OAK RIDGE
NATIONAL LABORATORY
MANAGED BY UT-BATTELLE
FOR THE DEPARTMENT OF ENERGY

ORNL/TM-2009/165

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

August 2009

Prepared by
Sudip K. Seal and Kalyan S. Perumalla

UT-BATTELLE
ORNL-27 (4-00)

ORNL/TM-2009/165

COMPUTATIONAL SCIENCES AND ENGINEERING DIVISION

SCALABLE PARALLEL EXECUTION OF AN EVENT-BASED RADIO SIGNAL PROPAGATION MODEL FOR CLUTTERED 3D TERRAINS

Sudip K. Seal
Kalyan S. Perumalla

Date Published: August 2009

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831-6283
Managed by
UT-BATTELLE, LLC
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

LIST OF FIGURES

Figure **Page**

ABSTRACT

Radio signal strength estimation 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) than 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×. The speed and scale achieved by our parallel execution enable larger scenarios and faster execution than has ever been reported before.

1.INTRODUCTION

A novel event-based model was recently proposed[1,2] 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.[3,4] Validation studies[2] 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.

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 very intensive computationally, 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.[2] 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.[6,7] The reader is referred to references such as refs. 1–2 for additional details behind the motivation.

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 or ray-tracing requires huge grid sizes to ensure numerical accuracies of the final solutions. This makes 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[1,2] to bridge the gap between low precision input data and accuracy considerations. In ref. 2 authors have shown empirical results that indicate the 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 ref. 2 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, 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[8] to parallelize the discrete event formulation of the TLM-based method described in only exhibited limited scalability, with self-relative parallel speedup reported up to 25 processors.

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 $i \rightarrow j$ 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 ref. 2, that govern the propagation of radio signals in terms of the total and partial voltages defined above are:

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

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

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 Eq. (2). A computation of the voltage profile [set of total voltages across all the n^3 nodes in the grid (Figs. 1–2)] at a time step t requires the availability of all the partial voltages at the previous time step.

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 view 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 these 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. Parallelization of these 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 refs. (9–10).

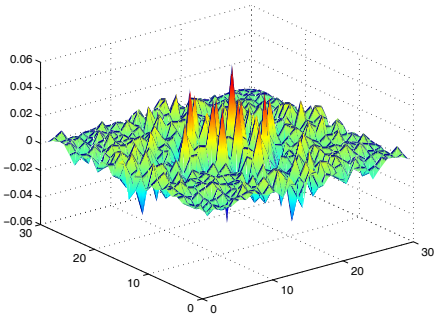

Fig. 1. Voltage profile along the $z = 15$ plane at $t = 50s$ during a simulation with one voltage source at the center of a $30 \times 30 \times 30$ domain.

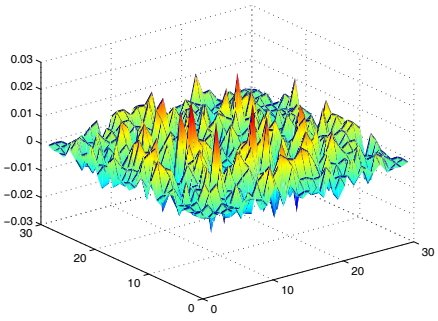

Fig. 2. Voltage profile along the $z = 15$ plane at $t = 75s$ during a simulation with one voltage source at the center of a $30 \times 30 \times 30$ domain.

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 reverse computing techniques to minimize the now well-understood overheads associated with traditional PDES approaches.[10,11] 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. 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 ref. 2, and realizing full 3D support under different parallel domain decomposition schemes.

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 report is organized as follows. Section 2 presents a vector update formulation of the TLM problem as described in Sect. 1. In Sect. 3, we derive the reversal equations explicitly to prove that each reversal can be carried out in linear time. Section 4 describes the parallel discrete event scheme underlying our algorithm. Our experimental setup and performance results are discussed in Sect. 5–7 followed by a discussion of the future scope of this work in Sect.8.

2. VECTOR UPDATE FORMULATION

Let the linear mapping of the 3D Cartesian coordinates to a 1D array be carried out using the mapping $f(i,j,k)=i+nj+n^2k$ where (i,j,k) are the 3D coordinates of a node. The simulation based on the TLM equations proceeds as follows:

Algorithm 1: TLM Simulation

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

It is evident from the above TLM equations that the total voltage is an intermediate variable that can be completely eliminated by substituting Eq. (2) in Eq. (1) (or vice-versa). The resulting set of equations is generally cast into a matrix-vector update form:

$$X^{t+1} = A \cdot X^t \quad , \tag{3}$$

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 step through a $O(N)$ matrix-vector multiplication.

In principle, the forward simulation defined by Eq. (3) can be reversed by computing the inverse matrix-vector multiplication:

$$X^t = A^{-1} \cdot X^{t+1} \quad . \tag{4}$$

Note that the structure of the connectivity matrix A depends on the structure of the vector X. As such, care needs to be exercised because a naïve formulation could result in: (a) cubic work for the matrix inversion computation (though computed only once) and (b) quadratic work for each inverse matrix-vector multiplication during rollback stemming from the fact that the inverse of a sparse matrix need not necessarily be sparse. In fact, when X is constructed by concatenating six smaller vectors, $X_{+x}, X_{+y}, X_{+z}, X_{-x}, X_{-y}$ and X_{-z}, 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. As a result, each application of the inverse matrix-vector multiplication would require $O(n^2)$ work, thereby degrading the performance of speculative execution using reverse computing. In addition, the cost of computing the inverse matrix, though carried out only once, is $O(n^3)$.

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

3. LINEAR TIME REVERSE COMPUTATION

Though a naïve approach such as the one outlined above results in a connectivity matrix whose inverse is dense, there is no *a priori* reason to assume there does not exist such a matrix-vector representation whose inverse is linear. For our purpose, it suffices to demonstrate that there exists *at least* one such case. For this purpose, let us define the variable:

$$y_{ij}^{\alpha}(t) = \left[\frac{V_i(t)}{3} - x_{ij}^{\alpha}(t) \right] \quad , \tag{5}$$

where the superscript α denotes one of the six possible directions along which the node j is a nearest neighbor of the node i. We use the notation $-\alpha$ to denote the direction opposite to α. Eq. (1) can now be rewritten as:

$$x_{ij}^{\alpha}(t+1) = R_{ij}^{\alpha} y_{ij}^{\alpha}(t) + T_{ji}^{-\alpha} y_{ji}^{-\alpha}(t) \quad . \tag{6}$$

Let denote the concatenation operation. For further ease of presentation, we adopt the following notations:

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

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

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

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

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

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

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

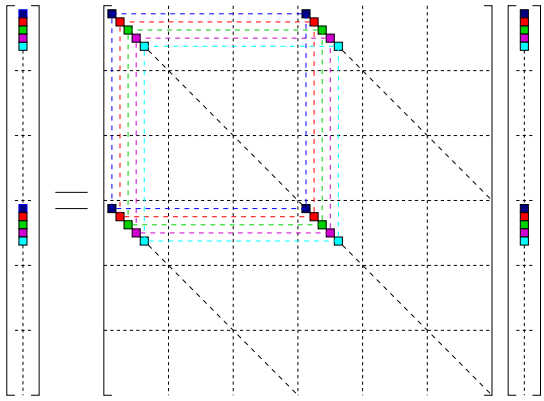

Fig. 3. Structure of the connectivity matrix, A.

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

$$Y(t)=\left[Y_0(t)\odot Y_1(t)\odot Y_2(t)\odot \overline{Y}_0(t)\odot \overline{Y}_1(t)\odot \overline{Y}_2(t)\right]^T . \tag{7}$$

In terms of the above definitions, Eq. (1) can be written as a vector update problem of the form represented by Eq. (3) but with X replaced by Y and the connectivity matrix A taking the form as shown in. Fig. 3. In Fig. 3, the like colored elements participate in a single partial voltage computation. Note that A is composed of $3n$ independent 2×2 matrices, each of the form:

$$A^{\alpha}_{i,j}=\begin{bmatrix} R^{\alpha}_{ij} & T^{-\alpha}_{ji} \\ T^{\alpha}_{ij} & R^{-\alpha}_{ji}\end{bmatrix}=\begin{bmatrix} a_{i,i} & a_{i,i+3m} \\ a_{i+3m,i} & a_{i+3m,i+3m}\end{bmatrix} , \tag{8}$$

and defined for each nearest neighbor pair (i,j) in a direction α with respect to i. Note that any 2×2 matrix A and its inverse A^{-1} are related through the following:

$$A=\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11}\end{bmatrix} \Leftrightarrow A^{-1}=\frac{1}{(a_{00}a_{11}-a_{01}a_{10})}\begin{bmatrix} a_{11} & -a_{01} \\ -a_{10} & a_{00}\end{bmatrix} . \tag{9}$$

Based on this, A^{-1} can be constructed by simply multiplying each off-diagonal element of A by -1 and exchanging the elements $A_{i,i} \leftrightarrow A_{i,i+3n}$ of A for all $0 \le i < 3m$, in both cases remembering to multiply each modified element by the corresponding determinant $1/(a_{i,i}a_{i+3m,i+3m} - a_{i,i+3m}a_{i+3m,i})$. Clearly, A^{-1} can be computed in $\Theta(N)$ time. In addition, the structures of the matrix and its inverse are identical implying that the inverse matrix vector multiplication can also be accomplished in linear time.

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

$$R_{ij}^{\alpha} = -R_{ji}^{-\alpha} \qquad \text{and} \qquad R_{ij}^{\alpha} + T_{ji}^{-\alpha} = 1 \quad . \tag{10}$$

As such, the determinant of the corresponding $A_{i,j}$ matrix is:

$$\left|A_{i,j}\right| = R_{ij}R_{ji} - T_{ji}T_{ij} = R_{ij}R_{ji} - \left[(1-R_{ij})(1-R_{ji})\right] = -1 + R_{ij} + R_{ji} = -1 \quad . \tag{11}$$

As above, from here on we will suppress the superscript with the understanding that for each neighbor pair (i, j), the subscript ij is associated with the direction α w.r.t point i and the subscript ji is associated with the direction $-\alpha$ w.r.t point j. Therefore:

$$A_{i,j}^{-1} = \frac{1}{|M_{i,j}|}\begin{bmatrix} R_{ji} & -T_{ji} \\ -T_{ij} & R_{ij} \end{bmatrix} = \begin{bmatrix} -R_{ji} & T_{ji} \\ T_{ij} & -R_{ij} \end{bmatrix} = \begin{bmatrix} R_{ij} & T_{ji} \\ T_{ij} & R_{ji} \end{bmatrix} = A_{i,j} \tag{12}$$

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 $V_i(t+1) = \sum_j x_{ij}(t+1) \;\; \forall\, i$.
- operation $\ominus$: compute $y_{ij}(t) = \left[\frac{V_i(t)}{3} - x_{ij}(t)\right] \;\; \forall\, i, j$.
- operation M : compute $X(t+1) = M \cdot Y(t)$.

Pictorially, this can be shown as:

$$X(0), V(0) \xrightarrow{\ominus} Y(0) \xrightarrow{B} \left\{ X(1) \xrightarrow{\oplus} V(1) \xrightarrow{\ominus} Y(1) \right\} \xrightarrow{B} \left\{ X(2) \xrightarrow{\oplus} V(2) \xrightarrow{\ominus} Y(2) \right\} \xrightarrow{B} \cdots$$

The reverse code should therefore achieve the following:

$$\cdots \xleftarrow{\ominus^{-1}} Y(t-2) \xleftarrow{B^{-1}} \left\{ X(t-1) \xleftarrow{\oplus^{-1}} V(t-1) \xleftarrow{\ominus^{-1}} Y(t-1) \right\} \xleftarrow{B^{-1}} X(t) \xleftarrow{\oplus^{-1}} \cdots$$

The input to the reverse procedure is therefore $X(t)$. In the above we showed that the inverse operation A^{-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:

$$\begin{bmatrix} x_{ij}(t) \\ x_{ji}(t) \end{bmatrix} = \begin{bmatrix} R_{ij} & T_{ji} \\ T_{ij} & R_{ji} \end{bmatrix} \begin{bmatrix} y_{ij}(t-1) \\ y_{ji}(t-1) \end{bmatrix} \quad . \tag{13}$$

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

$$\begin{bmatrix} y_{ij}(t-1) \\ y_{ji}(t-1) \end{bmatrix} = \begin{bmatrix} R_{ij} & T_{ji} \\ T_{ij} & R_{ji} \end{bmatrix} \begin{bmatrix} x_{ij}(t) \\ x_{ji}(t) \end{bmatrix} \tag{14}$$

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

$$V_i(t-1) = \sum_k y_{ik}(t-1) = \sum_k \left[R_{ik} x_{ik}(t) + T_{ki} x_{ki}(t) \right] \tag{15}$$

which constitutes the operation $\ominus^{-1}$. Equation (13) reads as:

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

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

At this point, $x_{ij}(t), x_{ji}(t), V_i(t-1)$ and $V_j(t-1)$ are known. The only unknowns are $x_{ij}(t-1)$ and $x_{ji}(t-1)$ which can be computed by solving the above pair of simultaneous equations. For convenience, we first compute two constants:

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

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

In terms of these constants, we obtain:

$$x_{ij}(t-1) = R_{ij} C_{ij}(t-1) + T_{ji} C_{ji}(t-1) , \tag{16}$$

$$x_{ji}(t-1) = T_{ij} C_{ij}(t-1) + R_{ji} C_{ji}(t-1) . \tag{17}$$

The above two equations constitute the operation $\oplus^{-1}$. Thus, using the inverse operations A^{-1}, $\ominus^{-1}$ and $\oplus^{-1}$ in that order, it can be easily seen that the cost of reverse computing from $X(t)$ to $X(t-1)$ takes linear time.

In this section, we have explicitly derived the reversal equations and shown that the time to restore a state at a time step *t* from a time step *t*+1 takes a linear amount of work. However, this work is serial. When data is distributed across processors, special care needs to be taken to ensure that relevant data needed for the above reversal to work is not irrecoverably lost due to destructive assignments. The following table summarizes the reversal steps in a nutshell.

Reversal Steps in a Nutshell

1. $V_i(t-1) \leftarrow \sum_j \left[R_{ij} x_{ij}(t) + T_{ji} x_{ji}(t) \right]$
2. $C_{ij}(t-1) \leftarrow \frac{1}{3}\left(R_{ij} V_i(t-1) + T_{ji} V_j(t-1) \right) - x_{ij}(t)$
3. $C_{ji}(t-1) \leftarrow \frac{1}{3}\left(R_{ji} V_j(t-1) + T_{ij} V_i(t-1) \right) - x_{ji}(t)$
4. $x_{ij}(t-1) \leftarrow R_{ij} C_{ij}(t-1) + T_{ji} C_{ji}(t-1)$
5. $x_{ji}(t-1) \leftarrow T_{ij} C_{ij}(t-1) + R_{ji} C_{ji}(t-1)$

4. PARALLEL DISCRETE EVENT SCHEME

Domain Decomposition: It is clear from Eq. (1–2), 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 $n^3 / P_x P_y P_z = n^3 / P$ 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.

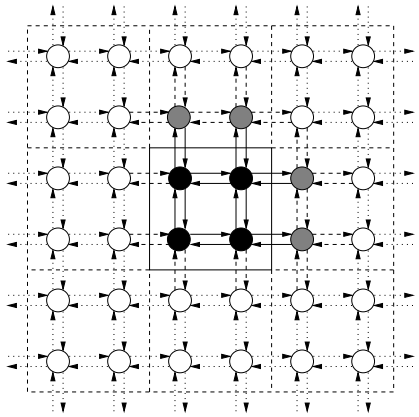

Fig. 4. A 2D illustration of the unique set V_U (set of filled circles), the local set X_L (set of bold arrows) and remote set X_R (set of dashed arrows) for the processor responsible.

For ease of presentation, we use a 2D example in Fig. 4 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. 1).
- V_L : set of all total voltages local to a processor (black circles in Fig. 4).
- V_R : set of all remote total voltages required for the computation of all $x_{ij} \in X_L$ (gray circles in Fig. 4).
- 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. 4).

The preceding parallel domain decomposition guarantees that

(a) for each local partial voltage $x_{ij} \in X_L$, the reverse partial voltage x_{ji} is also local.
(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, $O\left(N^{2/3} / P^{2/3}\right)$.

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. 5. Self-update events are processed at integral time-stamps t while threshold-cross events are processed at half-integer time-stamps $t+1/2$. When a self-update event is processed, each $x_{ij} \in X_L$ is updated though Eq. (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 time-stamped with $t+1/2$. 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 these 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 .

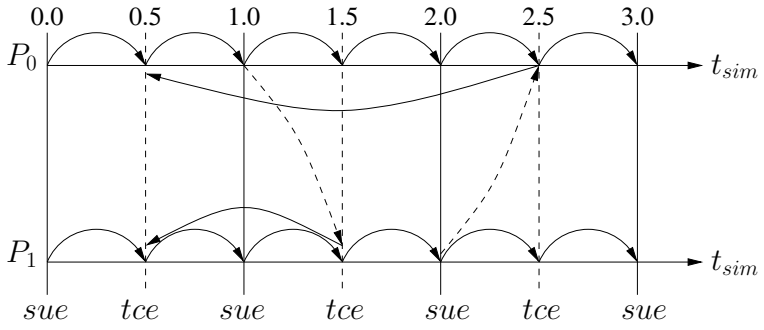

Fig. 5. Optimistic parallel discrete event processing in an example with two processors.

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. 6). 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. 6, the arrays that are overwritten are indicated by the gray boxes pointed to by the write pointers.

4.1 FORWARD EXECUTION

Forward execution of our discrete event approach in terms of self-update and threshold-cross events are shown in Fig. 6. 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 Eq. (2).
3. **COMPUTE-** X_L^t : X_L^t is computed from the X_L^{t-1} and V_U^{t-1} using Eq. (1).
4. **SELECTIVE-SEND** : To each processor that needs $x_{ij}^t \in X_L^t$, send x_{ij}^t if and only if $|x_{ij}^t - x_{ij}^{t-1}| \geq \delta$, 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 (Fig. 6). 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.

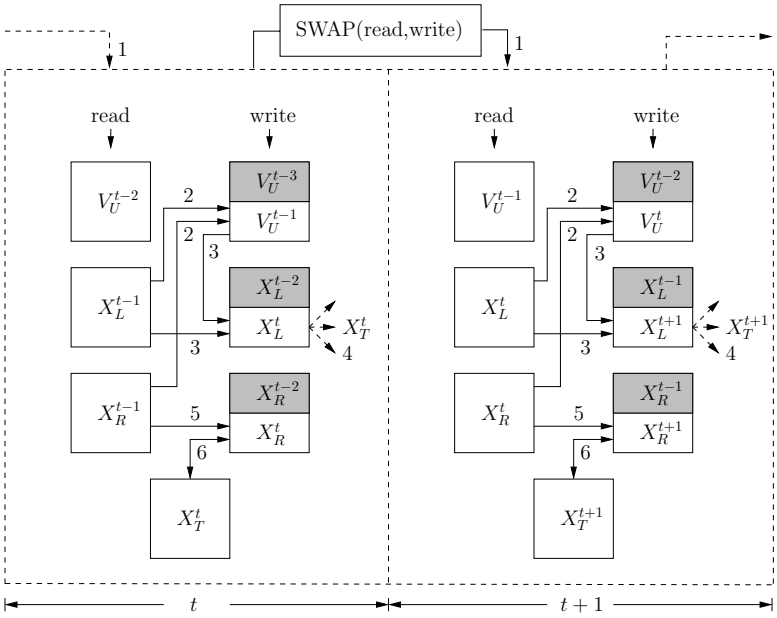

Fig. 6. Forward execution. Numbers on the arrows indicate the order in which the indicated steps are executed in a forward event o. The gray boxes indicate the arrays pointed to by the write pointer before they are overwritten by the arrays in the white boxes.

4.2 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 $t+1/2$, 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. 7:

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 (Fig. 7). 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 array X_R^t is copied to the array X_R^{t+1} pointed to by the write pointer. This reverses the operation in step 5 of Sect. 4.1 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

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

The above equation can be solved to yield:

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

Thus, the reversal equation to restore X_L^{t-1} using X_L^t and V_U^{t-1} is:

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

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:

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

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:

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

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

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

Thus, we have:

$$V_i^{t-2} = \sum_k \left[R_{ik} x_{ik}^{t-1} + T_{ki} x_{ki}^{t-1} \right] \quad . \tag{18}$$

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 (Fig. 7).

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

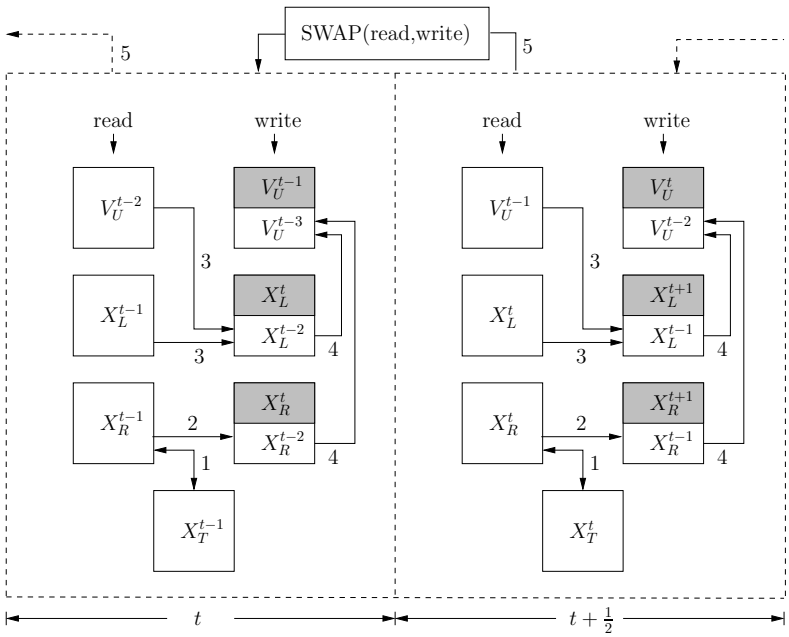

Fig. 7. Reverse execution. Numbers on the arrows indicate the order in which the indicated steps are executed during a reversal. The gray boxes indicate the arrays pointed to by the write pointer before they are overwritten by the arrays in the white boxes.

5. EXPERIMENTAL SETUP

Software Platform: We implemented the discrete event execution using the μsik engine.[12] μsik provides an application programming interface in the *C++* language that supports the concept of logical processes, events for exchanging time-stamped 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,[2,8] 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 time-stamped 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.

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.

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.

6. PERFORMANCE RESULTS

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 M partial voltages. The third, which is a very large scenario with $n = 130$, yields roughly 2 M total voltages and 13 M partial voltages. These grid sizes were tested with different threshold values, to exercise the performance effects of selective sends (asynchronous communications).

Speedup Analysis: Speed-up plots for the different scenarios are shown in Fig. 8. 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. 8 for all three scenarios. When the threshold becomes large, the wave propagates shorter grid cell distances.† This effect is seen from the

† When the asymptotic limit with threshold= ∞ in considered, 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

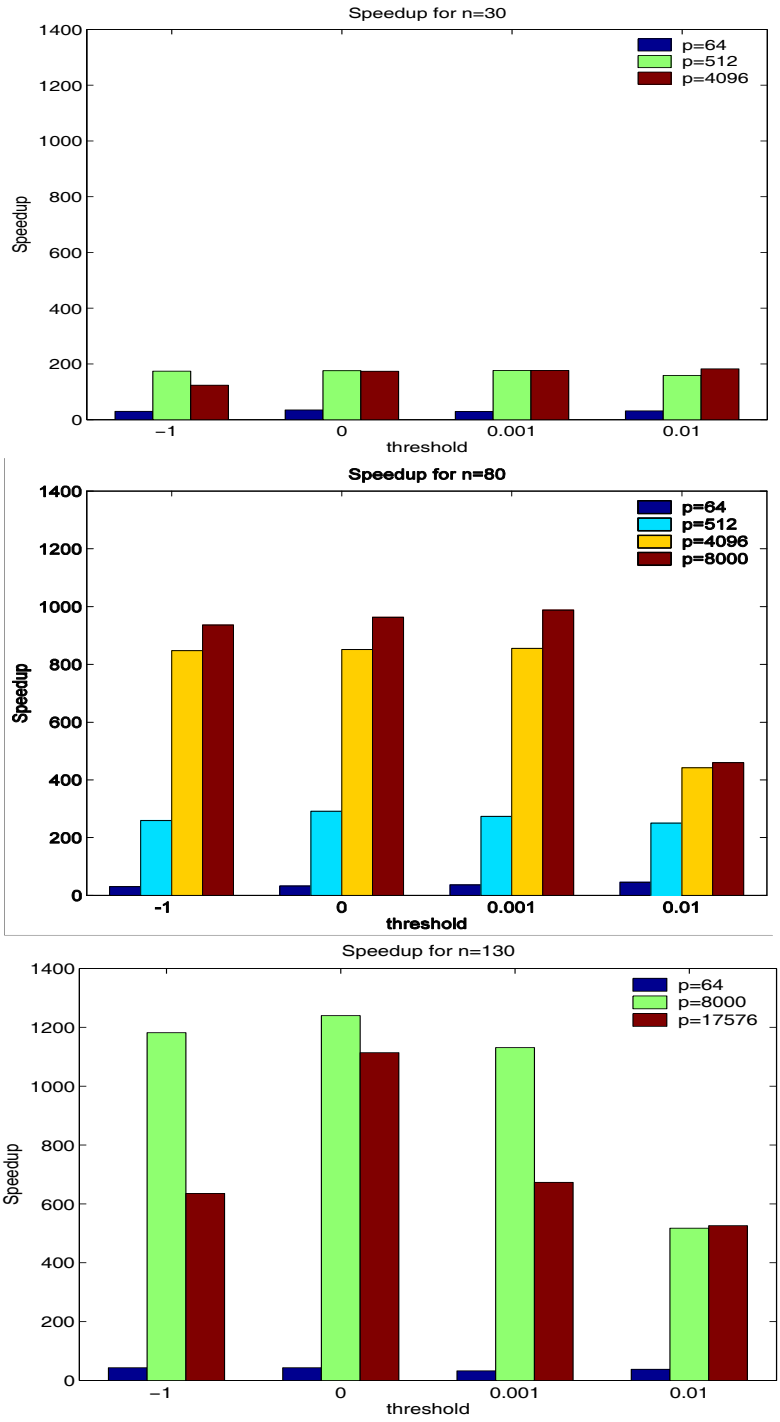

Fig. 8. Speedup for various problem sizes, n=30, 80, and 130, with varying threshold and number of processor cores.

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-

---

than ∞).

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.

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 h but only about 6 s on $p = 8000$ processors using the above algorithm. This is well within the scope of real time predictions of mobile wireless signal strength.

7. COMPARISON WITH TRADITIONAL SYNCHRONOUS EXECUTION

A time-stepped simulation can be efficiently executed using the following parallel algorithm.

Algorithm 2: Synchronous Simulation of Radio Wave Propagation

1: initialization and parallel decomposition
2: **for** $t = 0; t < \text{endtime}; t = t + \Delta t$ **do**
3: update X_L
4: synchronize
5: selectively send $x \in X_L$ needed in remote processors
6: receive $x \in X_R$
7: update V_U
8: **end for**

When the runtime t_p of parallel algorithms is viewed as a sum of computation cost t_{comp}, communication cost t_{comm} and synchronization cost t_{synch}, scaling of traditional time-stepped algorithms is adversely affected at large processor counts due to the amount of time needed for the algorithm to synchronize amongst increasing number of processors. This can be seen in Fig. 9 where the computation, communication and synchronizations costs have been presented for a problem of size $N \approx 3$ million against varying number of processors. It is clear that computation time dominates the total runtime for smaller number of processors. However, synchronization overhead continues to grow and finally exceeds both computation

and communication costs after around 1K processors. Beyond 1K processors, the synchronization overhead of the time-stepped approach accounts for almost 75% of the total runtime.

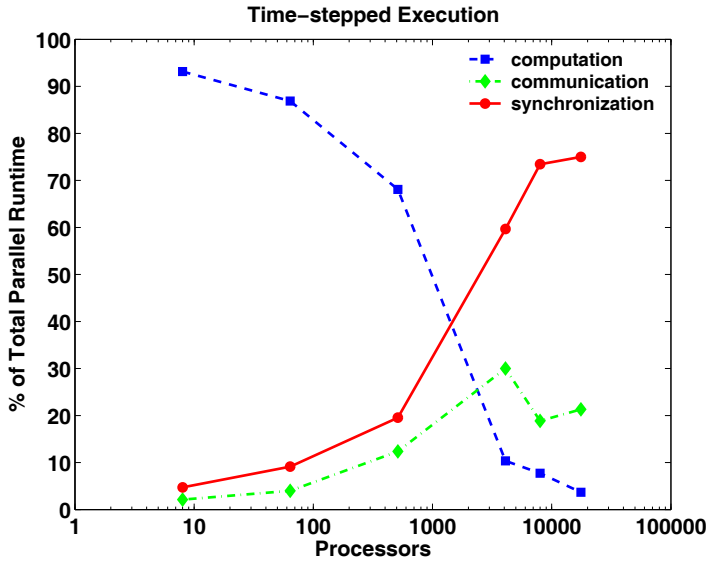

Fig. 9. Percentage of total runtime that is spent on computation, communication and synchronization on *P* = 8, 64, 512, 4096, 8000, and 17,576 with $N \approx 3$ M.

Fig. 10 compares the speedup (relative to execution time on 8 cores – smallest available core count) of the time-stepped and event-driven executions on *P*=8, 64, 512, 4096, and 8000. The time-stepped execution out-performs the event-driven approach as *P* grows to about 1K processors due to the extra computational overhead of event scheduling and global virtual time management necessary in any event-driven execution. Note that both types of executions were carried out with:

(a) threshold = -1,
(b) identical global linear ordering of the partial voltages, and
(c) identical parallel domain decomposition.

When threshold = -1, every update of a local partial voltage triggers a threshold –cross event at every half-integer time-step which forces the event-driven simulation to execute conservatively rather than speculatively. This ensures that there is no reverse computing overhead for the event-driven execution shown in Fig. 10. Same linear ordering of the partial voltages and identical parallel domain decomposition scheme ensures identical data distribution as well as identical number and volume of communication for both cases. As a result, t_{comp} and t_{comm} in t_p are equal for both types of executions. The remaining component of the total runtime is t_{synch} which is identically equal to zero in the event-driven simulation. Consequently, it out-performs the time-stepped algorithm as *P* grows. This is evident in Fig. 10. For non-negative thresholds, the only difference in the arguments above is that the computation cost of the event-driven execution increases due to reverse computing overhead.

But this added cost is proportion to the forward computation volume which only decreases as the processor number grows and continues to remain smaller than t_{synch} of the time-stepped execution. As a result, the event-driven simulation scales to a much larger number of processors when compared to the time-stepped simulation. This is evident from Fig. 11 in which threshold = 0.001 on P = 8, 64, 512, 4096, 8000, and 12,000.

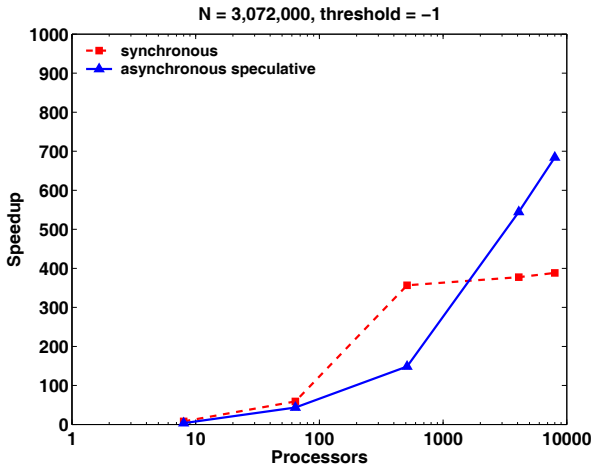

Fig. 10. Speedup comparison with threshold = -1.

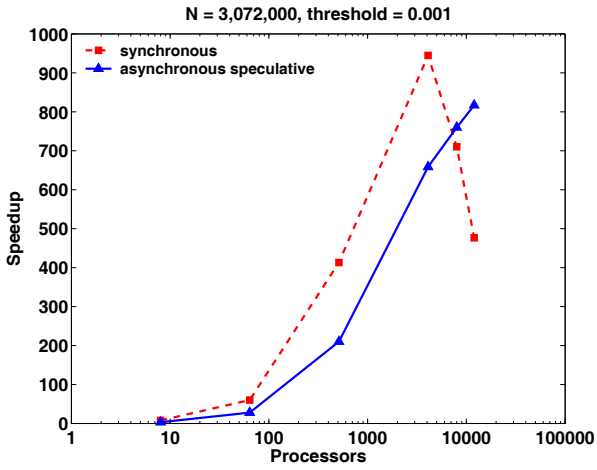

Fig. 11. Speedup comparison with threshold = 0.001.

8. 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 1000× 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. 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.

An exhaustive performance comparison of conventional time-driven parallel approaches with the event-driven parallel algorithm has been presented. The comparison clearly revealed that, unlike discrete event based schemes, barriered time-driven algorithms are prone to large synchronization overheads that grow as the number of processors increase. This point of

comparison is particularly poignant in an era of petascale computers to demonstrate the fact that synchronization overheads drastically hinder the performance of barriered time-driven codes. Convincing empirical results illustrative of the advantages of asynchronous speculative execution over synchronous time-stepped for the simulation of radio signal propagation have been provided.

ACKNOWLEDGEMENTS

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 DE-AC05-00OR22725.

This research used resources of the National Center for Computational Sciences at the Oak Ridge National Laboratory, which is supported by the Office of Science of the US Department of Energy under Contract DE-AC05-00OR22725.

REFERENCES

1. J. Nutaro, "A Discrete Event Method for Wave Simulation," *ACM Transactions on Modeling and Simulations*, vol. 16, pp. 174–195, 2006.
2. 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*, vol. 56, pp. 189–198, 2008.
3. S. D. Bilbao, *Wave and Scattering Methods for Numerical Simulations*: Wiley, 2004.
4. 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*, 1998, pp. 865–869.
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*, 2002, pp. 802–806.
6. 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*, 2002, pp. 38–43.
7. I. Gruber and H. Li, "Behavior of Ad Hoc Routing Protocols in Metropolitan Environments " in *Proc. of the Vehicular Technology Conference*, 2004, pp. 3175–3180.
8. 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*, 2007, pp. 676–684.
9. R. M. Fujimoto, "Parallel Discrete Event Simulation," in *Proc. of the Winter Simulation Conference*, 1989, pp. 19–28.
10. K. S. Perumalla, "Parallel and Distributed Simulation: Traditional Techniques and Recent Advances," in *Proc. of the Winter Simulation Conference*, 2006, pp. 84–95.
11. C. Carothers, K. S. Perumalla, and R. M. Fujimoto, "Efficient Optimistic Parallel Simulations using Reverse Computation," *ACM Transactions on Computer Modeling and Simulations*, vol. 9, pp. 224–253, 2006.

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

Appendix A. INITIALIZATION OF MODELS AND CONSISTENCY OF STATES

At every time step, we can define the state $S(t)$ of the system as the set $S(t) = [X(t), V(t)]$ where $X(t)$ and $V(t)$ are related to each other through the TLM equations that define the model being simulated. At any given time step t, $S(t)$ contains complete information about all the partial voltages $X(t)$ defined on the links of the grid and the total voltages $V(t)$ defined on each node. In the example above, a $3\times3\times3$ grid with periodic boundary condition was used. A non-zero voltage source of magnitude V_s was placed at the center of the grid at time $t = 0$, that is, $V_{\lfloor n/2 \rfloor}(0) = V_s \neq 0$. Execution of the simulation, began with initializations $X(0) = 0$ and $V_i(0) = 0$ at all nodes except at the center where $V_{\lfloor n/2 \rfloor} = V_s$ [i.e., $V(0) = [0, 0, \cdots, V_s, \cdots, 0, 0]$]. It is clear that such an initial state does not respect the mutual relationships between $X(0)$ and $V(0)$ as defined by the TLM equations since $V_s = V_{\lfloor n/2 \rfloor}(0) \neq \sum_{\alpha} x^{\alpha}_{\lfloor n/2 \rfloor}(0) = 0$.We refer to such a state as an *inconsistent state*. A state $S(t)$ at a time t is called *consistent* if all the partial and total voltages at the time step t respect both the TLM equations simultaneously.

When the state of the system is evolved using the forward TLM equations and the system is brought from the state, $S(0)$ to the state, $S(1)$, a careful look will reveal that using the reversal equations on $S(1)$ will *not* restore $S(0)$. This is because $S(1)$ is a consistent state but $S(0)$ is not. The initial state was, in some sense, "arbitrary" though it was chosen to best reflect the physical condition of the system at $t = 0$ which may not necessarily be consistent with the mutual relationships amongst the state variables for $t > 0$

A.1. PROPAGATION OF INCONSISTENCY AND CONVERSION POINT

The lesson learned from the previous section is that even if the model that drives a simulation is reversible, an inconsistency is injected into it due to initialization(s) which can potentially render it irreversible. The resulting inconsistency can potentially propagate through the time steps until such an iteration t_c when the system state reaches consistency We refer to such a time iteration as the *critical point*, t_c.

In order to be able to use reverse computing techniques, one therefore needs to detect the critical point at runtime. To do this, when the simulation advances from iteration t to $t+1$, that is, the system evolves from $S_f(t) \rightarrow S_f(t+1)$ where the subscript indicates that the state was obtained during a forward computation, the reverse equations R can be used on $S_f(t+1)$ to restore the previous state, $S_r(t)$ and checked to see if $R : S_f(t+1) \rightarrow S_r(t) = S_f(t)$. If $S_r(t) \neq S_f(t)$, we mark time step t, as belonging to the inconsistent phase of the simulation; otherwise, we mark it as consistent. If the state is inconsistent, information about it will need to be stored in memory before the simulation is advanced. During a rollback to iteration t, reverse computing can be applied if $t \geq t_c$ and a combination of reverse computing and state saving will be needed if $t < t_c$. We will refer to

this algorithm as the *toggle forward algorithm*. Clearly, the memory needed to store state information in such an algorithm will be proportional to $t_c / t_{\max}$, where $t_{\max}$ is the maximum simulation time. The advantages of reverse computing are limited by this ratio.

A.2. GENERALIZATION

Clearly, the strategy outlined above to address the irreversibility that is artificially introduced into the model by initialization artifacts can only be of limited usefulness. When $t_c > t_{\max}$, information about all the states during the simulation will need to be stored. This reduces the rollback mechanism in a speculative execution to the memory intensive state saving approach which we wish to improve upon through reverse computing.

To address this issue, we propose introduction of an additional pre-processing stage, which we call *input regularization*, in the computation path. This modification is schematically shown in Fig. A-12. Since the source of the irreversibility is an artifact of inconsistent initializations, the regularization procedure takes the original input set of variables that defined the starting state $S(0)$ and converts them into another set of variables that satisfy the consistency conditions of the model. These modified variables define the new starting state $S_R(0)$ that is used to simulate the system of interest. This implies that $t_c = 0$ which in turn ensures that no state information need be stored, thereby restoring the merits of using reverse computing in lieu of state saving.

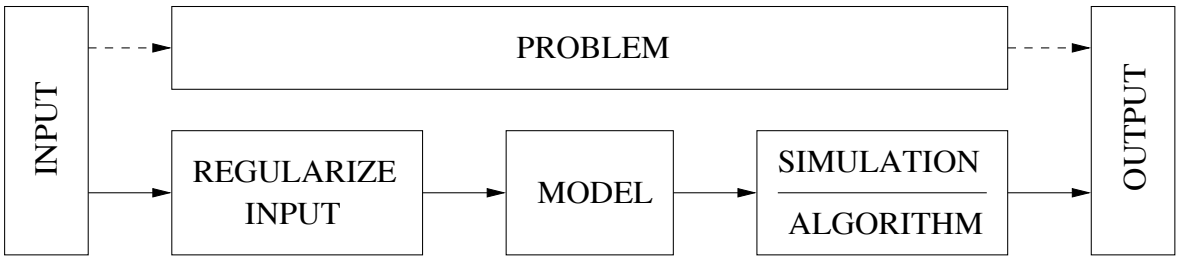

Fig. A-12. A preprocessing stage is introduced in the algorithm to regularize the input.

The regularization procedure should retain the description of the system state at $t = 0$ with the additional constraint that the state variables are consistent with the forward computation relations.

A.2.2 Example of Input Regularization for the EM Problem

At any given time step t, the consistency condition for the total and partial voltages can be defined using Eq.(1) and Eq.(2) as:

$$x_{ij}(t) + x_{ji}(t) = \frac{V_i(t-1) + V_j(t-1)}{3} - \left[x_{ij}(t-1) + x_{ji}(t-1) \right] \qquad \text{(A.1)}$$

Clearly, at $t < 0$, $V_i(t) = x_{ij}(t) = 0 \; \forall \; i, j$. This yields the following consistency conditions for the initial state $S(0)$:

$$x_{ij}(0) = -x_{ji}(0)$$
$$V_i(0) = \sum_k x_{ik}(0) \qquad \text{(A.2)}$$

Note that the magnitude and position of the source voltages at $t = 0$ are dictated by the physical problem and, hence, considered fixed. However, the partial voltages are an artifact of the TLM model that is used to describe the system and used to simulate its evolution. As such, any assignment of the partial voltages that satisfies the consistency conditions given by Eq. (A.2) constitutes a valid initial state. Keeping this in mind, the input regularization for the 3D-EM procedure can be defined by the following problem statement:

Problem: Given a periodic $n \times n \times n$ array of nodes with weights $W(i)$ assigned to each node i, find an assignment of weights $w(i, j)$ to each link (i, j) such that:

$$w(i, j) = -w(j, i)$$
$$W(i) = \sum_k w(i, k) \ , \ \forall\, k \in \{\textit{nearest neighbors of } i\} \qquad (21)$$

Fig. A-2–A-3 illustrate the input regularization of the TLM model in two dimensions with periodic boundary conditions. The direction of the links along which the indicated link weights are assigned is shown by the arrows. At $t = 0$, voltage sources of magnitude 1 and -1 are placed at the center and top left corner of a square domain that is decomposed into a $3 \times 3 \times 3$ array of grid points. The original input to the model is shown in Fig. A-2. Clearly, this state is inconsistent since the sums of the partial voltages do not equal the total voltages at the grid points to which the source voltages are assigned. Regularizing this input yields the assignment of partial voltages as shown in Fig. A-3. in which the total voltages at all the nodes are the same as those in the original with the additional property that the new set of partial voltages satisfy both consistency conditions ensuring that the initial state is consistent. This, in turn, implies that $t_c = 0$.

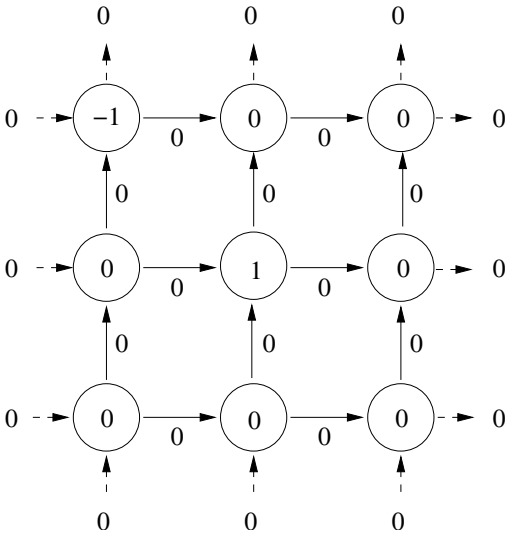

Fig. A-2. Un-regularized input state.

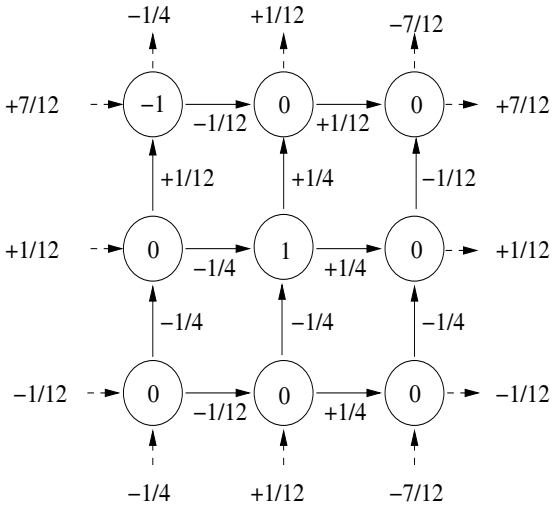

Fig. A-3. Regularized input state.

Appendix B. SOME RESULTS

Observation: Voltages after reversals are exactly equal to their values from forward computations.

Proof: let $\bar{V}_i(t)$ and $\bar{x}_{ij}(t)$ denote the total voltage and partial voltage at time step t computed at the grid point i using the reversal equations. From Eq. (15), we get:

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

Observation: Partial voltages after reversals satisfy the same consistency equations as in the forward computation.

Proof: Remember that $R_{ij} + T_{ji} = 1$ and $R_{ji} = -R_{ij}$. Then:

$$\begin{aligned}
&\bar{x}_{ij}(t) + \bar{x}_{ji}(t) = R_{ij} C_{ij}(t) + T_{ji} C_{ji}(t) + R_{ji} C_{ji}(t) + T_{ij} C_{ij}(t) = C_{ij}(t) + C_{ji}(t) \\
&= \frac{1}{3} [R_{ij} \bar{V}_i(t) + T_{ji} \bar{V}_j(t)] - x_{ij}(t+1) + \frac{1}{3} [R_{ji} \bar{V}_j(t) + T_{ij} \bar{V}_i(t)] - x_{ji}(t+1) \\
&= \frac{1}{3} [\bar{V}_i(t) + \bar{V}_j(t)] - [x_{ij}(t+1) + x_{ji}(t+1)] \\
&= \frac{1}{3} [\bar{V}_i(t) - V_i(t) + \bar{V}_j(t) - V_j(t)] + [x_{ij}(t) + x_{ji}(t)] \\
&= x_{ij}(t) + x_{ji}(t) = \frac{V_i(t-1) + V_j(t-1)}{3} - [x_{ij}(t-1) + x_{ji}(t-1)]
\end{aligned}$$

which is the same as Eq. (A.1).