Reversible Parallel Discrete Event Formulation of a TLM-based Radio Signal Propagation Model

SUDIP K. SEAL
and
KALYAN S. PERUMALLA
Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA

Radio signal strength estimation is essential in many applications, including the design of military radio communications and industrial wireless installations. For scenarios with large or richly-featured geographical volumes, parallel processing is required to meet the memory and computation time demands. Here, we present a scalable and efficient *parallel* execution of the *sequential* model for radio signal propagation recently developed by Nutaro et al. Starting with that model, we (a) provide a vector-based reformulation that has significantly lower computational overhead for event handling, (b) develop a parallel decomposition approach that is amenable to reversibility with minimal computational overheads, (c) present a framework for transparently mapping the conservative time-stepped model into an optimistic parallel discrete event execution, (d) present a new reversible method, along with its analysis and implementation, for inverting the vector-based event model to be executed in an optimistic parallel style of execution, and (e) present performance results from implementation on Cray XT platforms. We demonstrate scalability, with the largest runs tested on up to 127,500 cores of a Cray XT5, enabling simulation of larger scenarios and with faster execution than reported before on the radio propagation model. This also represents the first successful demonstration of the ability to efficiently map a conservative time-stepped *model* to an optimistic discrete-event *execution*.

Categories and Subject Descriptors: I.6.8 [**Simulation and Modeling**]: Parallel Discrete Event Simulation

General Terms: Rollback, Reverse Computing, Transmission Line Matrix

Contents

1. INTRODUCTION

Estimation of radio signal path loss is very important in the design and deployment of wireless communication networks [Chen and Hall 2002]. In military scenarios, typical geographical terrains of interest are large and often include physical features that range from buildings and mountains to natural reliefs and foliage. For scenarios with even a single source and a few receivers, traditional techniques exhibit large run times [Nutaro et al. 2008]. Faster simulation methods become necessary to handle large numbers of transmitters or receivers, or when the radios are mobile. Efficient real time estimation of radio signal strength for such scenarios remains an area of on-going research [Cavin et al. 2002; Gruber and Li 2004].

Finite-difference time-domain (FDTD) [Taflove and Hagness 2000] or ray-tracing models [Levy 2000] of radio wave propagation in such terrains are computationally very intensive, more so as the number of transmitters and receivers is increased. In FDTD methods, Maxwell's equations are discretized subject to specific boundary conditions and the resulting set of discrete equations is 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 signal path loss predictions for deployment of large wireless networks using FDTD or ray-tracing require very fine spatial resolution (hence very large grid sizes) and commensurately accurate initial conditions to ensure numerical accuracies of the final solutions. This makes the underlying computational problem very large. On the other hand, the input data and the initial conditions that describe the physical geometry of such cluttered study sites are very often of low precision and prone to large errors. As a result, despite their large computational effort, such high-precision techniques are unable to make accurate predictions. For additional details, see [Nutaro 2006; Nutaro et al. 2008].

To bridge the gap between low precision input data and accuracy considerations, an alternative event driven approach that is based on a transmission line matrix (TLM) method [Sadiku 2000] was proposed in [Nutaro et al. 2008]. A TLM method uses equivalent electrical networks that are based on the link between field theory and circuit theory to solve simplified partial differential equations stemming in electromagnetic field problems. Nutaro et al have shown empirical runtime performance comparison with earlier traditional methods that clearly motivate alternative models such as their new, event-based TLM model [Nutaro et al. 2008].

Parallel execution of the TLM approach becomes necessary when larger simulations of radio signal propagation are required to include a greater number of receivers with extended geographical reach. For example, 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. In addition, the need for parallel execution becomes obvious when real-time signal strength estimation is sought. For example, when transmitters and receivers are mounted on moving vehicles, the turnaround times for simulation of such loss computations must match the time scales that are associated with mobile platforms.

1.1 Parallel Execution Challenge

Several challenges are encountered when designing a parallel implementation of the above model to scale to hundreds of thousands of processors.

Of paramount importance to the scalability of parallel simulations is the nature of inter-processor coupling across simulation time. Processors in conventional time-stepped simulations are tightly-coupled and typified by data exchange between inter-dependent processors at every time-step. The sequential TLM model (to be described in more detail shortly) is grid-based, inherently time-stepped and affords selective updates of the state variables, that is, state variable are updated only if they differ from their previous values by a pre-defined threshold. Since the updates to state variables in the TLM model are threshold-based, one of the first parallel execution challenges is how best to exploit such selective updates of distributed data to relax inter-processor couplings resulting from delayed interactions of logical processes (LP).

Mapping the time-stepped TLM model to a discrete event based execution exposes the issue of computational granularity that results from the mapping [Bailey et al. 1994; Choi and Chung 1995; McBrayer and Wilsey 1995]. LPs and events need to be defined such that the overhead of computation local to an LP offsets the processing overhead of events defined on them. In a straightforward conversion to a discrete-event execution, each grid cell of the TLM model can be mapped to an LP. This is the approach adopted in [Bauer and Page 2007; Bauer et al. 2009]. As will be demonstrated later in this paper, such a mapping is highly sub-optimal in terms of scalability and yields unacceptably large execution times for signal strength estimation. Thus, a judicious definition of LP (which sets the granularity of subsequent computations) is key to achieving high efficiency of any large scale parallel implementation of the TLM model.

Optimistic execution is one of the state-of-the-art methods to relax inter-processor coupling in large scale parallel discrete event simulations (PDES). In particular, the advantages of reverse-computing based approaches over other optimistic PDES methodologies are well-known [Carothers et al. 1999; Perumalla 2006]. For this, however, one is posed with the challenge of developing a new reversible model from a forward-only model – a task that is often highly non-trivial as will be shown in the context of the TLM model shortly. Ill-designed reverse-computing approaches result in poor scalability as the overhead of reverse-computing can quickly offset its benefits. Formulating a reversible parallel approach is a formidable challenge to ensure that the asymptotic runtime does not exceed that of the forward TLM execution while requiring minimal extra memory.

Another important parallelization issue is the treatment of three dimensional (3D) geography, which imposes complex inter-processor dependencies, more so in this case since the dynamics are coupled with both the event-based behavior and the distribution of data across processing elements (parallel domain decomposition). Our interest is in supporting full 3D scenarios with multiple domain decomposition schemes that scale across hundreds of thousands of processors.

1.2 Contributions in this Paper

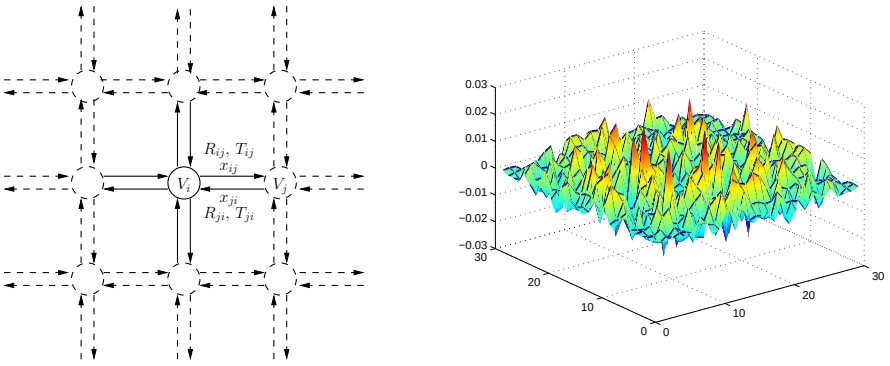
In this paper, we present a reversible vector-update reformulation of the event-based TLM model such that it can be executed optimistically in parallel. Our reformulation significantly lowers the computational overhead for event handling. We implement a parallel domain decomposition approach that allows for reverse computing with minimal computational and memory overhead. In conjunction, the resulting parallel execution is rendered coarse-grained with a computation-to-communication ratio that is kept well under control even up to hundreds of thousands of cores. Although, our implementation allows for any number of partial voltages to be mapped to the same LP, we adopt a coarse-grained mapping that allows multiple partial voltages to be updated as a block. This makes the parallel execution more competitive with an optimized, sequential execution. On the other hand, reverse execution becomes more challenging since expensive state-copying primitives save entire state for restoration upon rollback. To resolve this, we develop a fully reversible model of the original forward-only sequential method that can be executed in an optimistic parallel manner. Tight-coupling of processors in traditional time-stepped parallel simulations is also avoided, as a natural consequence. We present a transparent mapping of the original conservative time-stepped model to an optimistic parallel discrete event execution. Comparisons of our approach with other recently reported parallelization efforts are also presented to highlight the differences in parallel performance that can result from varying definitions of logical processes and, in turn, the granularity of computation. The performance comparison also underscores the inadequacy of event rate as a measure of parallel performance for this problem, despite its common usage in PDES. In addition, the comparisons highlight the significant advantage that our approach delivers over those reported until the time of this writing. We present exhaustive performance results of our implementation on both Cray XT4 and Cray XT5 platforms. Finally, to the best of our knowledge, the work presented in this paper embodies the first successful PDES of a non-trivial application that scales to over 100,000 processor cores.

1.3 Organization of this Paper

The rest of the paper is organized as follows. Section 2 describes the vector-update formulation that is at the core of both sequential and parallel reformulations of the radio wave propagation problem. It covers time-stepped as well as discrete event execution styles of the model, and presents the reverse formulation of the forward model. Section 3 provides a detailed description of the forward and reverse computational steps taken by our parallel discrete event scheme. Our experimental setup and results from performance study are discussed in Section 4, along with a comparison of the results in this paper with those in related work. Finally, we conclude in Section 5 with a discussion of the future scope of our approach.

2. VECTOR-BASED FORMULATION

In this section, a vector update formulation of the TLM-based wave propagation problem is presented, with the goal of controlling the LP granularity, thereby reducing event overhead.



(a) (b)

Fig. 1. (a) Variables of the TLM model in a 2D example. (b) Sample voltage profile along the $z = 15$ plane at $t = 75$ s during a simulation with one voltage source at the center of a $30 \times 30 \times 30$ domain.

2.1 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 v_{ij} and v_{ji} are defined across each link in the grid that connects two neighboring nodes i and j in the directions $i \rightarrow j$ and $j \rightarrow i$, respectively. Fig. 1(a) shows a 2D example. 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 the term *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 [Nutaro et al. 2008], that model the propagation of radio signals in terms of the total and partial voltages defined above are:

$$v_{ij}(t+1) \;=\; R_{ij}\left(\frac{V_i(t)}{3} - v_{ij}(t)\right) + T_{ji}\left(\frac{V_j(t)}{3} - v_{ji}(t)\right) \tag{1}$$

$$V_i(t+1) \;=\; \sum_{k=0}^{5} v_{ij_k}(t+1) \tag{2}$$

where j_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 (4). A computation of the voltage profile (set of total voltages across all the n^3 nodes in the grid [see Fig. 1(b)]) at a time step t requires the availability of all the partial voltages at the previous time step.

2.2 Event-driven Execution

Temporal updates of the voltage profile can be either *time-driven* or *event-driven*. Time-driven approaches regularly update the set of partial and total voltages after the passage of each pre-defined time interval. The internal is determined by convergence requirements such as the aspect ratio of finite-difference schemes. In event-driven approaches, the state of a physical system changes at irregularly-spaced instants of time through instantaneous transitions. An *event* is associated with each such transition. For example, an event can be triggered every time the change in a node's voltage exceeds a specified amount of voltage differential. Discrete event formulation, therefore, delinks the necessity for a synchronous (time-stepped) execution from the evolution of the physical system. Instead, it views the same simulation as a set of time-stamped events (containing temporal information about the physical state variables) processed without violating global causality. When such event-based simulations are distributed across multiple processors, preservation of global causality becomes very challenging since state updates across processors are no longer guaranteed to be concurrent. Parallelization of such discrete event algorithms has been known to require causality control mechanisms that are highly challenging to scale well across a large number of processors [Fujimoto 1989; Perumalla 2006].

2.3 Vector Formulation: Sequential Forward Execution

Let the 3D Cartesian grid coordinates be mapped to a 1D array using the mapping function $f(a, b, c) = a + nb + n^2c$ where (a, b, c) are the 3D node coordinates. In this section, unless otherwise specified, the partial voltage on a link $i \rightarrow j$ will be denoted by v_{ij}^{α} where $\alpha \in [0, 5]$ corresponds to the direction specified by $i \rightarrow j$. Eqn (1) and Eqn (2) can now be rewritten as:

$$v_{ij}^{\alpha}(t+1) \;=\; R_{ij}^{\alpha}\left(\frac{V_i(t)}{3} - v_{ij}^{\alpha}(t)\right) + T_{ji}^{-\alpha}\left(\frac{V_j(t)}{3} - v_{ji}^{-\alpha}(t)\right) \tag{3}$$

$$V_i(t+1) \;=\; \sum_{\alpha=0}^{5} v_{ij}^{\alpha}(t+1) \tag{4}$$

where we also use the notation $-\alpha$ to denote the direction opposite to α.

The TLM-based simulation proceeds as shown in Algorithm 1. Note that the total voltage in the TLM equations is an intermediate variable that can be eliminated by substituting Eqn (4) in Eqn (3). The resulting set of equations can be cast into the form of a matrix-vector update:

$$X(t+1) = A \cdot X(t)$$

where the matrix A, called the *connectivity matrix*, has a linear number of non-zero elements (reflection and transmission coefficients). A simulation of the signal propagation proceeds by updating the vector X of partial voltages at each time

Algorithm 1 A Time-stepped TLM Simulation

1: $x_{ij}^{\alpha}(0) \leftarrow$ User-defined initial values $\forall\, i, j \in [0, \cdots, n-1]$ and $\forall\, \alpha \in [0, 1, \cdots, 5]$
2: $V_i(0) \leftarrow$ User-defined initial values $\forall\, i \in [0, \cdots, n-1]$
3: **for** $t = 0$ to $T-1$ **do**
4: Compute $x_{ij}^{\alpha}(t+1)\ \forall\, i, j \in [0, \cdots, n-1]$ and $\forall\, \alpha \in [0, 1, \cdots, 5]$ {Eqn (3)}
5: Compute $V_i(t+1)\ \forall\, i \in [0, \cdots, n-1]$ {Eqn (4)}
6: **end for**

step through an $O(N)$ matrix-vector multiplication. Note that the structure of the connectivity matrix A depends on the structure of the vector X.

2.4 Vector Execution Reversal Challenge

In principle, a forward simulation defined by the preceding matrix-vector product can be reversed by computing the inverse matrix-vector multiplication:

$$X(t) = A^{-1} \cdot X(t+1)$$

But, care needs to be exercised because a naive construction of the connectivity matrix could result in: **(a)** cubic work for the matrix inversion (though computed only once) and **(b)** quadratic work for each reversal since the inverse of a sparse matrix need not necessarily be sparse. In fact, when X is constructed by concatenating six smaller vectors, X^0, X^1, X^2, X^3, X^4 and X^5, each of length n^3 and containing all the partial voltages along the directions indicated by the subscript, the inverse of the resulting connectivity matrix can be shown to be dense, thus requiring quadratic work for each reversal, even though the forward matrix-vector multiplication is linear. Each application of such an inverse matrix-vector multiplication would require $O\left(n^2\right)$ work, in turn degrading the performance of optimistic execution based on reverse computing. In addition, the cost of computing the inverse matrix, though carried out only once, would be $O\left(n^3\right)$.

In summary, using a vector update representation, the TLM based simulation can be shown to be reversible, although the complexity of each reverse computation can potentially become greater than that of the forward computation by one or even two orders of magnitude.

2.5 Vector Formulation: Sequential Reverse Execution

A naive approach such as the one outlined previously results in a connectivity matrix whose inverse is dense, making the reverse computation cost proportional to the cube of the problem size, which is unacceptably high. Here we develop an alternative matrix-vector representation whose inverse computation is linear.

Let us define the variable:

$$y_{ij}^{\alpha}(t) = \left[\frac{V_i(t)}{3} - x_{ij}^{\alpha}(t)\right]$$

Eqn (3) can then be rewritten as:

$$x_{ij}^{\alpha}(t+1) = R_{ij}^{\alpha} y_{ij}^{\alpha}(t) + T_{ji}^{-\alpha} y_{ji}^{-\alpha}(t)$$

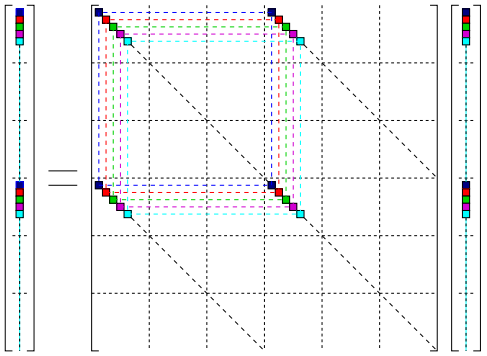


Fig. 2. Structure of the connectivity matrix M.

For ease of presentation, we adopt the following notation:

$$\overline{x}_{ij}^{\alpha} = x_{ij}^{-\alpha} \ , \ \overline{y}_{ij}^{\alpha} = y_{ij}^{-\alpha}$$

$$Y_0(t) = \left[y^0_{0,j_0}(t) \ y^0_{1,j_1}(t) \ \cdots \ y^0_{m-2,j_{m-2}}(t) \ y^0_{m-1,j_{m-1}}(t) \right]$$

$$Y_1(t) = \left[y^1_{0,j_0}(t) \ y^1_{1,j_1}(t) \ \cdots \ y^1_{m-2,j_{m-2}}(t) \ y^1_{m-1,j_{m-1}}(t) \right]$$

$$Y_2(t) = \left[y^2_{0,j_0}(t) \ y^2_{1,j_1}(t) \ \cdots \ y^2_{m-2,j_{m-2}}(t) \ y^2_{m-1,j_{m-1}}(t) \right]$$

$$\overline{Y}_0(t) = \left[\overline{y}^0_{0,j_0}(t) \ \overline{y}^0_{1,j_1}(t) \ \cdots \ \overline{y}^0_{m-2,j_{m-2}}(t) \ \overline{y}^0_{m-1,j_{m-1}}(t) \right]$$

$$\overline{Y}_1(t) = \left[\overline{y}^1_{0,j_0}(t) \ \overline{y}^1_{1,j_1}(t) \ \cdots \ \overline{y}^1_{m-2,j_{m-2}}(t) \ \overline{y}^1_{m-1,j_{m-1}}(t) \right]$$

$$\overline{Y}_2(t) = \left[\overline{y}^2_{0,j_0}(t) \ \overline{y}^2_{1,j_1}(t) \ \cdots \ \overline{y}^2_{m-2,j_{m-2}}(t) \ \overline{y}^2_{m-1,j_{m-1}}(t) \right]$$

Consider the vector $Y(t)$ defined by:

$$Y(t) = \left[Y_0(t) \ Y_1(t) \ Y_2(t) \ \overline{Y}_0(t) \ \overline{Y}_1(t) \ \overline{Y}_2(t) \right]^T$$

which is obtained by stacking $Y_k(t)$ and $\overline{Y}_k(t)$, $k = 0, 1, 2$. In terms of the preceding definitions, Eqn (3) can be written as:

$$X(t+1) = M \cdot Y(t)$$

where the transformation matrix M has the structure shown in Fig. 2. Note that the non-zero elements of M constitute $3n$ independent 2×2 matrices, each of the form:

$$M_{i,j} = \begin{bmatrix} R_{ij} & T_{ji} \\ T_{ij} & R_{ji} \end{bmatrix}$$

and defined for each nearest neighbor pair (i, j). Recall that a 2×2 matrix A and its inverse A^{-1} are related as follows:

$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \Longleftrightarrow A^{-1} = \frac{1}{(a_{00}a_{11} - a_{01}a_{10})} \begin{bmatrix} a_{11} & -a_{01} \\ -a_{10} & a_{00} \end{bmatrix}$$

Based on this, M^{-1} can be constructed by simply multiplying each off-diagonal element of M by -1 and exchanging the elements $M_{i,i} \leftrightarrow M_{i,i+3n}$ of M for all $0 \leq i < 3n$, in both cases multiplying each modified element by the corresponding determinant $1/(m_{i,i}m_{i+3n,i+3n} - m_{i,i+3n}m_{i+3n,i})$. As such, the structure of M^{-1} is the same as that of M. It follows that M^{-1} can be computed in $\Theta(n)$ time. Each product of the vector with the inverse matrix necessary to carry out a reversal can, therefore, be accomplished in $O(N)$ time.

Note that this is vastly superior to the straightforward method of simple spatial decomposition of X and its corresponding M.

2.6 Vector Formulation Reversal Analysis

For the remainder of this paper, we will suppress the superscript notation for direction to the extent possible with the understanding that (i, j) is a pair of nearest neighbor implying that j is i's nearest neighbor in some direction denoted by α and i is j's nearest neighbor in the opposite direction $-\alpha$.

In the preceding TLM equations, the reflection and transmission coefficients for any nearest neighbor pair (i, j) are related as follows:

$$R_{ij} = -R_{ji} \qquad \text{and} \qquad R_{ij} + T_{ji} = 1$$

implying that

$$|M_{i,j}| = R_{ij}R_{ji} - T_{ji}T_{ij} = R_{ij}R_{ji} - [(1 - R_{ij})(1 - R_{ji})] = -1$$

Therefore:

$$M_{i,j}^{-1} = \frac{1}{|M_{i,j}|} \begin{bmatrix} R_{ji} & -T_{ji} \\ -T_{ij} & R_{ij} \end{bmatrix} = \begin{bmatrix} -R_{ji} & T_{ji} \\ T_{ij} & -R_{ij} \end{bmatrix} = \begin{bmatrix} R_{ij} & T_{ji} \\ T_{ij} & R_{ji} \end{bmatrix} = M_{i,j}$$

for every nearest neighbor pair (i, j) which in turn implies that M is its own inverse.

A closer look at the forward computation reveals that it can be decomposed into three stages denoted by the following three operations:

- operation $\oplus$: compute $V_i(t+1) = \sum_{\alpha=0}^{5} x_{ij}^{\alpha}(t+1)$
- operation $\ominus$: compute $y_{ij}^{\alpha}(t) = \left[\frac{V_i(t)}{3} - x_{ij}^{\alpha}(t)\right]$.
- operation M : compute $X(t+1) = M \cdot Y(t)$

Pictorially, this can be shown as:

$$\left.\begin{matrix} X(0) \\ V(0) \end{matrix}\right\} \xrightarrow{\ominus} Y(0) \xrightarrow{M} \Big\{X(1) \xrightarrow{\oplus} V(1) \xrightarrow{\ominus} Y(1)\Big\} \xrightarrow{M} \Big\{X(2) \xrightarrow{\oplus} V(2) \xrightarrow{\ominus} Y(2)\Big\} \cdots$$

The reverse code should therefore achieve the following:

$$\cdots \xleftarrow{\ominus^{-1}} Y(t-2) \xleftarrow{M^{-1}} \Big\{X(t-1) \xleftarrow{\oplus^{-1}} V(t-1) \xleftarrow{\ominus^{-1}} Y(t-1)\Big\} \xleftarrow{M^{-1}} X(t) \xleftarrow{\oplus^{-1}} \cdots$$

The input to the reverse procedure is therefore $X(t)$. Earlier, we showed that the inverse operation M^{-1} can be performed in linear time. Consider a pair (i, j) of neighboring points. The corresponding x and y variables are related in the following manner:

$$\begin{bmatrix} x_{ij}(t) \\ x_{ji}(t) \end{bmatrix} = \begin{bmatrix} R_{ij} & T_{ji} \\ T_{ij} & R_{ji} \end{bmatrix} \begin{bmatrix} y_{ij}(t-1) \\ y_{ji}(t-1) \end{bmatrix}$$

1. $V_i(t-1)$	$\leftarrow$	$\sum_{j=0}^{5} [R_{ij}x_{ij}(t) + T_{ji}x_{ji}(t)]$
2. $C_{ij}(t-1)$	$\leftarrow$	$[R_{ij}V_i(t-1) + T_{ji}V_j(t-1)]/3 - x_{ij}(t)$
3. $C_{ji}(t-1)$	$\leftarrow$	$[R_{ji}V_j(t-1) + T_{ij}V_i(t-1)]/3 - x_{ji}(t)$
4. $x_{ij}(t-1)$	$\leftarrow$	$R_{ij}C_{ij}(t-1) + T_{ji}C_{ji}(t-1)$
5. $x_{ji}(t-1)$	$\leftarrow$	$T_{ij}C_{ij}(t-1) + R_{ji}C_{ji}(t-1)$

Table I. Reversal steps in a nutshell.

Since $M = M^{-1}$, we have:

$$\begin{bmatrix} y_{ij}(t-1) \\ y_{ji}(t-1) \end{bmatrix} = \begin{bmatrix} R_{ij} & T_{ji} \\ T_{ij} & R_{ji} \end{bmatrix} \begin{bmatrix} x_{ij}(t) \\ x_{ji}(t) \end{bmatrix}$$

From Eqn (5), summing up over all the neighbors of point i, we get:

$$V_i(t-1) = \sum_k y_{ik}(t-1) = \sum_k [R_{ik}x_{ik}(t) + T_{ki}x_{ki}(t)]$$

which constitutes the operation $\ominus^{-1}$. At this point, $V_i(t-1), V_j(t-1), x_{ij}(t)$ and $x_{ji}(t)$ are known. The remaining unknowns, $x_{ij}(t-1)$ and $x_{ji}(t-1)$, are computed by solving the following pair of simultaneous equations for $x_{ij}(t)$ and $x_{ji}(t)$.

$$\begin{aligned} x_{ij}(t) &= R_{ij}\left[\frac{V_i(t-1)}{3} - x_{ij}(t-1)\right] + T_{ji}\left[\frac{V_j(t-1)}{3} - x_{ji}(t-1)\right] \\ x_{ji}(t) &= T_{ij}\left[\frac{V_i(t-1)}{3} - x_{ij}(t-1)\right] + R_{ji}\left[\frac{V_j(t-1)}{3} - x_{ji}(t-1)\right] \end{aligned}$$

This yields:

$$\begin{aligned} x_{ij}(t-1) &= R_{ij}C_{ij}(t-1) + T_{ji}C_{ji}(t-1) \\ x_{ji}(t-1) &= T_{ij}C_{ij}(t-1) + R_{ji}C_{ji}(t-1) \end{aligned}$$

where

$$\begin{aligned} C(t-1)_{ij} &= \frac{1}{3}\left(R_{ij}V_i(t-1) + T_{ji}V_j(t-1)\right) - x_{ij}(t) \\ C(t-1)_{ji} &= \frac{1}{3}\left(R_{ji}V_j(t-1) + iT_{ij}V_i(t-1)\right) - x_{ji}(t) \end{aligned}$$

The preceding computation represents the operation $\oplus^{-1}$. Thus, using the inverse operations M^{-1}, $\ominus^{-1}$ and $\oplus^{-1}$ in that order, the state variables with time-stamp $t-1$ can be recomputed from those with time-stamp t. Further, this can be accomplished in linear time. Table I summarizes the linear-time reversal of the state variables in a sequential algorithm.

Note that the preceding analysis was entirely based on the fundamental assumption of *sequential* computation. When data is distributed across multiple cores, additional care is necessary. This will be discussed in Section 3.5.

3. PARALLEL DISCRETE EVENT EXECUTION

3.1 Domain Decomposition

It is clear from Eqn (3) and Eqn (4), which will be referred 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 $P_a \times P_b \times P_c$ topology where a, b and c refer to the three geometric dimensions. Each processor is therefore responsible for $n^3/P_aP_bP_c = n^3/P$ 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 a, b and c 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. 3(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. 3(a)].
- V_L : set of all total voltages local to a processor [black circles in Fig. 3(a)].
- V_R : set of all remote total voltages required for the computation of all $x_{ij} \in X_L$ [gray circles in Fig. 3(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. 3(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, $O\left(\frac{N^{2/3}}{P^{2/3}}\right)$.

3.2 Logical Processes and Granularity

As previously mentioned, the traditional event rate performance measure of discrete event simulations is not appropriate in the context of the TLM model. This is because the number of events can differ depending on the granularity of an LP. The earliest attempt [Bauer and Page 2007] to parallelize the TLM-based model exhibited limited scalability, with self-relative parallel speed-up reported up to 25 processors. A subsequent attempt [Bauer et al. 2009] showed rapid performance degradation beyond 5,000 cores. In both, an LP is defined to be a grid cell implying that $O(1)$ partial voltages are mapped to each LP. In our formulation, we use the data partitioning strategy described above to map $O(N/P)$ partial voltages to an LP which in turn is mapped to a processor. The performance advantage of defining LPs with $O(N/P)$ granularity will be demonstrated and explained in Section 4.5.

3.3 Discrete Event Formulation

Each LP is evolved by processing two types of events defined on them, namely, *self-update events (sue)* and *threshold-cross events (tce)*. An example with two processors is shown in Fig. 3(b). Self-update events are processed at integral time-stamps t while threshold-cross events are processed at half-integer time-stamps

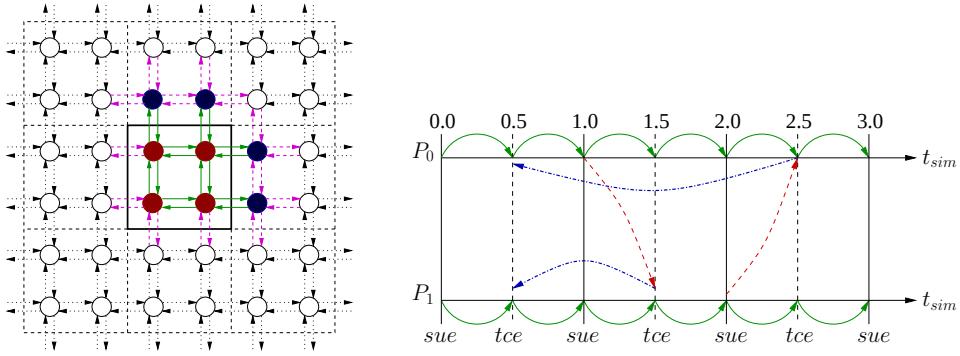


Fig. 3. (a) A 2D illustration of the unique set V_U (set of filled circles), the local set X_L (set of bold arrows) and remote set X_R (set of dashed arrows) for the processor whose computational domain is the solid square in the center. (b) Optimistic parallel discrete event processing in an example with two processors. Green (bold) arrows indicate forward execution, blue (dot-dashed) arrows reverse execution and red (dashed) arrows indicate time-stamped inter-processor messages.

$t + \frac{1}{2}$. When a self-update event is processed, each $x_{ij} \in X_L$ is updated though Eqn (3). Local updates that 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 is 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*, that is, 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 timetamp 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. 4 and Fig. 5). The preceding 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. 4 and Fig. 5, the arrays that are overwritten are indicated by the gray boxes pointed to by the write pointers.

3.4 Discrete Event Formulation: Parallel Forward Execution

Forward execution of our discrete event formulation in terms of self-update and threshold-cross events is shown in Fig. 4. 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 (4).
(3) COMPUTE-X_L^t : X_L^t is computed from the X_L^{t-1} and V_U^{t-1} using Eqn (3).
(4) SELECTIVE-SEND : To each processor that needs $x_{ij}^t \in X_L^t$, send x_{ij}^t iff $\mid x_{ij}^t - x_{ij}^{t-1} \mid \geq \delta$, 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. 4). 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 their values at that time. We discuss rollbacks in the next section.

3.5 Discrete Event Formulation: Parallel Reverse Execution

In our implementation, restoration of state upon rollback 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. 5:

(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. 5). 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.

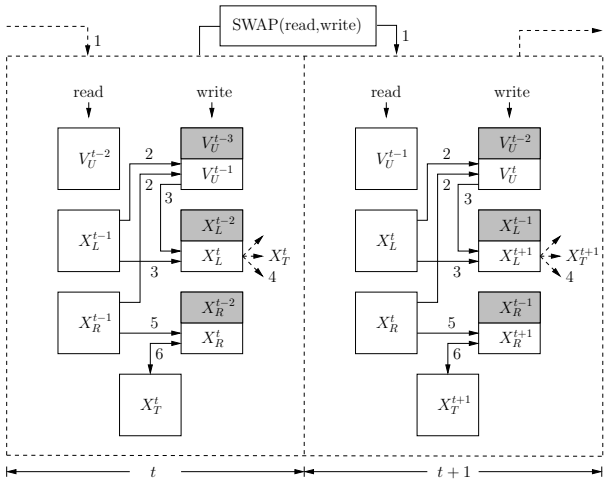


Fig. 4. Forward execution. Numbers on the arrows indicate the order in which the indicated steps are executed in a forward event or its reversal. The gray boxes indicate the arrays pointed to by the write pointer before they are overwritten by the arrays in the white boxes.

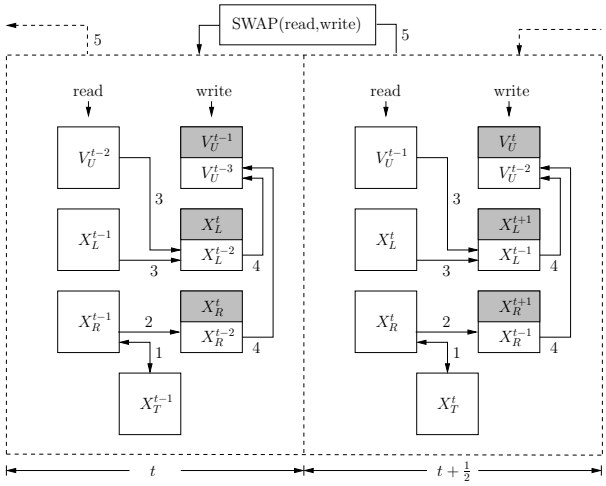


Fig. 5. Reverse execution. Numbers on the arrows indicate the order in which the indicated steps are executed in a forward event or its reversal. The gray boxes indicate the arrays pointed to by the write pointer before they are overwritten by the arrays in the white boxes.

(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 3.4 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 $x_{ij}^{t-1}, x_{ji}^{t-1} \in X_L^{t-1}$ as two unknowns and solve the following forward equations for $x_{ij}^t, x_{ji}^t \in X_L^t$:

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

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 result $M^{-1} = M$ derived in Section 2.5. Thus, the reversal equation to restore X_L^{t-1} using X_L^t and V_U^{t-1} is:

$$x_{ij}^{t-1} \leftarrow \left[\frac{V_i^{t-1}}{3} - R_{ij}x_{ij}^t - T_{ji}x_{ji}^t\right] \tag{5}$$

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:

$$V_i^{t-2} = \sum_k \left[R_{ik} x_{ik}^{t-1} + T_{ki} x_{ki}^{t-1} \right] \quad (6)$$

Note that, in the above equation, the partial voltages x_{ik}^{t-1}, $x_{ki}^{t-1} \in X_L^{t-1} \cup X_R^{t-1}$. 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. 5).

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.

4. PERFORMANCE STUDY

In this section, we study the performance of our implementation of the reversible parallel discrete event model. The study investigates the performance effects along four variables: (1) different threshold values for inter-LP updates, to vary inter-LP concurrency, (2) increasing problem/domain size, to observe modeling capacity, (3) different number of voltages mapped to each LP, to vary event computational granularity, and (4) increasing number of processors, to test scalability. The software and hardware platforms are described first, followed by descriptions of the variables and the values for each variable. The performance results are then presented and discussed.

4.1 Software Platform

We implemented the discrete event execution using the μsik engine [Perumalla 2005]. 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 an optimized time-stepped execution. 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.

4.2 Hardware Platform

Our hardware platform is a Cray machine with two partitions, namely, XT4 and XT5. In the XT4 partition, 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. In the XT5 partition, each compute node consists of two hex-core AMD Opteron processors with 16 GB of memory and a SeaStar 2+ router.

4.3 Performance Metric

As indicated earlier, 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. This will be discussed in greater detail in the next sections. 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. We use both *strong scaling speedup* in which the problem size is kept fixed as the number of processors is increased, as well as *weak scaling speedup* in which the problem size is increased proportionately with the increase in the number of processors.

4.4 Parameters

The following set of parameters were chosen in the performance study.

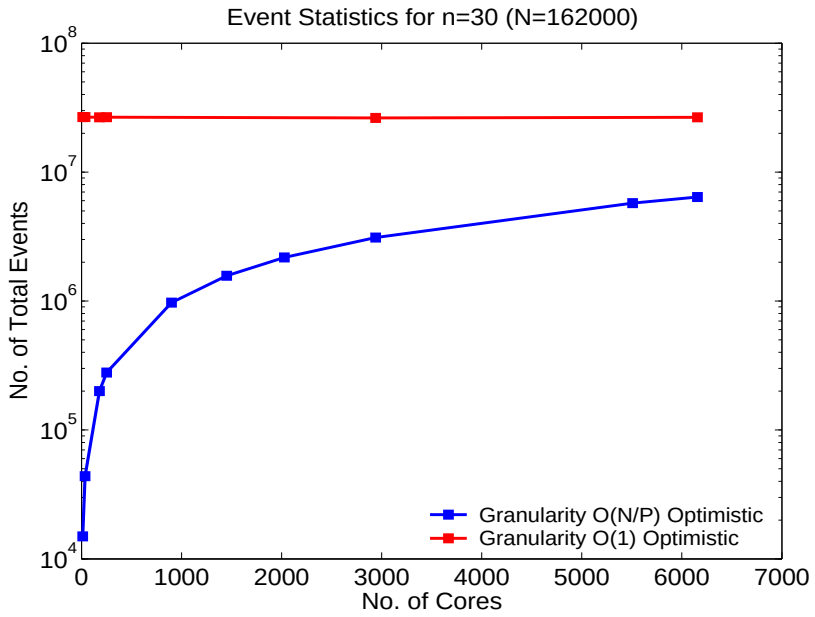
Granularity: We study the effects of event granularity by varying the number of rows mapped per LP. The two possible extremes are exercised, namely, one row per LP (fine-grained), and N/P rows per LP (coarse-grained). The fine-grained model is in fact the easiest to implement (and the most obvious parallelization scheme for [Nutaro et al. 2008], and used by [Bauer et al. 2009]), in sharp contrast to our vector formulation. The coarse-grained version is achieved as a generalization by our vector formulation.

Threshold Values: The effect of the threshold is implemented as a threshold-cross event is sent whenever $|V_{new} - V_{old}| \geq threshold$. The threshold values are chosen from -1, 0, 0.001, and 0.01. These control when the value any partial voltage is sent to the LP containing the peer of that partial voltage. A threshold of -1 sends the value after every update irrespective of the newly computed value. A threshold of 0 sends the value if and only if the value changes by any amount at all. A threshold of 0.001 sends even for the smallest non-zero update up to 0.001, while 0.01 tolerates changes up to ± 0.01 before an update is sent.

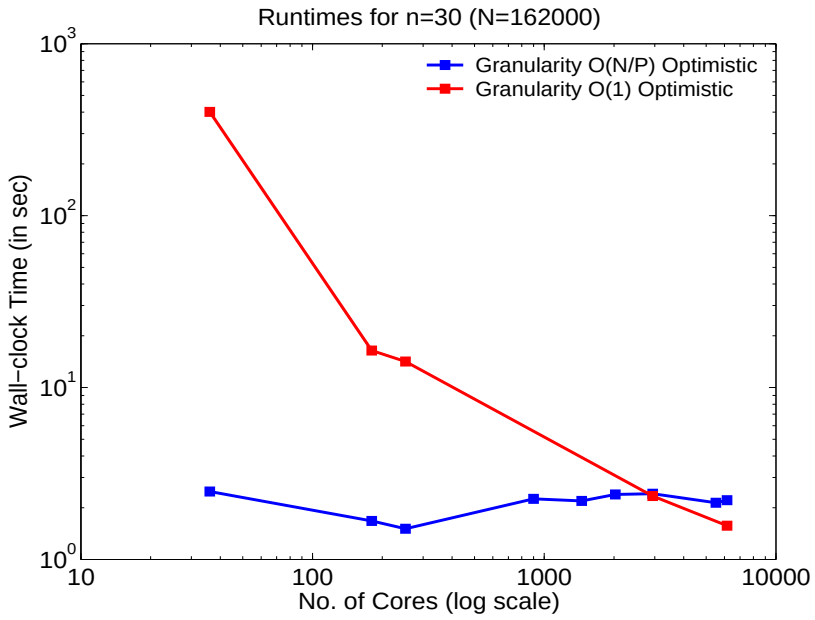
Problems Sizes and Number of Processors (Scalability): The number of processors is varied from small to large, choosing the configurations that satisfy the relation in Section 3 between the number of processors and its decomposition into the grid topology of the 3D problem domain. In the largest case, a 127,500 core configuration fits a process grid for $n = 324$ exactly. Strong-scaling experiments were performed with three grid sizes, corresponding to increasingly larger volumes that are typically encountered in wireless applications. The first is a medium-sized grid with $n = 30$ yielding 27,000 total voltages and 162,000 partial voltages corresponding to the volume of a large room or a hall. The second is a larger-sized grid with $n = 80$ that yields roughly half a million total voltages and 3 million partial voltages corresponding to a typical courtyard sized geometry. The third scenario with $n = 130$ yields roughly 2 million total voltages and 13 million partial voltages and roughly corresponds to a city block-sized volume. These grid sizes were tested with different threshold values, to exercise the performance effects of selective sends (asynchronous communications). Weak-scaling analysis was performed on a scenario with $n = 324$ (about 34 million total voltages and 200 million partial voltages corresponding to multiple city block-sized volumes) and 100 radio sources per core.

4.5 Performance Results

4.5.1 *Granularity.* In our approach, an LP consisting of $O(N/P)$ number of rows is mapped to a processor core (see Section 3.1). Each core is, therefore, equivalent to an LP. When the core count is increased for a fixed problem size (strong scaling), data locality per LP (core) suffers resulting in decreased concurrency which, in turn, generates more rollbacks. Consequently, the total number of events increase as data locality decreases at larger core counts. This trend is shown in Fig. 6(a) which plots the number of total events (sum of the numbers of committed and rollback events) against increasing core count for a small example with $n = 30$, $(N = 162,000)$. On the other hand, when an LP is defined to be a grid cell [Bauer et al. 2009], only $O(1)$ rows are mapped to each LP. In such an approach, the total number of rollback events increase dramatically because data locality per LP is highly fragmented despite data availability per core. Consequently, the total number of events is virtually insensitive to the core count, as demonstrated in Fig. 6(a). At large core counts (for the same problem size), granularity of LPs in both approaches converge as evinced by the similar trend in the event statistics at large core counts, as shown in Fig. 6(a). We believe that ultimately it is the parallel runtime that is of practical relevance for this application. When parallel runtime is used as the metric, the differing granularities of LPs have a profound effect on the parallel performance. This is shown in Fig. 6(b) for both conservative and optimistic executions. Since the granularities converge only as $P \rightarrow N$, the differences in the parallel runtime in the region of practical relevance ($P << N$) is very pronounced. Note that for larger $N(= 6n^3)$ than in Fig. 6(b), the convergence of the two approaches will be achieved at much higher core counts, thereby proportionately increasing the region in which our parallel approach based on an $O(N/P)$ granularity stands to deliver a significant advantage over one with $O(1)$ granularity.



(a)



(b)

Fig. 6. (a) Comparison of the number of total (committed + rollback) events based on two different definitions of logical processes. (b) Parallel runtimes based on two different definitions of logical processes.

4.5.2 *Threshold and Problem Size.* The results from varying the threshold values for different problem sizes are shown in Fig. 7. As described earlier, the communication pattern of our parallel execution is intimately dependent on the threshold voltage value. When threshold=-1, every state variable needs to be updated. Consequently every self-update event triggers a threshold-cross event resulting in a very synchronous communication pattern akin to a time-driven algorithm. As the threshold is increased, communication becomes increasingly asynchronous due to selective sends while the computation becomes more concurrent. This, in turn, improves the computation-to-communication ratio and better speedup gains are observed. For example, this trend is observed by comparing the speedup gained with increasing processors for threshold values of -1, 0, 0.001 and 0.01 in Fig. 7(a).

For relatively large problem sizes on small number of processors, computations remain highly concurrent resulting in a large computation-to-communication ratio. For such cases, the parallel runtimes remain largely unaffected by selective sends. This is evident from the near insensitivity of the speedups for all the scenarios on $P = 64$ in Fig. 7.

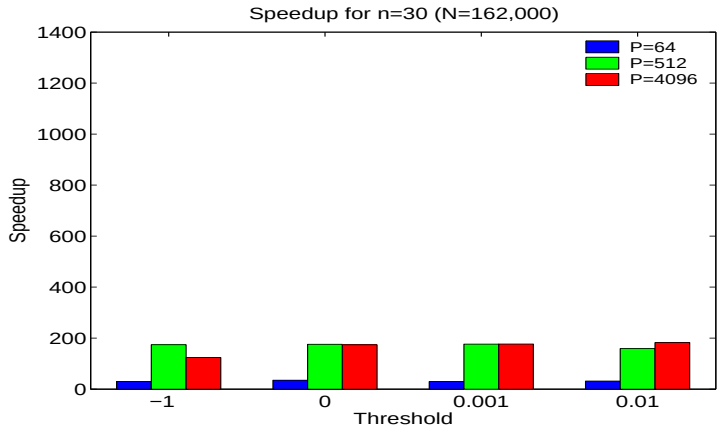
When threshold values become large, the wave propagates shorter grid cell distances. This can be understood by considering the asymptotic limit with threshold $= \infty$ for which the simulation ends after the very first self-update event (since the difference between the new and old values will always be less than ∞). This effect is seen in Fig. 7(b) and (c) when the threshold value changes from 0.001 to 0.01 for both $n = 80$ and $n = 130$. For coarser grids, such as for $n = 30$, diminishing speedups are expected to set in at larger threshold values, as will be discussed in the next subsection.

Overall, parallel speedups of over 1000× for large scale scenarios are observed on thousands of processors. Performance advantages of selective sends with increasing number of processors can be seen in Fig. 7 for all four threshold scenarios.

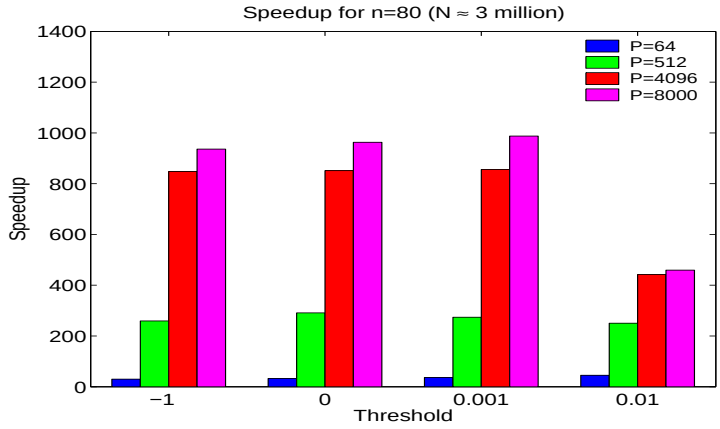
4.5.3 *Scalability.* To test scalability, we performed experiments with a very large problem size, configured the scenarios with a significant number of radio sources, and executed on increasingly larger parallel configurations, up to 127,500 processor cores. In order to exercise scenarios with sufficiently taxing radio activity, we configured the scenarios with a number radio sources evenly spaced in the domain mapped to each core. The performance metric of interest is the variation of the elapsed time with the number of processors, which represents weak scaling results.

Efficiency (defined canonically as $speedup/P$) of the algorithm is observed to be excellent 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 tested threshold values.

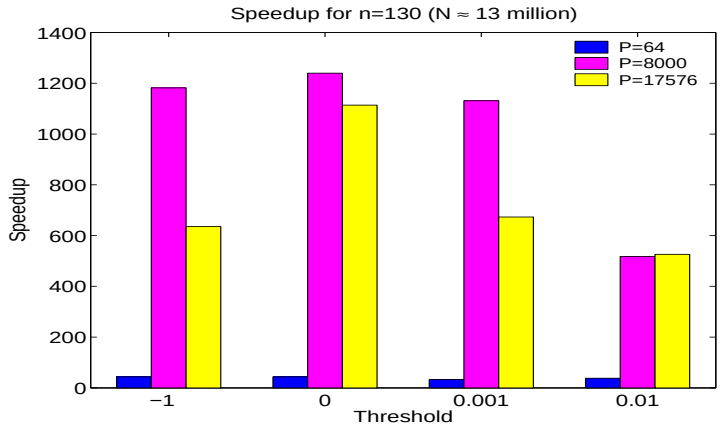
Fig. 8(a) shows weak scaling on a Cray XT5 platform for a problem size with $n = 324$ ($N \sim 200$ million). Though the size of the computational grid was kept constant, the number of radio sources was increased along with the core count at a rate of 100 sources per core. The speedup is observed to be nearly 100,000 executing on 127,500 cores. Fig. 8(b) shows the number of committed events and rolled back events. On smaller number of cores, the amount of computation per



(a)

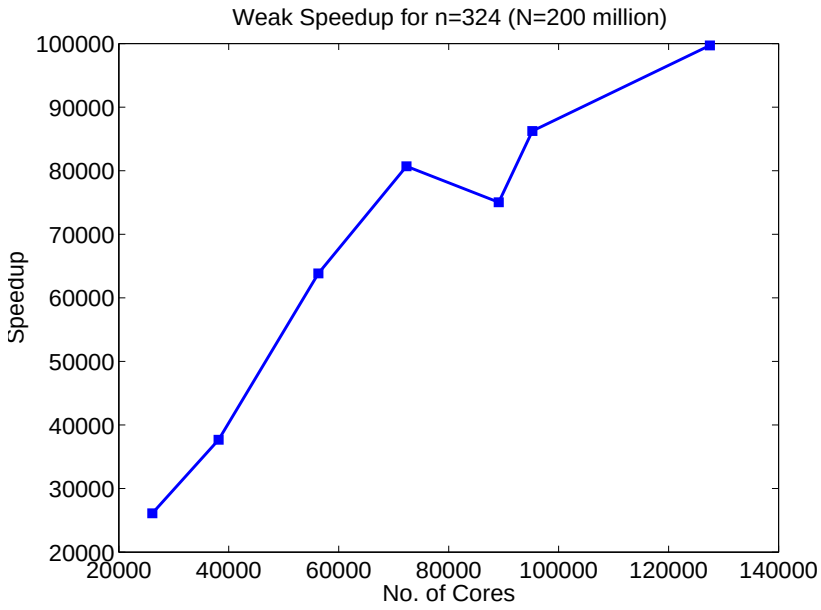


(b)

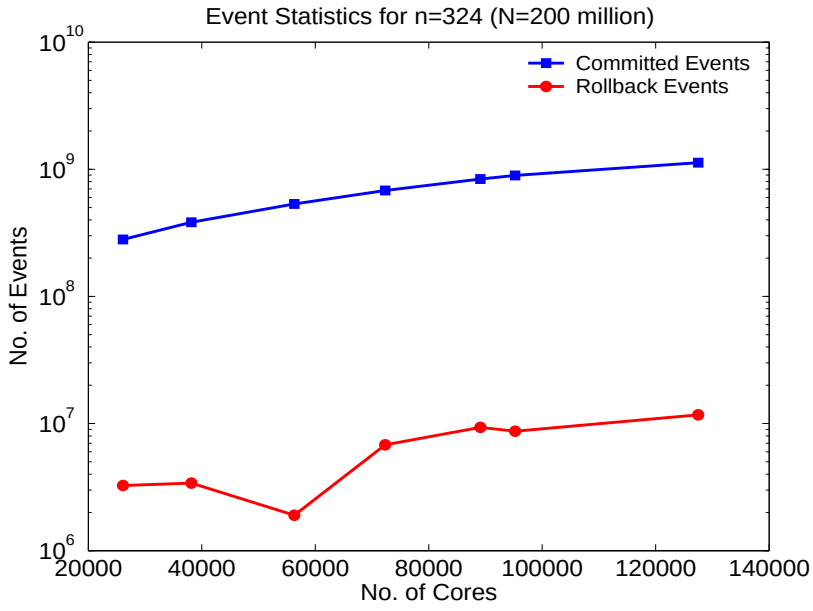


(c)

Fig. 7. Speedup (strong scaling) for problem sizes: (a) $n = 30$ (b) $n = 80$ and (c) $n = 130$ with varying threshold and number of processor cores on a Cray XT4.



(a)



(b)

Fig. 8. Scaling results on Cray XT5 with $n = 324$ and 100 radios per core: (a) weak scaling speedup (b) event statistics.

core was sufficiently large to keep all the cores heavily loaded, and hence nearly in synchrony, resulting in very few rollbacks. The number of rollbacks increased with larger number of processors, yet, overall, the percentage of rollbacks did not rise to alarming rates.

4.6 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.

5. CONCLUSIONS AND FUTURE WORK

An efficient parallelization and implementation of a recently-developed discrete-event based serial algorithm for the estimation of radio wave signal strength was presented. 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 has been used. The reversal equations that were subsequently used for rollbacks to restore the state of the system to a desired time in the past were explicitly derived.. The authors have demonstrated that such reverse computing based rollbacks can deliver unprecedented speedup for this problem. To the best of their knowledge, the results are also among the first to demonstrate 100,000 parallel speedup for any non-synthetic PDES application that is based on reverse computation. Also, such speedups for electromagnetic wave simulators have never been reported before. It has been shown 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 of object geometries contained in the volumes. The effect of granularity of logical processes on performance metrics such as event rates and parallel runtime was demonstrated and discussed.

An important issue that remains a subject of our on-going 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, barrier-based time-driven algorithms are prone to synchronization overheads. This point of comparison is particularly poignant in an era of peta-scale computing whose synchronization overheads can drastically deteriorate the performance of barrier-based time-driven codes.

6. 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.

The authors would also like to thank Vinod Tipparaju (ORNL) for many helpful suggestions.

REFERENCES

Bailey, M. L., Briner, J. V., and Chamberlain, R. D. 1994. Parallel logic simulation of VLSI systems. *ACM Computing Surveys 26,* 3, 255–294.

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

Bauer, D. W., Carothers, C. D., and Holder, A. 2009. Scalable Time Warp on Blue Gene Supercomputers. In *Proceedings of the Workshop on Principles of Advanced and Distributed Simulation*. 35–44.

Carothers, C., Perumalla, K. S., and Fujimoto, R. M. 1999. Efficient Optimistic Parallel Simulations using Reverse Computation. *ACM Transactions on Computer Modeling and Simulations 9,* 3, 224–253.

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

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

Choi, E. and Chung, M. J. 1995. An Important Factor for Optimistic Protocol on Distributed Systems: Granularity. In *Proc. of the Winter Simulation Conference*. 642–649.

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

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

Levy, M. 2000. *Parabolic Equation Methods for Electromagnetic Wave Propagation*. Institution of Engineering and Technology.

McBrayer, T. J. and Wilsey, P. A. 1995. Process Combination to Increase Event Granularity in Parallel Logic Simulation. In *Proc. of the Int'l. Parallel Processing Symposium*. 572.

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

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

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

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

Sadiku, M. N. O. 2000. *Numerical Techniques in Electromagnetics*. CRC Press.

Taflove, A. and Hagness, S. 2000. *Computational Electrodynamics: The Finite-Difference Time-Domain Method*. Artech.