Scalable Parallel Execution of an Event-based Radio Signal Propagation Model for Cluttered 3D Terrains

Sudip K. Seal and Kalyan S. Perumalla
Oak Ridge National Laboratory
Oak Ridge, TN 37831, U.S.A.
sealsk, perumallaks@ornl.gov

***Abstract*—Estimation of radio signal strength is essential in many applications, including the design of military radio communications and industrial wireless installations. While classical approaches such as finite difference methods are well-known, new event-based models of radio signal propagation have been recently shown to deliver such estimates faster (via serial execution) when compared to other methods. For scenarios with large or richly-featured geographical volumes, however, parallel processing is required to meet the memory and computation time demands. Here, we present a scalable and efficient parallel execution of a recently-developed event-based radio signal propagation model. We demonstrate its scalability to thousands of processors, with parallel speedups over 1000$\times$. The speed and scale achieved by our parallel execution allow for larger scenarios and faster execution than has ever been reported before.**

***Keywords*-parallel discrete event simulation; radio signal propagation**

I. Introduction

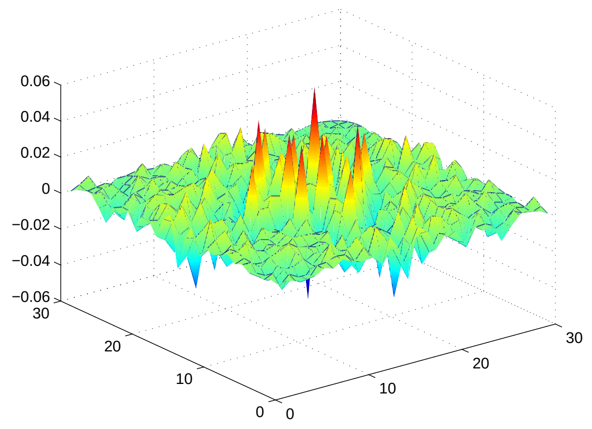
A novel event-based model was recently proposed in [8], [9] for the prediction of signal strength in radio wave propagation. The proposed technique has been shown to yield accurate enough predictions without the high computational overhead of more traditional techniques such as finite difference time domain (FDTD) or ray-tracing methods [2], [12]. Validation studies [9] have shown that the new model is more runtime efficient, albeit in the context of serial execution, compared to FDTD or ray-tracing methods. Parallelization of the above technique becomes necessary due to very large memory and computational time demands associated with simulations of radio signal propagation over large or richly-featured geographical areas, particularly when they need to be carried out in real-time.

A. Motivation

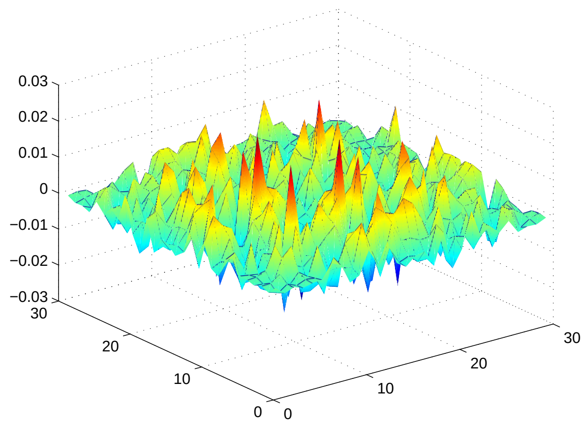
Knowledge of radio signal path loss is of significant interest in the design and deployment of wireless communication networks [5]. For example, in military scenarios, typical geographical terrains of interest are very large and often include physical features that range from buildings and mountains to natural reliefs and foliage. FDTD or ray-tracing models of radio wave propagation in such terrains is computationally very intensive, more so as the number of transmitters and receivers are increased. For scenarios with even a single source and a few receivers, traditional techniques exhibit large runtimes [9]. In particular, faster turnaround times are needed for simulated mobile units, for example, when the transmitters and/or receivers are in moving vehicles. Efficient real time estimation of radio signal strength for such scenarios remains an area of on-going research [4], [7]. The reader is referred to references such as [8], [9] for additional details behind the motivation.

B. Related Work

In FDTD methods, Maxwell's equations are discretized subject to specific boundary conditions and the resulting set of discrete equations are numerically solved. Ray tracing methods are based on geometrical optics and are often more useful in scenarios where the feature sizes of the scatterers are large compared to the wavelength of the radio signals. Computing radio channel path loss predictions for deployment of large wireless networks using FDTD methods requires huge grid sizes to ensure numerical accuracies of the final solutions. Similarly, in a ray tracing model, the number of rays need to be proportional to that of signal receivers. Both make the underlying computational problem very large. On the other hand, the input data that describes the physical geometry of the study site is very often of low precision and prone to large errors. As a result, despite their large computational overhead, such high-precision techniques are unable to make accurate predictions. An alternative event driven approach that is based on a transmission line matrix (TLM) method was proposed in [8], [9] to bridge the gap between low precision input data and accuracy considerations. The authors of [9] have shown empirical runtime performance results of earlier traditional methods that clearly motivate the need for alternative models such as their new, event-based TLM model. A TLM method uses equivalent electrical networks that are based on the link between field theory and circuit theory to solve certain types of partial differential equations stemming in EM field problems. Parallelizing the event driven TLM approach proposed in [9] renders the methodology applicable to larger simulations of radio channel propagation that can include a greater number of receivers with extended geographical reach. For example,



(a)



(b)

Figure 1. Voltage profile along the $z = 15$ plane at two different time instants (a) $t = 50$ s (b) $t = 75$ s during a simulation with only one voltage source at the center of a $30 \times 30 \times 30$ domain.

while serial execution is sufficient to deal with room-sized volumes, parallel execution enables signal strength estimation from city block-sized urban scenarios to even larger volumes encountered in wider, mountainous terrains. A number of FDTD and ray-tracing based methods are available for the simulation of radio signals. But, as mentioned earlier, inherent lack of precision in the input data renders such methods largely ineffective. An earlier attempt [1] to parallelize the discrete event formulation of the TLM-based method in [8] only exhibited limited scalability, with self-relative parallel speedup reported upto 25 processors.

C. Model Description

The computational (simulation) domain is modeled by a three-dimensional (3D) grid. Each grid point i is a node that computes the time-varying electrical potential V_i of the wave that is traveling through the grid. Partial voltages x_{ij} and x_{ji} are defined across each link in the grid that connects two neighboring nodes i and j in the directions $i \longrightarrow j$ and $j \rightarrow i$, respectively. Partial voltages on the links capture information related to the permittivity and permeability constants of the medium, which in turn define the rate at which the wave travels between those two nodes. In this paper, the term *voltage* will always be used to refer to the total time-varying voltage defined at a node (grid point) while *partial voltage* will always be defined on links between two neighboring nodes. Note that an $n \times n \times n$ grid contains n^3 grid points (voltages) and $N = 6n^3$ directional links (partial voltages). When the power at any point in the simulation domain falls below a cut-off voltage, a radio antenna cannot detect it. This effect is captured in terms of a *threshold voltage* below which a node is not required to transmit. The TLM equations, as defined in [9], that govern the propagation of radio signals in terms of the total and partial voltages defined above are:

$$x_{ij}^{t+1} = R_{ij}\left(\frac{V_i^t}{3} - x_{ij}^t\right) + T_{ji}\left(\frac{V_j^t}{3} - x_{ji}^t\right) \quad (1)$$

$$V_i^{t+1} = \sum_{k=0}^{5} x_{ik}^{t+1} \quad (2)$$

where k corresponds to the indices of the six neighbors of the grid point indexed by i, and t and $t+1$ are consecutive units of discretized time. The constants R_{ij} and T_{ji} are the reflection and transmission coefficients that correspond to the links ij and ji, respectively. These constants encapsulate properties such as the permittivity and permeability of the medium that is modeled by the grid. We will use the term *components* of V_i to refer to the partial voltages that add up to yield V_i through Eqn (2). A computation of the voltage profile (set of total voltages across all the n^3 nodes in the grid [see Fig. 1]) at a time step t requires the availability of all the partial voltages at the previous time step.

D. Event-driven Execution

Temporal updates of the voltage profile can be either *time-driven* or *event-driven*. Time-driven approaches continuously update the set of partial and total voltages after the passage of each pre-defined time interval (which is often constrained by convergence requirements such as the aspect ratio of finite-difference schemes). In event-driven approaches, the state of a physical system changes only at certain instants of time through instantaneous transitions. An *event* is associated with each such transition. For example, in the aforementioned problem, an event can be triggered every time a node exceeds the threshold voltage. Discrete event formulations, therefore, delink the necessity for a global clock from the evolution of the physical system and instead views the same simulation as a set of time-stamped events (containing temporal information about the physical state variables) that are processed as efficiently as possible without violating global causality. When such event-based simulations are distributed across multiple processors, preserving global causality becomes very challenging as data dependencies across processors are no longer guaranteed to be concurrent. Parallelizations of such discrete event algorithms have been known to be very complicated, often requiring causality control mechanisms that are highly challenging to scale well across a large number of processors. A more detailed discussion of parallel discrete event simulations (PDES) can be found in [6], [11].

E. Parallel Execution Challenge

FDTD and related techniques conform to time-driven algorithms. Parallelization of the newer TLM-based model built on barrier methods results in the processors becoming too tightly coupled, thereby, diminishing the returns of the event-based paradigm. Parallelizing event-based algorithms, while relieving the tight coupling, exhibit a different challenge, most notably distributed causality preservation. The state-of-the-art solution to parallel event-based models is the use of optimistic simulation techniques employing "reverse computation" as the rollback method, for maximal parallelism with minimal overhead. Problems such as our parallel event-based execution of TLM are best suited to utilize such as method. We use such reverse computing techniques to minimize the now well-understood overheads associated with traditional PDES approaches [3], [11]. Another important parallelization challenge is the treatment of 3D, which imposes interesting dynamics coupled with event-based behavior and domain decomposition. Our interest is in supporting full 3D scenarios with multiple domain decomposition schemes that scale across thousands of processors. As such, in addition to those that are already inherent to PDES, the challenges undertaken in this work include designing and developing an efficient, perfectly-reversible parallelization of the novel serial model in [9], and realizing full 3D support under different parallel domain decomposition schemes.

F. Contributions in this paper

To the best of our knowledge, this paper presents the first parallel discrete event formulation of radio signal propagation that scales to thousands of processors. Our approach is based on a reverse computing technique with full 3D support for realistic scenarios. Though our algorithm can efficiently support multiple domain partitioning schemes, the results presented here are based on only one due to space limitations. It may also be noted that our algorithm exhibits potential for vector processing. In addition, its applicability is not confined to only TLM-based problems such as the one described above. Finite differencing parabolic partial differential equations (*e.g.,* diffusion equations) that arise in a multitude of scientific applications, can also be solved within the parallel execution framework presented here. We implement our algorithm on a Cray XT4 platform and demonstrate its scalability to thousands of processors with speedups over $1000\times$. This enables real-time deployment capability with turn around times that are commensurate with time scales for mobile wireless signal strength predictions.

The rest of the paper is organized as follows. Section II describes the parallel discrete event scheme underlying our algorithm. Our experimental setup and performance results are discussed in Sections III and IV followed by a discussion of the future scope of this work in Section V.

II. Parallel Discrete Event Scheme

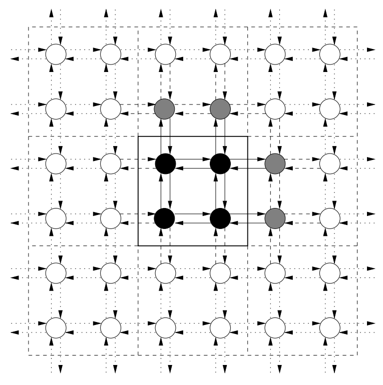
A. Domain Decomposition

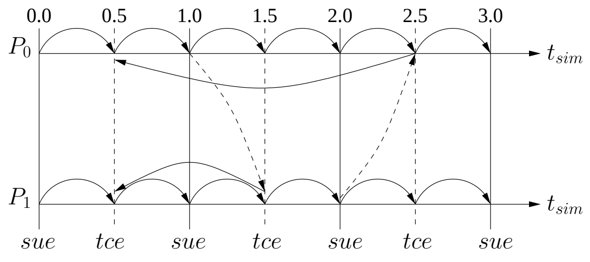
It is clear from Eqn (1) and Eqn (2), which we will refer to as the *forward* equations, that a good parallel domain decomposition for this problem is one in which: (a) for each x_{ij} that is local to a processor, x_{ji} is also local and (b) for each V_i that is local to a processor, as many components of V_i are local as is possible. Guided by this observation, we block partition the 3D grid across P processors arranged in a Cartesian $P_x \times P_y \times P_z$ topology. Each processor is therefore responsible for $n^3/P_xP_yP_z = n^3/P$ voltages (one for each local node). For each local node, a processor is responsible for the six partial voltages defined on the links connecting it to its nearest neighbors along the positive x, y and z directions only. Thus, each processor is responsible for $6n^3/P = N/P$ number of partial voltages.

For ease of presentation, we use a 2D example in Fig. 2(a) to illustrate the following notation that will be adopted in the remainder of this paper:

- X_L : set of all partial voltages local to a processor [bold arrows in Fig. 2(a)].
- V_L : set of all total voltages local to a processor [black circles in Fig. 2(a)].
- V_R : set of all remote total voltages required for the computation of all $x_{ij} \in X_L$ [gray circles in Fig. 2(a)].
- V_U : $V_L \cup V_R$.
- X_R : set of all remote partial voltages required for the computation of all $V \in V_U$ [dashed arrows in Fig. 2(a)].

The preceding parallel domain decomposition guarantees that (a) for each local partial voltage $x_{ij} \in X_L$, the reverse partial voltage is also local, i.e., $x_{ji} \in X_L$ (b) for each total voltage $V_i \in V_L$, its components along the positive directions are guaranteed to belong to X_L (c) the number of sending and receiving processors are both constants (d) the partial voltages defined on links that cut a processor's domain boundaries along the positive directions are local and (e) the inter-processor communication bandwidth is proportional to the surface area of each block partition and, hence, $O\left(\frac{N^{2/3}}{P^{2/3}}\right)$.


(a)



(b)

Figure 2. (a) A 2D illustration of the unique set V_U (set of filled circles), the local set X_L (set of bold arrows) and remote set X_R (set of dashed arrows) for the processor responsible for the solid square in the center. (b) Optimistic parallel discrete event processing in an example with two processors.

In our formulation, the partial voltages which are mapped to a processor are evolved through two types of event processing, namely, *self-update events (sue)* and *threshold-cross events (tce)*. An example with two processors is shown in Fig. 2(b). Self-update events are processed at integral time-stamps t while threshold-cross events are processed at half-integer time-stamps $t + \frac{1}{2}$. When a self-update event is processed, each $x_{ij} \in X_L$ is updated though Eqn (1). Those local updates for which the results vary by more than a pre-defined threshold value are sent to the appropriate destination processors in a message timestamped with $t+\frac{1}{2}$. This style of sending conditionally will be referred to as *selective sends*. The receiving processors process the arrival of the updates as threshold-cross events. Processing a threshold cross event involves modifying the set X_R according to the updates received. Note that such selective sends result in an *asynchronous* communication pattern. A numerically correct self-update of X_L requires concurrent values of $V_i, V_j \in V_U$. However, V_i and V_j may depend on remote partial voltages $x_{ik} \in X_R$. Thus, correctness of self-updates depend upon the concurrency of the sets X_L, X_R and V_U.

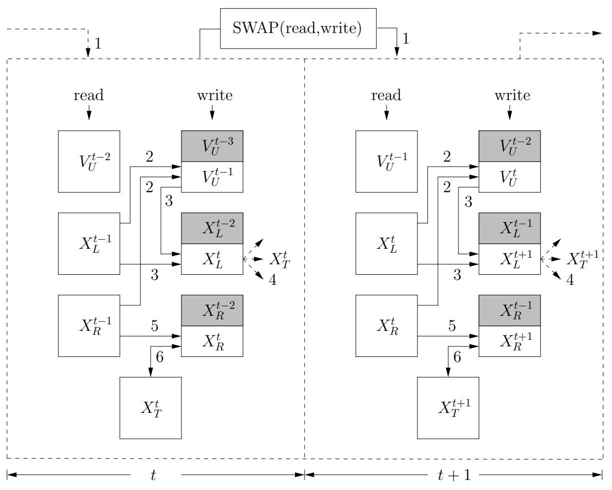
In our approach, forward execution is carried out *optimistically*, *i.e.*, each processor continues to execute forward in simulation time under the assumption that the set X_R that contains the remote data necessary for local forward computations (via self-update events) are locally-usable, correct values until a threshold cross event with a more recent timestamp is processed. As part of processing such a threshold-cross event, a rollback to the appropriate simulation time in the past is initiated.

The state variables defined by the sets X_L, X_R and V_U contain complete information about the local portion of the domain for which a processor is responsible. These sets are stored as arrays. Note that $|V_U| = \Theta(N/P)$ and $|X_L \cup X_R| = \Theta(N/P)$. In addition, two pointers, labeled *read* and *write* pointers are maintained. At any given simulation time t, each processor maintains the following state variables: V_U^{t-2}, X_L^{t-1} and X_R^{t-1} that are pointed to by the read pointer and V_U^{t-1}, X_L^t and X_R^t that are pointed to by the write pointer (see Fig. 3(a)). The above two pointers are maintained by each processor in order to facilitate reverse computing for rollbacks, as will become clearer in the next section. Operations performed during a forward execution overwrite the arrays pointed to by the write pointers. In Fig. 3, the arrays that are overwritten are indicated by the gray boxes pointed to by the write pointers.

B. *Forward Execution*

Forward execution of our discrete event approach in terms of self-update and threshold-cross events are shown in Fig. 3(a). The following operations execute the forward code:

1) SWAP (read,write) : The pointers to the read and write copies of the state variables are swapped.
2) UPDATE-V_U^{t-1} : V_U^{t-1} is computed using X_L^{t-1} and X_R^{t-1} through Eqn (2).
3) COMPUTE-X_L^t : X_L^t is computed from the X_L^{t-1} and V_U^{t-1} using Eqn (1).
4) SELECTIVE-SEND : To each processor that needs $x_{ij}^t \in X_L^t$, send x_{ij}^t iff $| x_{ij}^t - x_{ij}^{t-1} | \geq \delta$, where δ is a pre-defined threshold. All such partial voltages that are destined for a particular destination are collected and sent in a single message.
5) COPY-X_R : Copy X_R^{t-1} to X_R^t.
6) PROCESS-TCE : This operation is performed if and only if there is a pending threshold-cross event. The operation SWAP (X_R, X_T) is performed when the pending threshold-cross event has a current time-stamp. In this operation, partial voltages $x_{ij}^t \in X_T^t$ that are received from the sending processors are swapped with the corresponding values in X_R^t (see Fig. 3(a)). A threshold-cross event with a future time-stamp is held in queue to be processed later. If the pending threshold-cross event has a past time-stamp, then a rollback is carried out to restore the state variables to


(a)

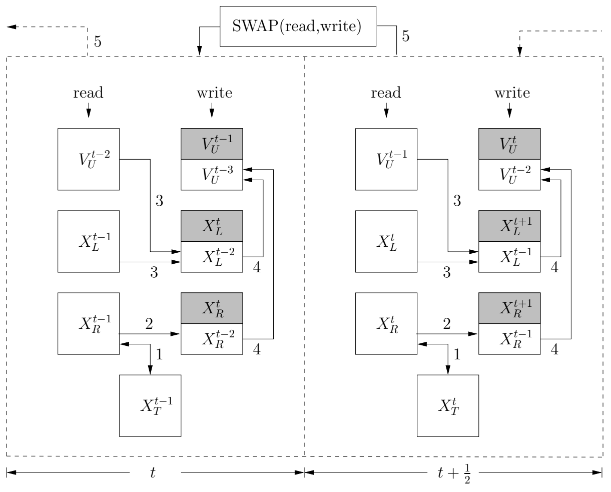

(b)

Figure 3. (a) Forward execution. (b) Reverse execution. Numbers on the arrows indicate the order in which the indicated steps are executed in a forward event or its reversal. The gray boxes indicate the arrays pointed to by the write pointer before they are overwritten by the arrays in the white boxes.

their values at that time. We discuss rollbacks in the next section.

C. Reverse Execution

In our implementation, restoration of state upon rollbacks is realized through reverse computing. Recall that when a rollback is initiated by a threshold-cross event that is processed at time-stamp $t+\frac{1}{2}$, the physical system needs to be restored to that corresponding to simulation time t which is defined by the arrays V_U^{t-2}, X_L^{t-1} and X_R^{t-1} pointed to by the read pointers and the arrays V_U^{t-1}, X_L^t and X_R^t pointed to by the write pointers. The following operations perform the reverse execution as illustrated in Fig. 3(b):

1) UNDO-PROCESS-TCE: Note that due to the most recent SWAP (read,write) operation in the forward execution, the arrays V_U^{t-1} and X_L^t currently pointed to by the read pointer hold the same elements as the arrays V_U^{t-1} and X_L^t when it was pointed to by the write pointer in the preceding time-stamp (see Fig. 3(b)). The array X_R^t, however, may need to be restored explicitly since the most recent threshold-cross event, if there was one, could have swapped out some of its elements with X_T^t. Thus, reversing the forward threshold-cross event involves swapping back the values of X_R^t with those in X_T^t thereby restoring X_R^t.
2) RESTORE-X_R : The X_R^t is copied to the array X_R pointed to by the write pointer. This reverses the operation in step 5 of Section II-B and restores X_R^{t-1}.
3) RESTORE-X_L^{t-1}: Note that in the forward execution, the local partial voltages X_L^t are computed from X_L^{t-1} and V_U^{t-1}. Therefore, we need a function g such that $X_L^{t-1} = g(X_L^t, V_U^{t-1})$. To find g, we treat x_{ij}^{t-1} and x_{ji}^{t-1} as two unknowns and solve the following two forward equations:

$$x_{ij}^t = R_{ij}\left[\frac{V_i^{t-1}}{3} - x_{ij}^{t-1}\right] + T_{ji}\left[\frac{V_j^{t-1}}{3} - x_{ji}^{t-1}\right]$$

$$x_{ji}^t = R_{ji}\left[\frac{V_j^{t-1}}{3} - x_{ji}^{t-1}\right] + T_{ij}\left[\frac{V_i^{t-1}}{3} - x_{ij}^{t-1}\right]$$

which can be re-written as:

$$MX^{t-1} = \frac{1}{3}MV^{t-1} - X^t$$

where

$$M = \begin{pmatrix} R_{ij} & T_{ji} \\ T_{ij} & R_{ji} \end{pmatrix} \quad , \quad X^{t-1} = \begin{pmatrix} x_{ij}^{t-1} \\ x_{ji}^{t-1} \end{pmatrix}$$

$$V^t = \begin{pmatrix} V_i^{t-1} \\ V_j^{t-1} \end{pmatrix} \quad , \quad X^t = \begin{pmatrix} x_{ij}^t \\ x_{ji}^t \end{pmatrix}$$

The above equation can be solved to yield:

$$X^{t-1} = \frac{1}{3}V^{t-1} - MX^t$$

where we have used the fact that $M^{-1} = M$ (see Appendix). Thus, the reversal equation to restore X_L^{t-1} using X_L^t and V_U^{t-1} is:

$$x_{ij}^{t-1} \leftarrow \left[\frac{V_i^{t-1}}{3} - R_{ij}x_{ij}^t - T_{ji}x_{ji}^t\right] \qquad (3)$$

At this point, the read pointers point to the correct values of V_U^{t-1}, X_L^t and X_R^t and write pointers point to the correct values of X_L^{t-1} and X_R^{t-1}. The correct values of V_U^{t-2} still need to be restored.

4) RESTORE-V_U^{t-2} : Consider the following equations for a pair (i, j) of nearest neighbors:

$$\begin{aligned} x_{ij}^{t-1} &= R_{ij} y_{ij}^{t-2} + T_{ji} y_{ji}^{t-2} \\ x_{ji}^{t-1} &= T_{ij} y_{ij}^{t-2} + R_{ji} y_{ji}^{t-2} \end{aligned}$$

where $y_{ij}^t = \left(\frac{V_i^t}{3} - x_{ij}^t \right)$. Solving the above equations for the two unknowns y_{ij}^{t-2} and y_{ji}^{t-2} yields:

$$y_{ij}^{t-2} = R_{ij} x_{ij}^{t-1} + T_{ji} x_{ji}^{t-1}$$

Summing up over all the neighbors of point i, we get:

$$\begin{aligned} \sum_k y_{ik}^{t-2} &= \sum_k \left[R_{ik} x_{ik}^{t-1} + T_{ki} x_{ki}^{t-1} \right] \\ \sum_k \left(\frac{V_i^{t-2}}{3} - x_{ik}^{t-2} \right) &= \sum_k \left[R_{ik} x_{ik}^{t-1} + T_{ki} x_{ki}^{t-1} \right] \\ V_i^{t-2} &= \sum_k \left[R_{ik} x_{ik}^{t-1} + T_{ki} x_{ki}^{t-1} \right] \end{aligned}$$

Thus, we have:

$$V_i^{t-2} = \sum_k \left[R_{ik} x_{ik}^{t-1} + T_{ki} x_{ki}^{t-1} \right] \qquad (4)$$

At this point, the read pointer points to the correct arrays V_U^{t-1}, X_L^t and X_R^t and the write pointer points to the correct arrays V_U^{t-2}, X_L^{t-1} and X_R^{t-1}.

5) SWAP (read,write) : The read and write pointers are swapped to restore the state to the previous time-stamp (see Fig. 3(b)).

Since each partial voltage is updated exactly once, the runtime for each reversal (steps 1 through 5 above) is $O(N/P)$, as it is for each forward execution phase.

III. Experimental Setup

A. Software Platform

We implemented the discrete event execution using the μsik engine[10]. μsik provides an application programming interface in the C++ language that supports the concept of logical processes, events for exchanging timestamped messages among logical processes, and virtual time-synchronized delivery of events to logical processes. The API invokes a callback method into the logical processes when an event is to be processed. Another callback method is invoked if and when an event is to be undone, which could be either due to violation of timestamp order as a result of optimistic processing or due to cancellation of an event by the sender of that event. We use the event handler and undo handler to realize the forward and reverse execution portions of the updates to partial and total voltages. The send primitive is used to send threshold crossing events to remote processors, and also to schedule a self update to advance the local partial voltages. Specific care is taken to only pack data corresponding to the local data that have actually exceeded the threshold since the last update sent to neighboring processors. This ensures that the number of updates across processors is minimized while keeping the performance competitive with non-event-based execution.

Our implementation allows for any number of partial voltages to be mapped to the same logical process. This feature is critical to minimize the event processing overhead. In a simpler scheme adopted before [1], [9], only one grid point is mapped to a logical process, which can make the event overhead a significant part of the total runtime. Our scheme allows for multiple partial voltages to be updated as a block, making it competitive with an optimized, sequential execution. Please note that reverse execution is more challenging in our scheme, as we cannot rely on expensive state-copying primitives to restore state upon rollback.

For the best performance, we map one logical process per processor core, which is empirically observed to deliver better performance compared to when more than one logical processes are instantiated per core. μsik internally handles inter-processor communication to exchange timestamped events across processors and to synchronize global virtual time. We configured μsik to use the vendor-supplied Message Passing Interface (MPI) implementation native to the hardware platform.

B. Performance Metric

It is important to note that the traditional “event rate” performance metric of discrete event simulators is not relevant for the purposes of measuring efficiency in this application. Since events can be defined in many ways (consequently, with different granularity), the number of events is misleading. Instead, we use the more appropriate measure, namely, the speedup achieved by a parallel execution, relative to the execution time on the smallest core count that can be used to execute the scenario.

C. Hardware Platform

Our hardware platform is a Cray XT4 machine in which each compute node contains a quad-core 2.1 GHz AMD Opteron processor with 8 GB of memory. The nodes are connected via a high-bandwidth SeaStar interconnect. Internally, the MPI implementation is based on Cray’s implementation of Portals 3.3 messaging interface.

IV. Performance Results

A. Scenarios

In the experiments, we exercised our scheme with three grid sizes, corresponding to increasingly larger volumes encountered in wireless applications. The first is a medium-sized grid with $n = 30$, giving 27000 total voltages and 162,000 partial voltages. The second is a larger-sized grid with $n = 80$, giving roughly half a million total voltages and 3 million partial voltages. The third, which is a very large scenario with $n = 130$, yields roughly 2 million total voltages and 13 million partial voltages. These grid

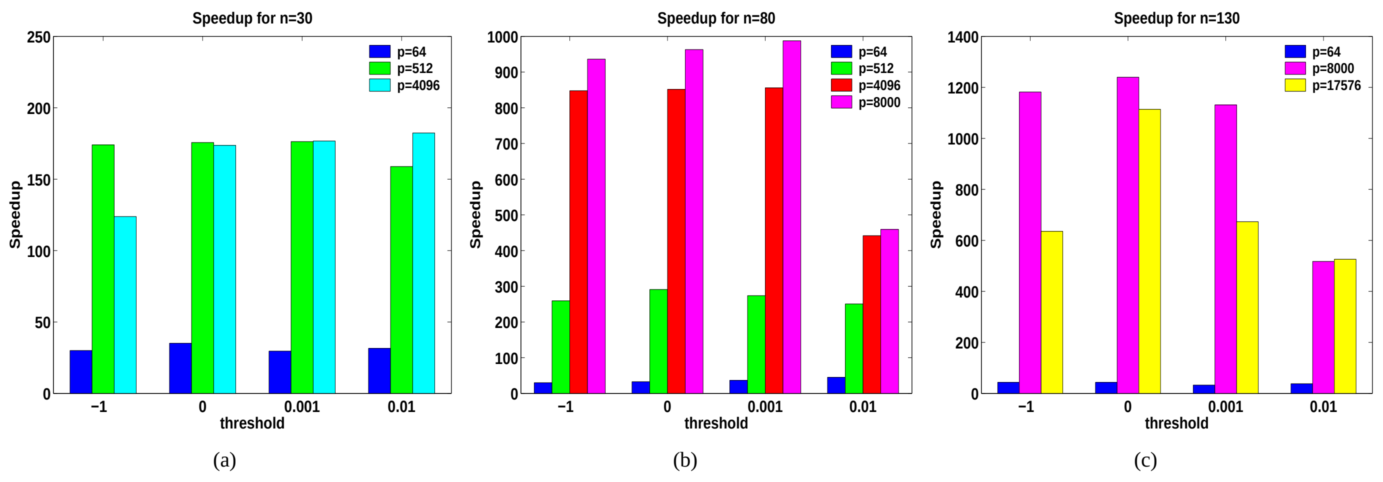


Figure 4. Speedup for various problem sizes, $n = 30, 80$ and 130, with varying threshold and number of processor cores.

sizes were tested with different threshold values, to exercise the performance effects of selective sends (asynchronous communications).

B. Speedup Analysis

Speed-up plots for the different scenarios are shown in Fig. 4. It demonstrates parallel speedups over $1000\times$ for large scale scenarios on thousands of processors. As described before, the communication pattern of our algorithm is intimately dependent on the threshold voltage value. When threshold=-1, every self-update event triggers a threshold-cross event resulting in a very synchronous communication pattern. This is akin to a time-driven algorithm. As the threshold increases, communication becomes increasingly asynchronous due to selective sends. Increasing selectivity of sends increases concurrent computations while decreasing the total communication. This, in turn, improves the performance of the algorithm. Performance advantages of selective sends with increasing number of processors can be seen in Fig. 4 for all three scenarios. When the threshold becomes large, the wave propagates shorter grid cell distances[1]. This effect is seen from the speedups at threshold=0.01 for $n = 80$ and $n = 130$. For relatively large problem sizes on small number of processors, computations remain highly concurrent and the computation-to-communication ratio is large. For such cases, the parallel runtimes remain largely unaffected by selective sends. This is evident from the near insensitivity of the speedups for the three scenarios on $p = 64$.

Efficiency (defined canonically as $speedup/P$) of the algorithm exhibits very good improvements when problem sizes are increased for a fixed number of processors. For example, efficiency of the parallel execution on $p = 512$ increases from 34% to 53% when the scenario changes from a medium sized grid to a large one (n changes from 30 to 80) while it increases to 21% from 4% on $p = 4096$ for the same change in the problem size. This trend continues across all the threshold values that were tested.

[1]This can be understood by considering the asymptotic limit with threshold=∞ when the simulation ends after the very first self-update event (since the difference between the new and old value of the partial voltages will always be less than ∞).

C. Implications

The demonstrated parallel speedup of the algorithm makes it possible for real time prediction of radio signal strength. For example, a serial computation based prediction for a scenario with $n = 80$ (roughly half a million total voltages) has a turn around time of about 3.5 hours but only about 6 seconds on $p = 8000$ processors using the above algorithm. This is well within the scope of real time predictions of mobile wireless signal strength in cluttered 3D terrains.

V. Conclusions and Future Work

We presented an efficient parallel implementation of a recently developed discrete-event based serial algorithm for the estimation of radio wave signal strength. We used a reverse computing based discrete event approach for this problem, aimed at circumventing other PDES approaches that are known to suffer from overheads that do not scale well to large processors counts. We explicitly derived the reversal equations that were subsequently used for rollbacks to restore the state of the system to a desired time in the past. We demonstrated that such reverse computing based rollbacks can deliver unprecedented speedup for this problem. To the best of our knowledge, our results are also among the first to demonstrate $1000\times$ parallel speedup for any non-synthetic PDES application that is based on reverse computation. Also, such speedups for EM wave simulators have never been reported before. We showed that the algorithm presented in this paper brings real time signal strength predictions well within the turnaround time scales needed for mobile wireless deployment simulations and design problems. Additionally, the effect of varying threshold values on the performance of the algorithm was studied systematically to understand their effect on the performance. The algorithm supports full 3D scenarios with support for rich heterogeneity.

An important issue that remains a subject of our ongoing investigation is an exhaustive performance comparison of conventional time-driven parallel approaches with the event-driven parallel algorithm presented here. Note that unlike discrete event based schemes, barriered time-driven algorithms are prone to synchronization overheads. This point of comparison is particularly poignant in an era of petascale computers whose synchronization overheads can drastically hinder the performance of barriered time-driven codes.

ACKNOWLEDGEMENTS

This paper has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the U.S. Department of Energy. Accordingly, the United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725.

This effort has been supported by research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

REFERENCES

[1] D. Bauer and E. Page. Optimistic Parallel Discrete Event Simulation of the Event-Based Transmission Line Matrix Method. In *Proc. of the Winter Simulation Conference*, pages 676–684, 2007.

[2] S. D. Bilbao. *Wave and Scaterring Methods for Numerical Simulations*. Wiley, 2004.

[3] C. Carothers, K. S. Perumalla, and R. M. Fujimoto. Efficient Optimistic Parallel Simulations using Reverse Computation. *ACM Transactions on Computer Modeling and Simulations*, 9(3):224–253, 2006.

[4] D. Cavin, Y. Sasson, and A. Schiper. On the Accuracy of MANET Simulators. In *Proc. of the Int'l Workshop on Principles of Mobile Computing*, pages 38–43, 2002.

[5] J. Chen and S. Hall. Efficient and Outdoor EM Wave Propagation in a Compact Terrain Database of the Urban Canyon Environment. In *Proc. of the Vehicular Technology Conference*, pages 802–806, 2002.

[6] R. M. Fujimoto. Parallel Discrete Event Simulation. In *Proc. of the Winter Simulation Conference*, pages 19–28, 1989.

[7] I. Gruber and H. Li. Behavior of Ad Hoc Routing Protocols in Metropolitan Environments. In *Proc. of the Vehicular Technology Conference*, pages 3175–3180, 2004.

[8] J. Nutaro. A Discrete Event Method for Wave Simulation. *ACM Transcations on Modeling and Simulations*, 16(2):174–195, 2006.

[9] J. Nutaro, T. Kuruganti, R. Jammalamadaka, T. Tinoco, and V. Protopopescu. An Event Driven, Simplified TLM Method for Predicting Path-loss in Cluttered Environments. *IEEE Trans. on Antennas and Propagation*, 56(1):189–198, 2008.

[10] K. S. Perumalla. μsik – A Micro-Kernel for Parallel and Distributed Simulation Systems. In *Proc. of the Workshop on Parallel and Distributed Simulation*, pages 59–68, 2005.

[11] K. S. Perumalla. Parallel and Distributed Simulation: Traditional Techniques and Recent Advances. In *Proc. of the Winter Simulation Conference*, pages 84–95, 2006.

[12] K. A. Remley, A. Weisshaar, and H. R. Anderson. A Comparison Study of Ray Tracing and FDTD for Indoor Propagation Modeling. In *Proc. of the Vehicular Technology Conference*, pages 865–869, 1998.

APPENDIX

In the TLM model, the reflection and transmission coefficients along a link $i \to j$ on the grid is defined as (see [9]):

$$R_{ij} = \frac{Z_i - Z_j}{Z_i + Z_j} \qquad \text{and} \qquad T_{ji} = \frac{2Z_j}{Z_i + Z_j} \qquad \text{(A-I)}$$

where Z_i and Z_j are material specific impedance values assigned to the grid points i and j. This yields the following relations:

$$R_{ij} = -R_{ji} \qquad \text{and} \qquad R_{ij} + T_{ji} = 1 \qquad \text{(A-II)}$$

Using Eqn (A-I) and Eqn (A-II) in M, we have:

$$\begin{aligned} |M| &= R_{ij}R_{ji} - T_{ji}T_{ij} \\ &= R_{ij}R_{ji} - [(1 - R_{ij})(1 - R_{ji})] = -1 \end{aligned}$$

Therefore, the inverse of M is:

$$\begin{aligned} M^{-1} &= \frac{1}{|M|} \begin{bmatrix} R_{ji} & -T_{ji} \\ -T_{ij} & R_{ij} \end{bmatrix} = \begin{bmatrix} -R_{ji} & T_{ji} \\ T_{ij} & -R_{ij} \end{bmatrix} \\ &= \begin{bmatrix} R_{ij} & T_{ji} \\ T_{ij} & R_{ji} \end{bmatrix} = M \end{aligned}$$

Hence, for each link ij, $M = M^{-1}$.