Application of Reduce Order Modeling to Time Parallelization

Ashok Srinivasan[1], Yanan Yu[2] and Namas Chandra[3]

[1] Computer Science, Florida State University,
Tallahassee FL 32306, USA
asriniva@cs.fsu.edu
[2] Computer Science, Florida State University,
Tallahassee FL 32306, USA
yu@cs.fsu.edu
[3] Mechanical Engineering, Florida State University,
Tallahassee FL 32310, USA
chandra@eng.fsu.edu

Abstract. We recently proposed a new approach to parallelization, by decomposing the time domain, instead of the conventional space domain. This improves latency tolerance, and we demonstrated its effectiveness in a practical application, where it scaled to much larger numbers of processors than conventional parallelization. This approach is fundamentally based on dynamically predicting the state of a system from data of related simulations. In earlier work, we used knowledge of the science of the problem to perform the prediction. In complicated simulations, this is not feasible. In this work, we show how reduced order modeling can be used for prediction, without requiring much knowledge of the science. We demonstrate its effectiveness in an important nano-materials application. The significance of this work lies in proposing a novel approach, based on established mathematical theory, that permits effective parallelization of time. This has important applications in multi-scale simulations, especially in dealing with long time-scales.

1 Introduction

Many problems in science are formulated as initial value problems. The initial state of a physical system at some time is given, along with, possibly, some boundary conditions. A differential equation describes how the state changes with time, and possibly space. The problem is solved by iteratively computing the states at successive points in time, using a differential equation solver. We shall refer to each iteration as a *time step*. A large computational effort can be involved when the state of the system is large, or when the number of time steps is large. In order to reduce the computation time, parallelization is often used, especially with large physical systems.

Even when the state space is not large, the computational effort can be large if we need to compute for a large number of time steps. This has been identified

as one of the important challenges in nanoscale simulations and computational materials science [3]. Conventional parallelization is not effective in such problems, because the granularity becomes fine, limiting scalability.

We recently proposed [5, 6] a time parallelization approach, to improve scalability. Here, results from related simulations are used to parallelize along the time domain. The basic idea is to have each processor simulate a different interval of time. The difficulty here is that each processor needs the state of the system at the beginning of the time interval it simulates, since it solves an initial value problem. We use the fact that typically the results of prior, related, simulations are available, to predict, in parallel, the states of the current simulation at desired points in time. The prediction mechanism also 'learns", thereby attempting to predict better as the simulation proceeds. The predicted states are verified in parallel to ensure accuracy of the results.

An important limitation of the earlier approach was that the prediction mechanism required some detailed knowledge of the the science of the problem. This is not easy to obtain in complicated simulation conditions. The main focus of this work is to show how reduced order modeling can be used to perform predictions for time parallelization.

The significance of this work lies in presenting an approach to time parallelization that is based on stronger mathematical foundations. This makes time parallelization of more complex problems too feasible. Time parallelization, especially for Molecular Dynamics (MD) simulations, has important applications to long time simulation simulations in computational Chemistry, Physics, Biology, Materials, and Engineering, making this work important.

The outline of the rest of the paper is as follows. In § 2, we describe a nanomaterials application that will be used to demonstrate the effectiveness of our technique. We then summarize the time-parallelization approach in § 3. In § 4, we describe prior and related work. We outline a particular reduced order modeling method, called *Proper Orthogonal Decomposition* (POD), and describe its use in time parallelization, in § 5. In § 6, we show the details of the steps involved in using POD to time-parallelize our application. We present conclusions and future work in § 7.

2 Carbon Nanotube Application

Tensile Test on a Carbon Nanotube. The physical system we consider is a Carbon Nanotube (CNT) [4]. One important application of CNTs is in nanocomposites, where CNTs are embedded in a polymer matrix. In such applications, it is important to determine the mechanical properties of the CNT. Some important data for this is obtained from the "tensile test", in which one end of the CNT is held fixed, while the other end is pulled at a constant velocity. The response of the material is characterized by the *stress* (force required to pull the tube, divided by it cross-sectional area) experienced at a given *strain* (the elongation of the nanotube, relative to its original length). A stress-strain curve, as shown in Fig. 3 later, describes the response of the material when it is

pulled at the specified velocity (more formally, strain-rate). Another important property is the strain at which the CNT starts to break.

Molecular Dynamics Simulation of a CNT. The state S_t of the CNT at any time t is defined by the position and velocity, at time t, of the atoms in the CNT. For a CNT with N atoms, there are $6N$ quantities (three position coordinates and the three velocity coordinates per atom) that define the state. The mechanical properties of the CNT at time t can be determined from S_t. Given S_t, we compute the state $S_{t+\Delta t}$ at the next time step $t + \Delta t$, by first computing forces on each atom due to other atoms [4], and then using Newton's laws of motion. A numerical time integration scheme (fourth-order Nordsiek in our implementation) is used in the latter. In MD computations, the time step size Δt is typically required to be less than a femto second (10^{-15}s), to ensure stability and accuracy.

This small Δt is an impediment to effective MD computation, because it makes a large number of time steps necessary. Furthermore, this computation will not parallelize efficiently using conventional parallelization, for physical systems of realistic sizes. As an alternative, researchers simulate by pulling the CNT at a faster rate than is realistic. The faster-rate simulation requires less time than that for a realistic rate, because the same strain is reached in less time with the former. It is assumed that the stress-strain relationship determined at the higher strain rate is the same as that at a lower strain rate. However, it is known that such an assumption is not accurate when the strain rates vary by several orders of magnitude [7]. On the other hand, if we could parallelize the computation efficiently on a large number of processors, then we could reach the desired time scale with more realistic strain-rates too.

3 Time Parallelization through Guided Simulations

We recently [5, 6] introduced the idea of guided simulations to parallelize along the time domain. We outline the approach below in Algorithm 1. Some details are deferred to § 6.

Let us call a few, say 1000, time steps as a *time interval*. Divide the total number of time steps needed into a number of time intervals. Ideally, the number of intervals should be much greater than the number of processors. Let t_i denote the beginning of the i th interval. Each processor $i \in \{1..P\}$, somehow predicts the states at times t_{i-1} and t_i, with the state at time t_0 being a known initial state S_0. Then it performs accurate computations (MD in our application), starting from the predicted state at time t_{i-1} up to time t_i. It then *verifies* if the prediction for t_i is close to the computed result. Both prediction and verification are done in parallel. If the predicted result is close to the computed one, then the initial state for processor $i+1$ was accurate, and so the computed result for processor $i+1$ too is accurate, provided the predicted state for time t_{i-1} was accurate. If the results differ significantly, then the predicted state for t_i was inaccurate, and we say that processor i *erred*. Computations for subsequent points

TimeParallelize(Initial State S_0, Number of processors P, Number of time intervals m)

- i=0; $\hat{S}_0$=S_0
- WHILE $i < m$
  - FOR each processor $j \in \{1..\min(P, m-i-1)\}$
    - T_{i+j-1} = PredictStateAt (time = i+j-1)
    - T_{i+j} = PredictStateAt (time = i+j)
    - $\hat{S}_{i+j}$ = AccurateComputation(StartState=T_{i+j-1}, StartTime=i+j-1, EndTime=i+j)
    - UpdatePredictionParameters(CurrentParameters, $\hat{S}_{i+j}$, T_{i+j})
    - IF IsDifferenceTooLarge($\hat{S}_{i+j}$, T_{i+j})
      - THEN $Next_j = j$
      - ELSE $Next_j$ = P
  - k = AllReduce(Next, min)
  - IF j=k
    - THEN Broadcast(Prediction Parameters)
  - SendReceive($\hat{S}_{i+k}$, From Processor k,To Processor 0)
  - FOR each processor $j \in \{1..P\}$
    - $i = i + k$

Fig. 1. The time parallelization algorithm.

in time too have to be discarded, since they might have started from an incorrect start state. The next phase starts from computed state for the latest time that is known to be accurate. The errors observed in the previous verification step are used to improve the predictor by better determining the relationship between the current simulation and old ones.

Note the following: (i) Processor 0 always starts from a state known to be accurate. (ii) The algorithm *always progresses* at least one time interval, since the accurate computations on processor 0 lead to accurate results on that processor. (iii) The prediction mechanism has two components – one that uses only prior simulation data, and another that "learns" the relationship between prior data and the current simulation based on the difference between the predicted and the computed states. The learning is represented using some prediction parameters. These parameters need to be identical on all processors. Otherwise, verification of prediction at time t_i at processor i does not imply that the prediction for initial state at time t_i on processor $i+1$ was correct. So these prediction parameters are broadcast in Algorithm 1. (iv) The answers are *always accurate*, if we correctly define "sufficient closeness" of the predicted and computed states; a good predictor enables greater speedup, while a poor one leads to it becoming a sequential computation. (v) If the time interval consists of a large number of time steps, then the communication cost can be made negligible, leading to a very *latency tolerant* algorithm.

4 Related Work

Prior Work. In Algorithm 1, we need implementations of the following functions: (i) PredictStateAt, (ii) UpdatePredictionParameters, and (iii) IsDifferenceTooLarge. We used knowledge of the science to implement those functions in [6]. A large amount of data, totaling around 500 MBytes, were required for accurate prediction. Determining the permissible difference between predicted and computed states is application dependent. We defined two states to be equivalent, in our application, if the differences in positions, potential energy (also a function of positions), and kinetic energy (a function of velocities) were less than inherent fluctuations, as described in [5, 6]. Using data from several time points of a single simulation that pulled a 1000-atom CNT at 10m/s, we predicted the behavior of a CNT pulled at 1m/s. The resulting time parallel code ran on 990 processors of the Xeon cluster at NCSA with efficiency greater than 97% [6].

Other Approaches. Due to its importance, there have been several works on the spatial parallelization of MD, including CNT computations specifically [4]. Time parallelization using the Parareal approach [1] is another promising alternative to conventional parallelization. We described its limitations in detail in [5]. The speedup and efficiency obtained have been limited. Speedups on simulated experiments (ignoring communication costs – the experiments were not on actual parallel machines) ranged between 8 to 130, with efficiency between 25% and 1.3% respectively on some model problems.

5 Application of Reduced Order Modeling to Time Parallelization

The basic idea behind reduced order modeling is that, while the state space involved in a simulation might be high dimensional, the states lie close to a lower dimensional subspace of the larger space. For example, the state of the CNT with N atoms is defined by $6N$ quantities, which can be represented by a vector $x \in \Re^{6N}$. If the CNT is pulled in the z direction, then most of the interesting changes take place in the z coordinates of the atoms. So we expect to find a smaller dimensional subspace of $\Re^{6N}$, close to which the states lie.

We use Proper Orthogonal Decomposition (POD) for reduced order modeling. The same idea is known by other names, such as Principal Component Analysis or Karhunen-Loève analysis. We next outline this method, and the intuition behind it. Further details can be found in [2].

Let $\hat{x}(t; v) \in \Re^m$ denote the state of a system at time t, simulated with parameters v. For example, in our application, v is a single parameter – the velocity at which the CNT is pulled. We assume that the states of the simulations lie close to a smaller dimensional affine subspace (a shifted linear subspace) of $\Re^m$. The shift is given by some vector μ, and the linear subspace S is given by the span of some vectors, which we represent as the columns of a matrix U. POD attempts to find a μ and a U that define this affine subspace $\mu + \text{span}\{U\}$.

We assume that a database of simulation results exists, with the states of simulations conducted under different parameters stored, for various values of time t. We choose n of these states to find a suitable basis U. Proper choice of these states is an important issue, which we shall not discuss here. Let us call the chosen states $\hat{x}_i,\ 1 \leq i \leq n$. We consider the case where $n < m$. We define $\mu = 1/n \sum_{i=1}^{n} \hat{x}_i$. Let $x_i = \hat{x}_i - \mu$. We then construct the *snapshot matrix* $A = [x_1 x_2 \cdots x_n] \in \Re^{m \times n}$.

Let $A = \tilde{U}\tilde{\Sigma}\tilde{V}^T$ be the singular value decomposition of A. Here, $\tilde{U} = [u_1 \cdots u_m] \in \Re^{m \times m}$ and $\tilde{V} \in \Re^{n \times n}$ are orthogonal matrices, and the rectangular diagonal matrix $\tilde{\Sigma} \in \Re^{m \times n}$ contains the singular values of A (which are non-negative) in descending order. The columns of $\tilde{U}$ form a basis for $\Re^m$. Column i of $\tilde{\Sigma}\tilde{V}^T$ gives the coefficients of x_i in the basis consisting of columns of $\tilde{U}$. The coefficients of columns $u_i,\ i \geq n$, are zero. If the x_i's lie close to a d dimensional linear subspace of $\Re^m$, then only the first d singular values are large. So we define $U \in \Re^{m \times d}$ to consist of the first d columns of $\tilde{U}$, $\Sigma \in \Re^{d \times d}$ the corresponding submatrix of $\tilde{\Sigma}$, and $V \in \Re^{n \times d}$ the first d columns of $\tilde{V}$. Then $A \approx U\Sigma V^T$, and $\sigma_i v_{ji}$ is the component of x_j along the direction of u_i. That is, $x_j \approx \Sigma_{i=1}^{d} c_{ji} u_i$, where $c_{ji} = \sigma_i v_{ji}$.

The linear subspace S spanned by the columns of U is optimal in the sense that among all linear subspaces S' of dimension d, it has a minimum value of $\Sigma_{i=1}^{n} D(S', x_i)$, where $D(S', x_i)$ is the square of the distance between x_i and S'. Note that if the x_i's lie close to S, then the original data $\hat{x_i}$'s lie close to the affine subspace $S + \mu$.

If we just wished to represent the database with less storage, then we can store μ, U, and the coefficients of U for each state in the database. Since the columns of U are orthogonal, the coefficients for any vector $\hat{x}$ are easily obtained as $U^T(\hat{x} - \mu)$, involving just a vector subtraction and d dot products. If there are M states in the database, then the total storage is $Md + (d+1)m$, with $d \ll M, m$, in contrast to Mm storage required for the original data. In simulating dynamical systems, the state is expressed as $\hat{x} \approx \mu + \Sigma_{i=1}^{d} c_i u_i$, and this is substituted in the equations defining the evolution of the state, to come up with equations that define the evolution of c_i's.

6 Experimental results

We first show various aspects of the application of POD to prediction, and then present speedup results. The physical system considered is a 1000-atom CNT at $300K$. The database consisted of tensile test simulation results at velocities given below, from zero strain to until the CNT starts to break. The time parallel simulations were performed for velocities of 2m/s and 0.1m/s.

Basis vectors for the CNT. The states at the following points in time were used to construct the snapshot matrix A: (i) Velocity = 10m/s: after time steps – 50,000, 100,000, 200,000, 300,000, and 350,000. (ii) Velocity = 5m/s: after time steps – 100,000, 200,000, 400,000, 600,000, and 700,000. (iii) Velocity = 1m/s:

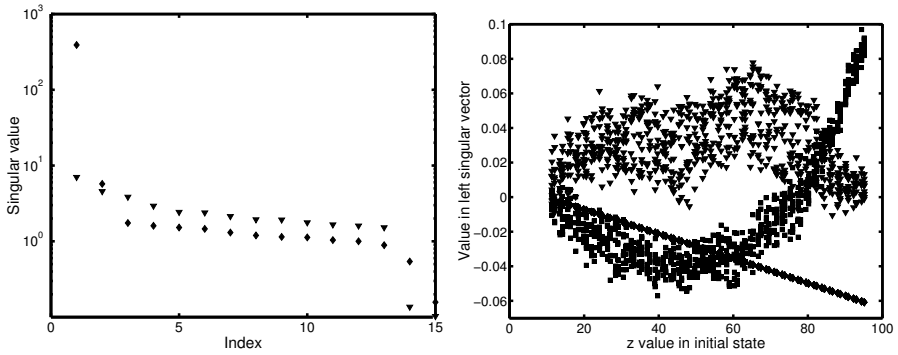

Fig. 2. Left: Plot of singular values against its index on a semi-log scale. The diamonds show the values for z and the triangles for y. Right: Plot of basis vectors against the z coordinate value of the corresponding atom in the initial state. The diamonds show the values for z's u_1, the squares for z's u_2, and the triangles for x's u_1.

after time steps – 500,000, 1,000,000, 2,000,000, 3,000,000, and 3,500,000. (iv) The initial state, which is identical for all velocities. The above times indicate that we chose a set of five different strain values, and noted the state at these strains for each of the three parameters. We generate separate bases for the x, y, and z coordinates, and so we created three different snapshot matrices, one for each set of coordinates. Each snapshot matrix is of dimension 800×16, with each row corresponding to the position coordinate of an atom moved using MD (the temperature is held constant, and so we did not include the velocities in the state).

We then determined μ, $\tilde{U}$, $\tilde{\Sigma}$, and $\tilde{V}$ as described in § 5. For each coordinate, *columns of $\hat{U}$ that corresponded to large singular values were usually placed in U.* Apart from the singular value, we also checked to see if the vector represented a pattern, or just corresponded to random "noise". Fig. 2 shows that the z coordinate has one very high singular value, followed by a moderately large one, followed by several smaller ones. We look further into each u_i corresponding to the larger singular values. Fig. 2 shows the values of the components of u_1 and u_2 for z are non-random. On the other hand, u_1 of x appears random.

Based on many such observations of the singular values and randomness, we used u_1 and u_2 as basis for z's lower dimensional subspace, and none for x and y. Consequently, the predicted values for the x and y coordinates are always μ_x and μ_y respectively, while for the z coordinate, we use a two dimensional subspace $\mu_z + \text{span}\{u_1, u_2\}$.

Relationship between velocity and time. The relationship between velocity and time is important for the following reason. We expect time parallelization to be effective if we can predict the behavior of a simulation with parameter v, from those of prior simulations performed under different parameters. We are, in effect, assuming similarity of behavior under different parameters. However, the

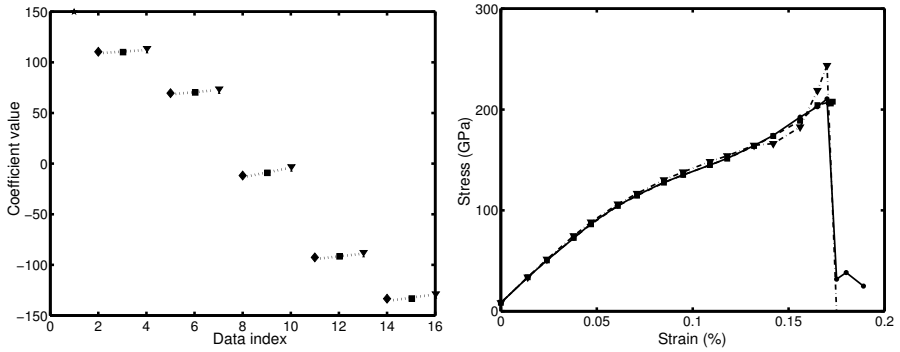

Fig. 3. Left: Plot of z's u_1 coefficient against the the index of the data point (that is, its column number in the A snapshot matrix). The diamonds show the values for velocity 1m/s, the squares for 5m/s, and the triangles for 10m/s. The dotted lines connect points at the same values of strain. Right: Plot of stress versus strain at 0.1m/s. The solid line represents the exact sequential MD results. The squares represents the time parallel code on 400 processors. The dash-dotted line with triangles represents direct prediction.

behavior at time t with parameter v may be similar to the behavior at a different time t_1 when simulated with parameter v_1. For example, in our prior work, we expected long time behavior at a low strain rate to be related to short time behavior at a large strain rate. This was based on knowledge of the science of the problem, and is not often easy to predict. Instead, *we identify similar behavior as having similar values of coefficients for basis vectors with large singular values.* In our application, only z's u_1 has a very large singular value. Fig. 3 shows the coefficients for z's u_1 are similar for similar strains. This provides a more formal justification for the assumption used in the previous work, which we use in this paper too.

Direct Prediction using Interpolation. We can predict the state $\hat{x}(t; v)$ for a time-parameter combination not in the database by first predicting the vector of coefficients, c. This is done by *interpolating or extrapolating coefficients of similar states from the database*. The parameter-time relation obtained from the previous step tells us which states can be expected to be similar. Then the actual state is easily obtained as $\hat{x}(t; v) = Uc + \mu$. We call this prediction as a *direct prediction*. We later explain how the learning mechanism modifies this prediction.

We now give some details of the implementation. We precomputed coefficients for 40 states at different times (or, equivalently, strains) for each of the velocities: 10m/s, 5m/s, and 1m/s. We also computed coefficients for the initial state, which was identical for all velocities. These coefficients are easily obtained as $U^T(\hat{x}-\mu)$ for a known state $\hat{x}$. The amount of data is small, and is stored in an array in our

code. Note that even the number of prior results required (121) is fairly small, requiring less than 10 MBytes of disk space.

We consider the coefficients to be functions of v and strain ϵ, in view of the relationship obtained earlier. When we need the state for velocity v and time t, we determine $\epsilon = vt/L_0$, where L_0 is the original length of the CNT. Then we determine two velocities v_1 and v_2 that are closest to v (preferably with $v_1 \leq v \leq v_2$) in our database. We identify strains closest to ϵ in the database, and then fit a linear surface to these known points and interpolate or extrapolate.

Updating the predictor. The time parallel code used in this paper performs a simple form of learning. The learning can be of two types. In the first, the actual state may still be in the subspace spanned by U, but the predicted coefficients may be systematically incorrect. We will refer to terms that correct for this difference as *corrector coefficients*. In the second case, we may need a new basis vector, which corresponds to a new physical process. Singular value decomposition of the residuals in the verification step, orthogonal to the subspace spanned by U, can yield such a basis. We have incorporated the former type of learning in our implementation.

The corrector coefficients are initially 0. Let processor i start from a predicted state at time t_i and perform accurate MD computations up to time t_{i+1}. Let the state reached be $\hat{S}$ and the predicted state at time t_{i+1} be T. Processor i then computes coefficients $c = U^T(\hat{S} - T)$ and adds it to the current corrector coefficients. The lowest indexed processor that erred, or processor P if none erred, broadcasts its corrector coefficients to all processors. In the next phase of computations, let T be the state predicted from the interpolation. Then the actual prediction for that time is taken to be $T + Uc$. If the systematic error in coefficients varies slowly with time, then this correction accounts for it. It does not account for errors that arise from the current simulation leaving the low dimensional subspace.

Direct prediction can be performed for any point in time. Time parallelization can be accurate even when direct parallelization is not, for the following reasons. (i) The corrector coefficients perform a simple form of learning, and make predictions better. (ii) The verification step can detect errors, and so the results are accurate, even though the computation slows down. (iii) The computed state can depart from the low-dimensional subspace, and come closer to the correct state. The computation may slow down in this case too, since differences between predicted and computed states are treated as errors in the verification step.

Validation. Fig. 3 shows the stress-strain relationship from the exact sequential simulation, direct prediction, and the time parallel code. This relationship is the primary material property of interest. Away from the point when the CNT starts to break (stress reduces with increase in strain then), direct prediction is quite accurate. However, close to the point of breakage, it errs. This error can be traced to its higher errors in potential energy. However, it predicts the time of start of breakage correctly as at around 17% strain. Note that the stress-strain

relationship obtained from the time parallel code is accurate until the point of breakage. From a practical point of view, the behavior of the CNT after its starts breaking is not relevant. We performed similar validation against the exact results for 2m/s simulations, and with different numbers of processors. We also compared other quantities, such as positions and potential energy.

The above observations suggest that direct prediction may be acceptable when interpolating in a parameter-time range between existing data. However, extrapolation can lead to errors. For example, the points close to the CNT breaking involve extrapolation in strain and in velocity. The time parallel code performs accurately until the point of breakage. After this, since the code does not correct for the new phenomenon of breaking, the predictions fail. Of course, this is detected during verification, and so the computation progresses slowly.

When extrapolating data over a wide range, such as performing a very low strain rate calculation, where new phenomena are likely to occur, direct prediction may not be accurate. Time parallelization, on the other hand, can be effective there, since it does not give erroneous results. The savings in time using direct prediction are enormous, when it can be applied. Determining the stress-strain relationship at a velocity of 0.1m/s, for example, requires about a week of sequential computing time when we simulate until the CNT starts to break. This can be done in 10^{-5}s per time point using direct prediction. Time parallelization yields accurate results up to the point of breakage, and we have simulated it on hundreds of processors with nearly ideal speedup. So the above simulation can be completed in less than an hour, accurately, in parallel.

Speedup Results. Speedup results are reported on the *Tungsten* Xeon cluster at NCSA. This cluster consists of Dell PowerEdge 1750 servers, with each node containing two Intel Xeon 3.2 GHz processors, 3 GB ECC DDR SDRAM memory, 512 KB L2 cache, 1 MB L3 cache, running Red Hat Linux. Myrinet 2000 and Gigabit Ethernet interconnects are available. We used the Myrinet interconnect. The ChaMPIon/Pro MPI implementation was used with gcc/g77 compilers for our mixed C/Fortran code, compiled with '-O3' optimization flags set. The MPI calls were purely in the C code. The timing results are based on wall clock time when run in non-dedicated mode.

Fig. 4 shows the speedup results, for a simulation at 0.1m/s. Similar results were obtained with 2m/s simulations, which was run on up to 250 processors. We can see that speedup is good on up to 400 processors on the Xeon cluster. The efficiency is greater than 95% for all cases except for 400 processors, where the efficiency is around 90%. The processors never err in the course of the simulation (up to the point where the CNT starts to break), and so loss in speed is only due to the overheads of prediction and communication. These overheads are small, compared with the computation time. For example, the prediction related computations take less than 10^{-4}s, the AllReduce $\approx 10^{-4} - 10^{-3}$s for 50-1000 processors, Broadcast $\approx 10^{-4}$s for 50-1000 processors, and the Send/Recv about 10^{-4}s. Load imbalance is not an issue, since each processor performs, essentially, the same amount of computation. All the overheads are insignificant, relative to

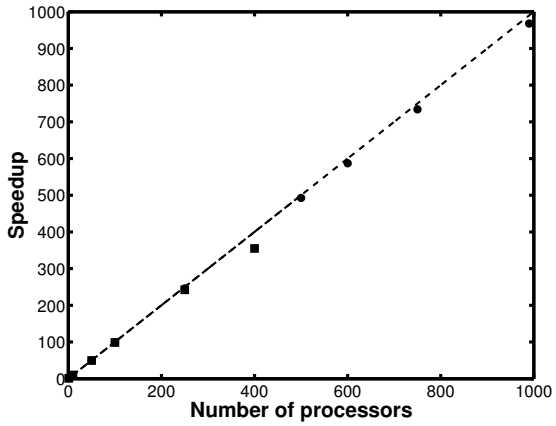

Fig. 4. Speedup curve. The dashed line shows the ideal speedup, and squares show the speedup on the NCSA Xeon cluster for a velocity of 0.1m/s, run in *non-dedicated* mode, with the new prediction scheme. The circles show speedup on the same machine with the earlier predictor, for a velocity of 1m/s, run in *dedicated* mode.

the computation time ($\approx$ 13s) for simulating a single time interval. We expect the loss in efficiency, especially for the 400 processor run, to be due to running in a non-dedicated mode. For example, we got over 97% efficiency on up to 1000 processors with our previous predictor, when run in dedicated mode. However, the actual prediction and communication overheads of the previous predictor are slightly larger than for the new one. In particular, the earlier prediction related computations took $\approx 10^{-3}$s, and Broadcast $\approx 10^{-1} - 10^{-2}$s for 50-1000 processors. In any case, the efficiency is very good with both predictors and they scale to much larger numbers of processors than conventional parallelization.

7 Conclusions

We have shown that reduced order modeling can provide a systematic procedure for choosing a basis for modeling the data, without much apriori information on the physical processes the system is undergoing. Such modeling enables us to predict the states, for different time and parameter values. Furthermore, the amount of prior data needed is less. Parallelization of time, using our approach, scales to at least two orders of magnitude larger numbers of processors than conventional parallelization. Our results are, therefore, of much significance, since they suggest a general technique for more complicated problems, were less knowledge of the physical processes is available.

Future work will consist of simulating multiple parameter systems, such as strain rate, temperature, and CNT diameter. We will also include, in the implementation, the ability to learn about new phenomena the CNT undergoes, orthogonal to the selected subspace. We plan to use other reduced order modeling techniques too, such as Centroidal Voronoi Tesselations [2]. We will also

perform time parallel simulations under more experimental conditions, apart than tensile tests, and at lower strain rates, and include the material responses in multi-scale FEM simulations.

Acknowledgments

This work was funded by NSF grant # CMS-0403746. We thank ORNL (CNMS /NTI grant #CNMS2004-028) and NCSA (proposal #ASC050004) for providing computer time. We also thank Xin Yuan at Florida State University for permitting use of his Linux cluster, where our codes are first tested, and Max Gunzburger, for drawing our attention to reduced order modeling literature. Most of all, A.S. thanks Sri S.S. Baba, whose inspiration and help were crucial to the success of this work.

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. J. Burkardt, Q. Du, M. Gunzburger, and H. Lee. Reduced order modeling of complex systems. In D. F. Griffiths and G. A. Watson, editors, *Proceedings of the 20 th biennial conference on Numerical Analysis*, pages 29–38, Dundee, Scotland, U.K., 2003. University of Dundee.
3. 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.
4. J. Kolhe, U. Chandra, S. Namilae, A. Srinivasan, and N. Chandra. Parallel simulation of Carbon nanotube based composites. In L. Bougé and V. K. Prasanna, editors, *Proceedings of the 11 th International Conference on High Performance Computing (HiPC), Lecture Notes in Computer Science – 3296*, pages 211–221. Springer-Verlag, 2004.
5. A. Srinivasan and N. Chandra. Latency tolerance through parallelization of time in scientific applications. *Parallel Computing*, 31:777–796, 2005.
6. A. Srinivasan, Y. Yu, and N. Chandra. Scalable parallelization of molecular dynamics simulations in nano mechanics, through time parallelization. Technical Report TR-050426, Department of Computer Science, Florida State University, 2005.
7. B. I. Yakobson, M. P. Campbell MP, and C. J. Brabec. High strain rate fracture and C-chain unraveling in Carbon nanotubes. *Computational Materials Science*, 8:341–348, 1997.