0)$ produced by the verifier for time t_1 is accurate, since it started from the correct, known, initial state $u(0, 0) = u_0$. So the computation makes progress even in the worst case, for this iteration.

Iteration 1 predicts the states by first sequentially predicting the state at time t_i by the recursion $u(i, 1) = G(u(i-1, 1)) + c(i, 0)$. That is, the state at time t_i is predicted by applying the inaccurate predictor to the newly predicted state at time t_{i-1}, but then corrected by the error that was encountered the previous iteration. Note that at this time, the predicted state too at time t_1 is accurate, since $u(1,1) = G(u(0,1)) + c(i,0) = G(u(0,0)) + c(i,0) = G(u(0,0)) + \hat{u}(1,0) - u(1,0) = G(u(0,0)) + F(u(0,0) - G(u(0,0)) = F(u(0,0))$. Since $F(u(0,0))$ is the accurate solution for time t_1, the computation has made progress. The verification and correction phase are now applied as before, to complete this iteration. If the solutions have not converged, then the computation repeats until convergence. Note that after iteration 1, $\hat{u}(2,1) = F(u(1,1))$ too is correct, since it was determined through an accurate computation starting from a correct initial state $u(1,1)$. Similarly, it can be seen that the computation progresses each iteration, finding the exact solution after P iterations in the worst case, and after one iteration in the best case. Thus this approach always produces accurate solutions; a good predictor will improve the performance but not affect correctness, since the verifier step ensures that an incorrect computation is not accepted.

We mention a couple of schemes by [1] to come up with suitable predictors. One is to use a larger time step size in the solution of the differential equation. Yet another is to use a different model, which can be computed fast, but is not very accurate (for example, it might miss certain physical phenomena that the more accurate model can capture; however, if such physical phenomena occur infrequently, then the prediction will be acceptable most of the time).

Limitations of this technique: Let the number of iterations for convergence be denoted by k, and the ratio of time taken by the verifier to the time taken by the predictor by r. If we ignore communication costs, then the speedup is given by:

$$Speedup = \frac{1}{k}\frac{Pr}{P+r}, \tag{2}$$

and the efficiency by

$$E = \frac{1}{k}\frac{r}{P+r}, \tag{3}$$

One might wonder about the justification for ignoring communication costs, when one of the motivations for this approach is computing environments with high communication cost. Since time parallelization is attractive only when the time period $t_P - t_0$ is very large, for realistic numbers of processors, the time intervals $t_i - t_{i-1}$ to be simulated on each processor too will be fairly large, and so the accurate computation F too will take much time. Thus the communication costs may often be small in comparison. However, communication cost increases linearly with the number of processors (due to the sequential computation of the predictor), and may not always be negligible. Our technique will overcome this limitation.

Another factor that limits the speedup is r, the ratio of the time taken by the verifier to the time taken by the predictor. If r is very large, and convergence is attained in one iteration, then close to linear speedup is obtained. However, as P increases, the speedup is bounded above by r, even if convergence takes place in one iteration. So it is important for r to be very large in order to get good speedup and good efficiency. We argue later that it is often difficult to attain this with the techniques of [1], and we propose an alternate approach to ensure that r is very large.

The number of iterations for convergence clearly affects the efficiency significantly. Note that as the iterations proceed, even though states for many of the points in time would have already been computed correctly, the processors responsible for those points are still involved in the computation. This can be easily changed so that we do not compute for processor i, if states for $t_j, j \leq i$ have already been computed efficiently. However, the processors are still idle, and so the efficiency is still low. Our technique will overcome this problem too.

Application of the parareal technique to certain "toy" molecular dynamics simulations in [1] yielded results that were not very good, for reasons mentioned above. For example, we mentioned that a larger time step can be used in the predictor. However, the value of r attained from such a technique would be rather low in a real application. For example, in nanomaterials simulations, [5] carefully choose different time steps for different types of forces. But only a factor of 3 or 4 improvement in time is obtained by such a method. If the predictor is to be reasonably accurate, then this limits the value of r tremendously. If the predictor is not very accurate, then the number of iterations for convergence affects both, the efficiency and the speedup. A toy problem in [1] used $r = 1000$ to get a speedup of 130. However, it may be difficult to achieve this in a real simulation, for reasons mentioned above. Furthermore, the efficiency even in this toy problem was just 1.3%, even ignoring communication costs. There is one another problem with the use of a larger time step. If predictors with larger time steps are fairly accurate, then it is quite probable that an adaptive time stepping scheme would be very effective for this problem, and so a good sequential algorithm would take about the same time as the predictor, which is done sequentially anyway. Thus the speedup compared to a good sequential algorithm would not be very high. Other toy MD simulations performed by [1] use different models for prediction, which we feel is a more promising strategy. The speedups obtained are quite modest – around 8, with an efficiency of 25%, ignoring communication costs. Maday et.al. [4] use the parareal approach with a quantum control problem, and get a speedup of around 14, with the efficiency being around 1.5%.

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.

3 Guided simulations for more effective prediction

Our technique attempts to remove some of the bottlenecks encountered in the parareal approach, which we hope will enable effective realization of this promising idea. In particular, we will reduce the communication cost and the sequential predictor cost, reduce the inefficiency caused by idle processors, and increase the ratio r. The major emphasis will be on producing fast and effective predictors using the concept of guided simulations, where the results of old simulations are used to speed up the simulation being performed.

We first describe a slight modification to the parareal simulation schematic of Fig. 1, which will enable us to better describe our strategy, apart from being more efficient. This is presented in Fig. 2.

Guided simulations: 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 much of the time, and much faster than the verifier. We use data from related simulations, determine a relationship dynamically between the the current simulation and old ones, and use it to predict the state of the new simulation. Furthermore, at each time step, we can determine the error in the predictor, and modify the predictor so that it "learns" more about the relationship between the simulation being performed and old ones. We shall give specific details for specific problems in the next two sections. The computational benefit here is that the processors independently predicts the start state, thereby making this part of the computation too parallel. Furthermore, communication cost no longer increases linearly with the number of processors. Also, since the predictor is not based on performing an actual simulation (instead, the old simulations provide the information of an accurate simulation), it is very fast, leading to an r sufficiently high, that it is no longer a bottleneck to obtaining good speedup and efficiency.

We also need to deal with the issue of avoiding idle processors. This is fairly straightforward to handle. 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. The simulations are then performed as shown in Fig. 2. Processor $i, i \in [1, P]$ predicts the states at times t_{i-1} and t_i, with the state at time t_0 being the known initial state u_0. It then performs an accurate simulation, starting from the predicted state at time t_{i-1} up to time t_i, in order to verify the accuracy of the predicted state at time t_i. If the prediction was not accurate, we say that the computation *failed* for processor i. The processors perform a reduction to determine the smallest ranked processor that failed. The computation is repeated with the start time being the end time on the processor that failed. (It is the end time on the processor that failed, rather than the start time, since the verification step will produce a correct value for the state at the end time on the first processor to fail. Either this state will need to be broadcast to all the processors, or some information for updating the predictor will need to be broadcast. If no processor failed, then the broadcast step may be omitted, depending on the details of the prediction scheme.) The correction step of the original parareal method is replaced by letting the predictor learn about the relationship between the current simulation and old ones that are being used for prediction. Such learning can occur even when the computation does not fail.

Ignoring communication costs is more realistic here for practical values of P, since it increases only logarithmically with P. (Of course, as P approaches infinity, it will theoretically be the dominant factor, and reduce speedup by a factor of $\log P$.) In the equations below, we need to account for failures. This is analogous to the number of iterations in the original parareal method. However, not all failures are equal, since a failure of the first processor forces us to discard a large number of results, while failure of the last processor does not force us to discard any result. We lump this effect into a single failure factor α, which will be computed after observing the results of the simulations. This factor will vary between 1, when there is no failure, to P when there is a failure on the first processor each time. The speedup and efficiency are given below, assuming that communication costs can be ignored.

$$Speedup = \frac{1}{\alpha}\frac{P}{1+1/r} \approx \frac{P}{\alpha}, \tag{4}$$

$$E = \frac{1}{\alpha}\frac{1}{1+1/r} \approx \frac{1}{\alpha}. \tag{5}$$

4 A model problem and experimental results

We first consider a "toy" model problem illustrated in Fig. 3, and a simple predictor. A material is subjected to a "pull-out" test, where one end is kept fixed and the other end is moved at a constant velocity. In order to make the predictor a little challenging, we assume that the actual material behavior is given by: $x = x_0 + (x_0/L_0)^{\alpha} L_0 * v * t$, where x is the position at time t of a point that was initially at x_0. L_0 is the initial length of the sample, v is the velocity with which one end is being pulled out, and α is a property of a material. In a real application, use of this formula

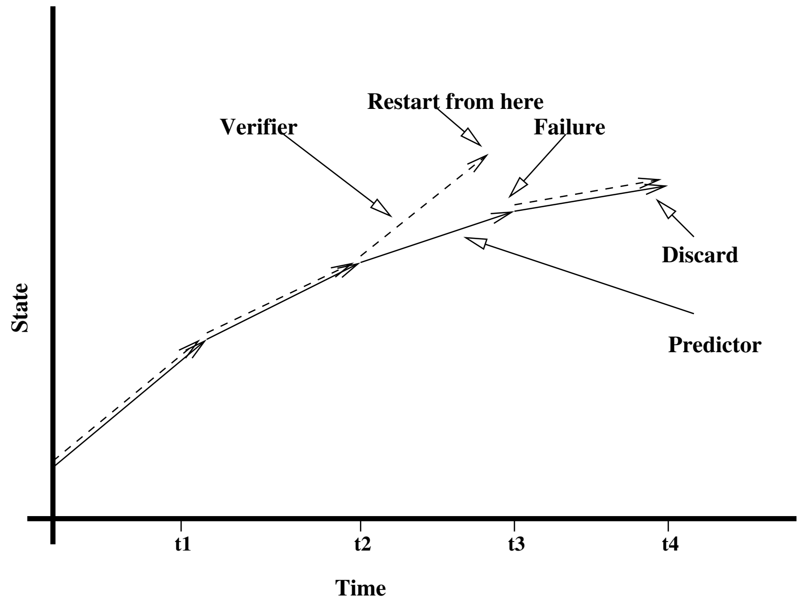

Figure 2. Schematic of parallelization of time. The states at time $t_i, 1 \leq i \leq 4$ are somehow predicted using the results of old simulations, and their relationship to the current simulation. Then processor i performs accurate simulations starting from the predicted state for time t_{i-1} up to time t_i, and verifies if the predicted state at 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. 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 prediction step can be used to improve the predictor by better determining the relationship between the current simulation and old simulations.

would be replaced by a time-consuming verifier simulation, which will be performed in parallel.

We will use a single "old" simulation, which we call the *base* simulation, to predict the states of the new simulation. We choose the following parameters for the base simulation: $L_0 = 2$ (in some "dimensionless" unit), $v = 0.1$ $\alpha = 1.5$, and we track 10 points, other than the origin, which are initially uniformly spaced on the sample. This simulation is carried out to time units = 10, and we store the results every 0.1 time units. Note that the size of the sample increases by 50% in this time period, and also that $\alpha \neq 1$ causes points that were initially uniformly spaced to become non-uniformly spaced later. In the new simulation, all the parameters are identical to the base simulation, expect that α is taken to be 2.0.

The simulation conceptually proceeds as follows[1]. We will assume 10 processors. Each processor simulates a time interval of 0.1. Each processor predicts the solution at the end of the time interval, and then runs an accurate simulation to verify the results. We have replaced the accurate simulation in our tests by the use of the formula given above. Next, it updates predictor information based on the relationship between the base and accurate simulation results. If there is no failure, then the algorithm will repeat the process starting from the time on the last processor. Otherwise it will start from the time of the first processor that failed.

The relative error is defined to be the ratio of the norm of the difference of the accurate and inaccurate position vectors, to the norm of the accurate position vector. Fig. 4 compares the base simulation to the new simulation, just to show that they are very different. The relative difference is calculated similar to the relative error defined above, except that position vectors from the two simulations are compared. Fig. 5 shows the use of a simple predictor of the following form:

$$x_{pred} = x_{old} + \Delta x_{old} / \Delta Base_{old} * \Delta Base, \qquad (6)$$

where x_{pred} is the predicted position for a point that was at position x_{old} the previous time step, Δx_{old} is the amount it had moved the previous time step, $\Delta Base_{old}$ is the amount a corresponding point in the base simulation moved the previous time step, and $\Delta Base$ is the amount a corresponding

[1]We did not perform an actual parallel experiment, but simulated such an experiment.

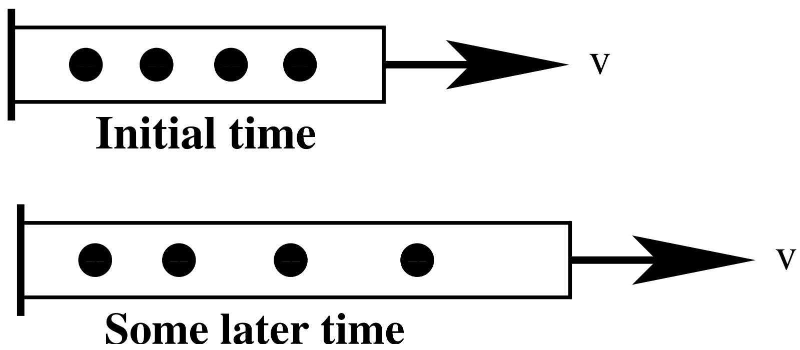

Figure 3. Schematic of the model problem. A material is held fixed at one end, and pulled at the other end. In order to make the prediction more challenging, we used a model were points that were initially equally spaced, later become unequally spaced.

point in the base simulation moved in this time step. This prediction scheme corresponds to predicting the displacement of a point this time step based on the displacement of a similar point in the base, and on the ratio of the displacements of the new and base simulations the previous time step. The ratio $\Delta x_{old}/\Delta Base_{old}$ determines the relationship between the base simulation and the new simulation (with a different value for each point), and is updated every time step, thus leading to a rudimentary form of learning. Clearly, more sophisticated approaches to learning can lead to better predictors. We choose the failure threshold to be a relative error of 10^{-8} (which corresponds to the accuracy possible for single precision arithmetic; the actual computations used double precision arithmetic). We observe that each processor fails during the first round of simulations (the curve at the top), since the predictor does not know any relationship between the two simulations, at that point in time. (Though we know the relationship between the two simulations in terms of the form of the exact solution and the model parameters, the algorithm did not use this knowledge to determine the predictor, and instead automatically learned about the relationship between the simulations.) The computation restarts from the second time step, and does not fail again (lower curve). Note that the errors here are very low. This is, indeed, due to good prediction, and not because the base simulation is very close to the actual simulation. The purpose of Fig. 4 is to demonstrate that the base is quite different from the new simulation.

If communication overhead and the overhead of determining the predictor were not significant, this would lead to an efficiency of around 90%, and this will be even greater in a longer run (simulations indicate 99% efficiency if the simulation proceeded to time = 100). Since the global communication and predictor are computed only every time interval, they will be relatively lower if the time interval is sufficiently large. Of course, a large time interval may make the predictor poor, leading to more failures.

5 A nanomaterials application

5.1 Application

Development of nanomaterials require understanding phenomena that take place at a scale of a few atoms to develop useful products. 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. For example, a simple device with 1 cubic micro meter comprises about 10^{12} atoms. Computations involving such a large number of atoms are all but impossible to handle through MD alone; however there is a significant level of research to solve the space limitation problem using massively parallel systems and multi-scale modeling methods. For example, one potential approach is to determine the behavior at the nanoscale using MD, and then use this with regular finite element methods (FEM). This can yield acceptable time and space scales. However, we will often need to determine the behavior during the course of an FEM the computation, whenever we encounter a situation for which we do not already know the behavior, and this has to be done fast.

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

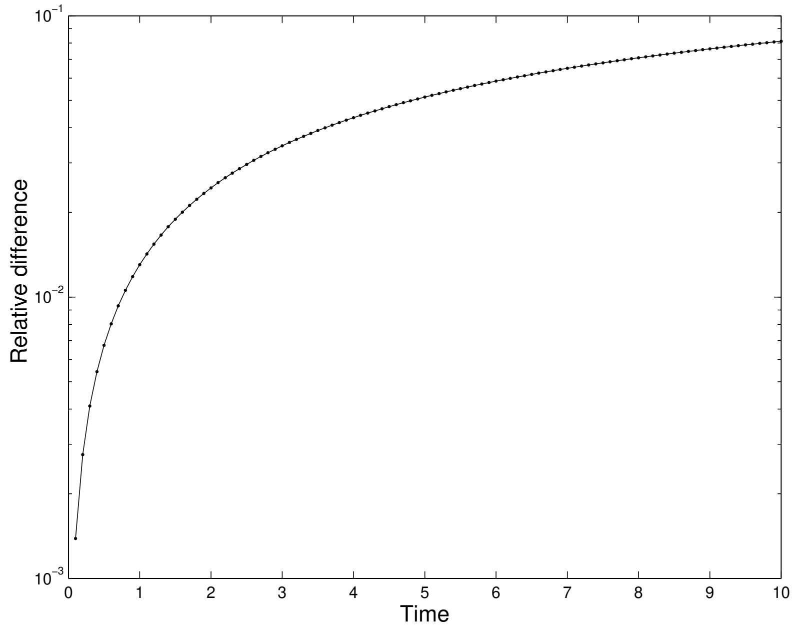

Figure 4. Relative difference between the base simulation and the new simulation. This figure demonstrates that the base simulation and the new simulation are quite different, especially when we contrast with the relative errors in Fig. 5.

of nanotechnology research is to contribute to its progress. Thus we need methods to extend the simulation times.[2]

5.2 Predictors

Determining the behavior at the nanoscale in our application requires simulations of small to moderate sized systems, say with around 3000 atoms, for long time periods. This provides an example of an important class of applications where a small system needs to be simulated to long time scales. We need to determine a suitable predictor for this, which is more difficult than for the model problem presented in § 4. Our approach is based on the observation that such prediction is not new 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.

MD simulations bring an interesting issue that is not present in the the model problem mentioned in § 4 – that of determining the equivalence of two dynamic states. The reason for this is that MD simulations track the vibrational motion of an atom around its mean position. 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 in-

[2]Some time extension schemes, such as hyper MD [6, 8] and Kinetic Monte Carlo [3], 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, 2, 7].

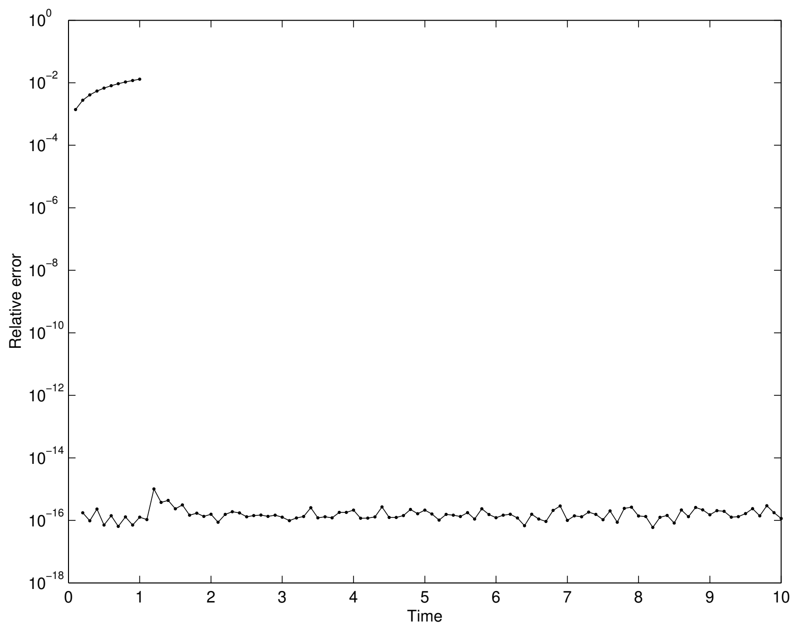

Figure 5. Error in the predictor. The curve at the top shows a large error in the first iteration. Since this is above the error threshold, the computation restarts from the second time step. The lower curve, for subsequent iterations, shows that the error is always smaller than the threshold.

finite number of possible states. We need to verify if the predicted and the accurate states could be samples from the same distribution.

As a simple example, if we consider only the positions of atoms vibrating about their mean positions, and if we know the probability distribution of their position from the mean, then we can determine if the predicted positions differ from the accurate positions in a manner consistent with the probability distribution. (If the probability distributions do not change much, then the base simulations can provide an empirical probability distribution.) Other important factors too need to be account for, such as the minimum information that will yield an accurate predictor. For example, the atom position are not sufficient, since other factors, such as energy too are important (which requires velocities too to be estimated accurately).

6 Experimental results with molecular dynamics simulations

Experimental parameters: Both the base and the actual simulation subject a Carbon nanotube containing 1000 atoms to a pull out test, as in the model problem, except that MD simulations on a real Carbon nanotube are used in these simulations. 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 200 atoms at that end to remain stationary. The tube is pulled at a fixed rate in the z direction by forcing around 200 atoms at the other end to increase their z coordinate value at a rate of s Angstroms per time step, where s is 0.00005 for the base simulation, and 0.0000625 for the simulation being performed. The remaining around 600 atoms are simulated using MD, with the end atoms providing the boundary conditions. Note that with just 600 atoms, we cannot achieve much speedup through spatial parallelization, since the granularity will become very fine.

We first performed an accurate MD simulation for both the base case (since it is needed for the prediction), and also for the new simulation (so that we can compare our predictor against it, and these were also used to provide appropriate start states), and output the results of every 1000 iterations. Simulations were performed for $200,000$ time steps (and we therefore had output the results for 200 time points for each simulation, in addition to the initial state, which was identical for both simulations.)

The predictor: We used only the positions of atoms to define a state. We have mentioned ideas for more sophisticated predictors above, but use here only the same form for the predictor as with the model problem, except that we need to predict for three coordinates. The factor $\Delta x_{old}/\Delta Base_{old}$, which determines the relationship between the two simulations, was expressed in polar coordinates (the use of Cartesian coordinates leads to poor results, as can be expected). We also needed to determine the equivalence of two states. We observed the vibrations of atoms around their mean position in the base simulation (after subtracting out the effect of the deterministic strain that is applied to them), and observed that the range of motion around the mean was about 0.15 Angstroms, and used this as a measure of the range of movement of atoms about their mean position. In order to determine the error, we determined the distance between the positions of atoms in the predicted and accurate computations, and then computed the average for the 600 atoms that actually evolve through MD. We declared a computation to fail if the average was greater than 0.3 Angstroms (since the predicted and accurate positions might be at opposite ends, at a distance of 0.15 Angstroms each from the mean).

Experimental setup: A parallel run with 10 processors was actually simulated in Matlab, rather than in a real parallel machine.

Results: The results of the simulation are presented in Fig. 6. All the processors fails during the first iteration, but do not fail subsequently. Thus the speedup is about 9.5, if we ignore communication costs and the cost of the predictor, and the efficiency is about 95%. We next explain why ignoring the communication cost and the cost of the predictor can be justified. 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 (for example, $\approx 15s$ on an IBM SP3). In contrast, a reduction on an IBM SP3, even with 100 processors, takes just around 0.0005 seconds. 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^{-2} seconds, and so is a few orders of magnitude smaller than the verification time. Thus we are justified in ignoring these costs. We also note that while we simulated a situation with around 10 processors, the fact that the results remain good after the first iteration suggests that we even increase the number of processors and the time scale being simulated, and get good speedup. (In fact, a simulation with 40 processors indicates a speedup of 28, at around 70% efficiency. The loss in efficiency is primarily due to the time taken for "learning" the relationship initially, and so a long simulation can be expected to yield close to perfect speedup.)

Limitations of the experiments: One of the limitations of the experiment is that it is a simulation of a parallel run, rather than a real parallel run. However, the computation and communication costs mentioned above clearly indicate that the speedup and efficiency would be maintained in a longer simulation. The important limitation of the experiments is that choosing the positions of the atoms to define the state of the physical system is not sufficient, and higher derivatives of the position, and also the energy, would need to be considered. Another consequence of our definition of the state not being sufficient was that we had to use the computed results of the new simulation to provide start states. Furthermore, we would need a better definition of the equivalence of two dynamical states than the simple one considered here. Yet another limitation is that have considered a only a pull-out test here. Other types of experiments, such as use of a higher temperature to predict long term behavior at a lower temperature, would be very useful.

7 Conclusions and future work

We have proposed a strategy which can enable high latency environments, such as the Grid, to be effective for problems where a typical spatial decomposition will lead to too fine a granularity. We have demonstrated its effectiveness on a model problem without randomness, and also on a simple version of an important nanotechnology application, where random motion complicates matters. We have also outlined ideas for making it effective in more realistic situations. Though the experiments performed here were simple, the results indicate much promise for this technique. Further work needs to be done to bring to fruition the promise of this approach, as explained below.

We are working on a better definition of the equivalence of two dynamical states. For example, a state can be considered a sample from an ensemble, and so we need to only determine if the predicted and accurate states could be from the same ensemble; a point by point comparison of the coordinates is not necessary. On the other hand, we need to include additional factors in the definition of the state, such as energy. We also need to address the fact that in many problems, only certain global properties of the state, such as the energy, are of interest, rather than the exact state of each atom. A suitable selection of such global quantities will also permit us to use a larger class of simulations as potential base simulations. For example, it should be feasible to use a simulation with 990 atoms as a base simulation for another simulation with 1000 atoms, which cannot be done in the point-wise predictor we have used here. Additional computational issues to be addressed include organization of a suitable database of simulations. It will be useful to determine schemes to automatically and efficiently choose appropriate base simulations by observing the trend of the simulation being performed. The simulation used as the base may perhaps need to be changed during the course of the simulation, if the physical system experiences some physical phenomenon that causes it to behave in a manner different from the current base. We are also working on better prediction and learning techniques, which will enable this approach to be much more effective in realistic situations.

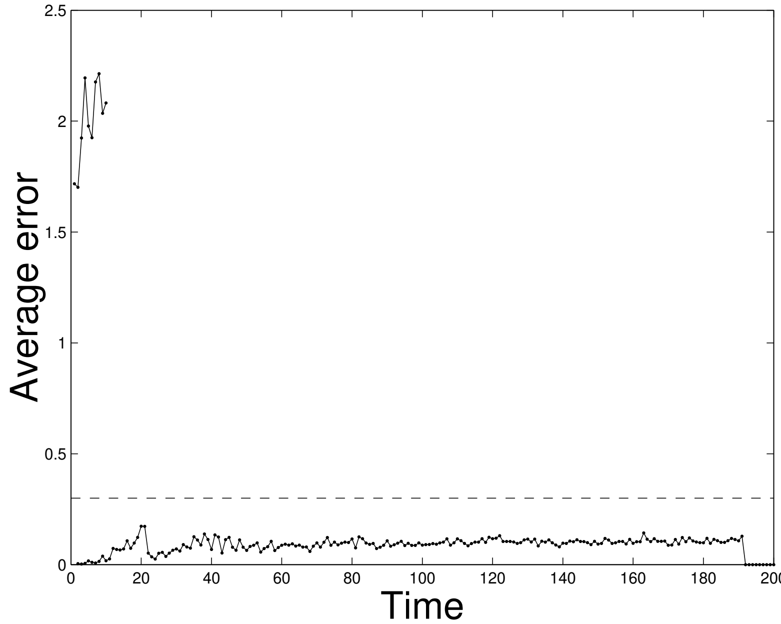

Figure 6. Error comparing the predictor with the exact MD simulations. The curve at the top shows a large error in the first iteration. Since this is above the error threshold (shown by the dashed line), the computation restarts from the second time step. The lower curve, for subsequent iterations, shows that the error is always smaller than the threshold. The error units are in Angstroms.

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

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] 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.

[3] 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.

[4] 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.

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

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

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

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

8 Biographies

Ashok Srinivasan is an assistant professor in the Computer Science department at Florida State University. After obtained his Ph.D. from the University of California at Santa Barbara, he worked at the National Center for Supercomputing Applications (NCSA), the Indian Institute of Technology – Bombay, and UCSB, before joining FSU. His research interests are in parallel algorithms, high performance and scientific computing, applications, and mathematical software. In particular, he is currently working on scalable algorithms for emerging architectures, which address latency tolerance, fault tolerance at an algorithmic level, and low over-head load balancing. In particular, he is working on applications in computational nanotechnology and applications of Monte Carlo linear algebra. He has previously worked on computational fluid dynamics, computational finance, image compression, and also developed the SPRNG parallel random number software which is widely used by the high performance computing community. Further details are available at: http://www.cs.fsu.edu/ãsriniva.

Namas Chandra is the Karamcheti Professor of Engineering at Florida State University. He received his B.S., M.S. and Ph.D. (1986, Texas A&M) in Mechanical Engineering and also AMIE (equivalent to BS) in metallurgy. He joined the Florida A&M-Florida State Universities College of Engineering in 1986, where he is a (named) full professor in the Department of Mechanical Engineering. During this period, he has published over 137 publications including 63 in archival journals primarily in the area of computational mechanics, superplasticity and composites. He has also edited four books. He has been continuously funded approximately $3 Million from federal, state and private agencies. Seven Ph.D.s (including one African-American) and twelve M.S. (including three African-American) have graduated under his supervision. Professor Chandra has offered several workshops on superplasticity and superplastic forming in U.S. and abroad (Taiwan, China, Japan and India) to scientists and practicing engineers. He was the first engineering faculty to receive the Developing Scholar Award in 1993, and the only engineering faculty to receive the Outstanding Service to the Profession Award of the Florida Engineering Society in 1990. He was elected as a Fellow of ASME in 1997. He is the associate editor of the premier ASME journal of Materials, ASM journal of processing, and Mechanics of Advanced Materials and Structures.