

IMPROVING PERFORMANCE OF PARALLEL SIMULATION KERNEL FOR WIRELESS NETWORK SIMULATIONS

M. Thoppian, S. Venkatesan, H. Vu, R. Prakash, N. Mittal
Department of Computer Science, The University of Texas at Dallas,
Richardson, TX
and
J. Anderson
PACE Lab, Rockwell Collins,
Richardson, TX

ABSTRACT

For decades, simulations have been the means to study the behavior and analyze the performance of engineering, computer science, and military applications. In the sequential discrete-event driven simulations, the events are executed sequentially in an increasing order of their timestamp. Simulation of large complex applications using sequential simulators often turns out to be infeasible due to computer resource limitations. Parallelizing discrete-event simulation is acknowledged as the most promising approach for simulating large complex applications. Parallel discrete-event simulation refers to the execution of events from a single discrete-event simulation program in parallel. In this paper we focus on extracting parallel events for wireless ad hoc network simulations based on the distance and terrain information.

I. INTRODUCTION

With recent growth in the area of wireless communications, there is an increasing need to develop techniques to improve the simulation tools used to model large-scale wireless ad hoc networks. Sequential discrete-event driven simulation traditionally refers to the use of a global event list, a global clock, and a single processor to simulate the entire application. In sequential discrete-event driven simulation, the events are executed in an increasing order of their timestamp, one after another. Simulation of large complex applications often turns out to be beyond the capability of sequential simulators. Parallel simulations is one way of overcoming the resource limitations and achieving simulation speed-up.

In this paper we propose techniques to extract events that are non-conflicting, i.e. events that are causally independent. These non-conflicting events can be executed in parallel without violating the correctness of the simulation.

In a discrete event-driven simulation, the nodes schedule events and insert them in a global event list. The scheduler executes events in the order of their timestamp. Upon execution, the event is removed from the list and the simulation clock advances to the time of next event in the event list.

In sequential simulations, the events are executed one by one. Consider a sample event list shown in Figure 1. The

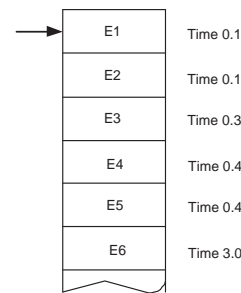


Fig. 1. Sample Event List

events E_1, E_2, \dots, E_6 are scheduled by different nodes in the network. These events are sorted in the increasing order of their timestamp. At the beginning of the simulation, events with timestamp 0 are executed. If there are no such events in the event list, the simulator advances the simulation clock to the timestamp of the next event in the event list. If there are multiple events scheduled for the same time, the scheduler executes these events one-by-one in the case of sequential simulations. In our example, after executing events E_1 and E_2 , the simulator advances the clock to 0.3 seconds. It then executes event E_3 and so on. However in the case of parallel simulations, events that are independent of each other can be scheduled in parallel. In our example, using parallel kernel the scheduler can execute both the events E_1 and E_2 simultaneously without affecting the correctness of the simulation. In addition to the events occurring at the same instance of time, there could be many other independent events in the simulation that can be processed in parallel.

Suppose events E_i and E_j had timestamps T_i and T_j , respectively. Let $T_i < T_j$. If event E_i changes the state of a variable referenced by E_j , then these two events are said to be causally related and executing E_j before E_i would result in a causal error. For example consider the Figure 2. Node N_1 sends messages M_1 and M_2 to node N_2 , such

that M_1 is sent before M_2 . The receipt of message M_1 has to precede the receipt of message M_2 at node N_2 to avoid causal order violation. In the past, several algorithms for parallel simulation have been proposed. The parallel simulation techniques proposed in the literature [1] have been classified into the following two categories:

- 1) Conservative approach: In conservative approach, all possible causality errors are prevented by strictly adhering to the local causality constraint.
- 2) Optimistic approach: In optimistic approach, the local causality constraint is not strictly adhered to. An optimistic approach instead guarantees to detect and recover from causality errors.

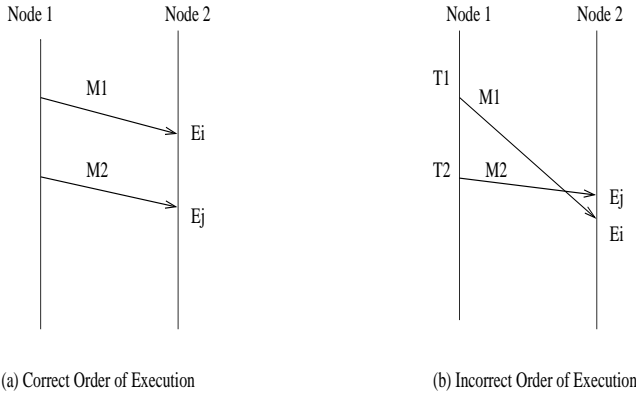


Fig. 2. Causal Relationship between E_i and E_j : (a) Correct order execution, (b) Incorrect order of execution resulting in causality error

In this paper we focus on the conservative approach for parallel simulations. To execute events in parallel it is necessary to know if the events are causally independent. The biggest challenge faced is: how does one identify all the events in the event list that are independent of each other? In this paper we enumerate ways of extracting events that are safe for parallel execution in static wireless ad hoc network simulations.

II. RELATED WORK

Several parallel simulation techniques have been proposed in the literature [1]. The simulation tools such as OPNET, GloMoSim, and Qualnet employ conservative parallel simulation techniques.

In GloMoSim [2], each protocol layer in a node is represented as an entity. Each of this entity could be a logical process. Meyer *et al.* [3] present ideas to improve lookahead based on the message paths through a protocol stack. This was termed as “path lookahead”. Each layer maintains a lookahead value with its neighbors in the stack and uses null messages to advance the lookahead.

The OPNET parallel simulation kernel [4] executes events in parallel if they occur within the same parallel

execution window. If two events occur within the same parallel execution time window and if they belong to different process models or different nodes, then the events are candidates for parallel execution. The current release of OPNET Modeler 11.5 has only the MAC and physical layer made parallel safe. As all the events within the window are candidates for parallel execution, the choice of this window value is crucial. Ideally, the size of the window should be dynamically set in the simulations based on the amount of available parallelism. However, the value of parallel execution time window in OPNET Modeler has to be set before running the simulation experiment and is a static value.

In Qualnet [5], the entire network is divided into partitions with each partition acting as a logical process. Zhengrong *et al.* [6] use MAC layer, PHY layer and cross-layer interaction between the MAC sub-layer and the PHY layer to extract lookahead values. At the MAC sub-layer the back off timers, inter frame spaces and duration of current frame being received are used to determine lookahead. At the PHY layer lookahead is extracted by considering the transition time between receiving and transmitting and by looking at interference signals. By exploiting the cross-layer interaction, a larger lookahead is extracted.

III. EXTRACTING PARALLELISM IN WIRELESS NETWORKS

In this section we present techniques to extract events that are non-conflicting. Two events are said to be non-conflicting if they are independent: the order in which they are executed does not affect the correctness of the simulation. Given a list of events, our objective is to find a set of events (say \mathcal{P}) that can be executed in parallel without violating the correctness of the simulations. The set \mathcal{P} can be extracted at the following three levels (i) Event level, (ii) Node level, and (iii) Group level.

A. Extracting Parallelism at Event Level

At the event level, events in the event list that are independent of each other can form the set \mathcal{P} . A simple algorithm to do this would be to pick the event with the smallest timestamp in the event list. All the events that are causally independent of this smallest event form the set \mathcal{P} . Once these events are executed, the next smallest event from the list is chosen and all the events that are causally independent of this event forms the next set \mathcal{P} . This is repeated until there are no more events in the event list. To determine if two events are causally independent, the application, routing, MAC, PHY layer information can be used. For example if there are two application layer

sessions running simultaneously at a node, then events from these two sessions can be executed in parallel such that the causal relationship is not violated. One way to ensure that causal relationship between events are not violated is by using lookahead. Lookahead represents a time interval within which the events are considered independent of each other. If the lookahead value is say δ , then the events within the time interval $(t, t + \delta)$ are considered to be independent of each other and can be executed in any order without violating the correctness of the simulation. Thus, lookahead gives a lower bound on when an action at a node would affect the state of another node. At the event level the lookahead value is computed by using cross-layer interaction information and node level interaction. Thus, events that belong to different layers within the same node or events that belong to different nodes can be executed in parallel as long as they are within this pre-determined lookahead value.

B. Extracting Parallelism at Node Level

At the node level, each event has a node ID attached to it. Lookahead between each pair of nodes is computed using (i) distance information between nodes, (ii) message inter-arrival time between nodes, (iii) terrain information, and (iv) channel characteristics. Based on the lookahead value computed at the node level, set \mathcal{P} can be extracted from the event list.

- 1) *Lookahead based on Distance Information:* Using the routing information at nodes, the hop distance between nodes can be computed. If the number of hops between two nodes A and B is say x and if t is the lower bound on the propagation delay for a message to travel one-hop, then xt is the value of lookahead between nodes A and B . Events at node A and node B can be executed in parallel as long as they are within xt time units of each other.
- 2) *Lookahead based on Inter-arrival times:* The traffic generators at the nodes can be used to compute the time interval between successive communication between a pair of nodes. For example suppose node A sends an application packet every T seconds to node B . Using the traffic generator at the node A , the value of T can be computed and this value can be used as the lookahead between nodes A and B .
- 3) *Lookahead based on Terrain Information:* Based on the terrain information, nodes can determine if they can communicate to each other directly. If nodes A and B are separated by an intervening terrain, then events from these nodes can be executed in parallel without violating the correctness. If a multi-hop route exists between node A and node B , then the hop

distance and/or interval of communication between nodes A and B can be used as a lookahead value.

- 4) *Lookahead based on Channel Characteristics:* In network scenarios where nodes have access to multiple channels, nodes using mutually orthogonal channels do not interfere with each other. Events at nodes using different channel for communication can be executed in parallel without violating the correctness of simulation. Based on the hop distance and/or interval of communication between nodes, a lookahead value can be derived upon.

C. Extracting Parallelism at Group Level

At the group level, each event has a logical process ID associated with it. The entire network is divided into groups, with each group acting as a logical process. Each logical process can be simulated on an separate physical processor. A network can be partitioned based on the following (i) Distance information, (ii) Terrain information, and (iii) Channel characteristics. Between each pair of logical processes, a separate value of lookahead is computed. The lookahead value can be determined by computing the hop distances between the logical processes or by using the interval of inter-group communication. Two events from two different logical processes can be executed in parallel if they are within the lookahead value of each other.

1) *Distance Information:* We present a simple algorithm for partitioning a wireless network into logical processes based on the distance between nodes in the network. We assume that the nodes are assigned unique IDs from $N1 \dots Nk$, with k being the number of nodes simulated in the network.

Partitioning Algorithm

A node is said to be covered if it has been assigned a logical process ID. The logical process ID indicates the logical process to which the node belongs.

- 1) Node with node id $N1$ and all its neighbors form a logical process (say $LP1$). Thus all of these nodes are assigned logical process ID $LP1$.
- 2) The lowest id uncovered node (say Ni) in the two-hop neighborhood of the node $N1$ forms the next logical process $LP2$. All the nodes that have not been assigned logical process ID and are neighbors of Ni would be assigned the logical process ID $LP2$.
- 3) This procedure is repeated until all the nodes in the network are covered.

Figure 3 shows the result of partitioning in a sample network.

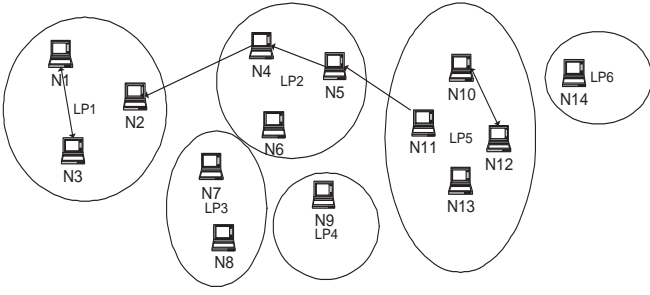


Fig. 3. Partitioning due to distance

2) *Terrain Information:* Based on the terrain information, nodes in a network can be partitioned into logical processes. Consider the example given in Figure 4. In this network scenario, two groups of nodes are separated by a hill (obstruction). Direct communication between these groups of nodes is not possible because of the obstruction. A router placed on the hill is used for inter-group communication. Most of the communication is taking place within the group and very infrequently the router at the top of the hill is used for the inter-group communication. In military applications, such scenarios occur often where two battalions are separated by an obstruction. Most of the time the soldiers in a battalion communicate among themselves and occasionally a message is sent to the soldiers in the other battalion. In parallel simulations, each of these groups can be simulated as a separate logical process.

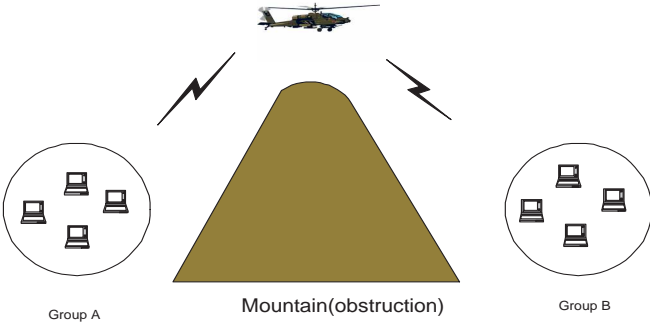


Fig. 4. Partitioning due to Terrain

3) *Channel Characteristics:* In network scenarios where nodes have access to multiple orthogonal channels, the channel characteristics information can be used to partition the network into groups. The network can be partitioned into groups, such that each group uses different channel for communication. Each of these groups can be considered as a logical process. Figure 5 gives an example of network using multiple channels. In this example nodes N1, N2 and N3 communicate with each other using channel 1, while

nodes N4, N5, and N6 use channel 2 for communication. Thus, nodes N1, N2 and N3 can form a logical process say $LP1$ and nodes N4, N5, and N6 can form another logical process $LP2$. After partitioning the network into logical

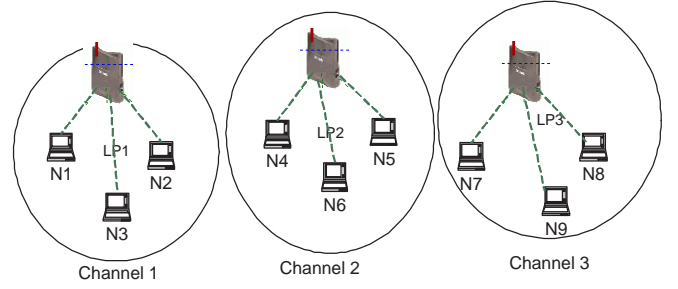


Fig. 5. Partitioning due to channel characteristics

processes, the lookahead value is determined as follows:

- Based on hop distance: The hop distance between logical processes is used to determine the lookahead. Suppose logical processes $LP1$ and $LP2$ are x hops apart. The events from $LP1$ and $LP2$ can be executed in parallel if they are with xt time units, where t is the lower bound on the one-hop propagation delay.
- Based on inter-group interaction: The lower bound on the time interval between inter-group communication can be used to determine the lookahead. Suppose a node within a logical process $LP1$ sends a message to a node within a logical process $LP2$ every T time units, then the value of T is an upper bound on the lookahead between $LP1$ and $LP2$.

IV. EXAMPLE

To illustrate how parallel events can be extracted at different levels, consider the network scenario in Figure 3. A sample event list is given in Figure 6.

E1, LP1, N1, 0.2
E2, LP1, N1, 0.24
E3, LP5, N12, 0.3
E4, LP1, N3, 0.35
E5, LP5, N12, 0.4
E6, LP5, N12, 0.44
E7, LP5, N10, 0.5
E8, LP5, N10, 0.54
E9, LP2, N5, 0.63
E10, LP2, N4, 0.71
E11, LP1, N1, 0.8

Fig. 6. Sample event list for the example network

Each event entry in the list is a four tuple $\langle EID, LID, NID, t \rangle$, where EID represents the event ID,

LID represents the logical process ID, NID represents the node for which the event is scheduled, and t represents the time at which the event is to be executed. Let the lower bound on the one-hop propagation delay be $p = 0.1$ seconds. Let us choose a value of 0.05 seconds as a lookahead within a node.

The Figure 7 shows the order in which events are executed in parallel simulation at the event level. At event level, if an event e_i occurs at node n at time t and another event e_j occurs at the same node within the interval $(t, t+\delta)$, where δ is the pre-determined lookahead value, then both the events can be executed in parallel. An event e_i occurring at node N_a at time t_i can depend on an event e_j occurring at time t_j at node N_b , if $t_j + (Hops_{ab} \times p) < t_i$, where $Hops_{ab}$ is the number of hops between nodes N_a and N_b . Hence these two events should be executed in parallel. In our example, events E1 and E2 occur at nodes N1 and N12, respectively. An event at time t at node N12 can change the state of node N1 not before $t + (6 \times p)$, where the number of hops between N12 and N1 is 6. Thus using the propagation delay value and the hop distance value, it is clear that events E1 and E2 are independent of each other and hence can be executed in parallel. Events E1 and E3 occurring at node N1 can be executed in parallel, as they fall within the lookahead value of 0.05 seconds. Event E4 cannot be executed with events E1, E2, and E3 because $(0.2 + p) < 0.35$ and $(0.24 + p) < 0.35$. Similarly, the rest of the events can be processed as shown in the Figure 7.

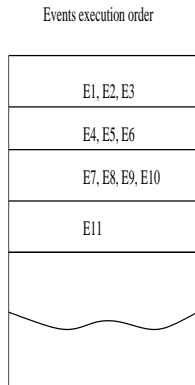


Fig. 7. Event execution order (event level)

Figure 8 shows the order in which events are executed in parallel at the node level. At the node level, we only consider the events occurring at different nodes. The events within the same node would be executed sequentially. Events E1 and E2 occur 0.1 seconds apart and at nodes that are six hops away of each other, so they are independent of each other. Hence this two events can be executed in parallel. Whereas events E1 and E3 occur at the same node and at the node level, we cannot execute them in parallel.

Events E3 and E4 occur 0.11 seconds apart and at nodes that are one hop away of each other. Hence there is a possibility that they are dependent events, so we cannot execute these events in parallel. The figure shows how remaining events can be processed at the node level.

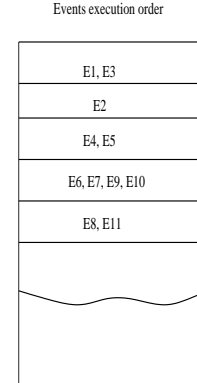


Fig. 8. Event execution order (node level)

Figure 9 shows the order in which events are executed in parallel at the group level. At the group level, the lookahead is computed based on the hop distance between logical groups. At group level, the lookahead between nodes N1 and N12 is $2p$ whereas at the node level it is $6p$.

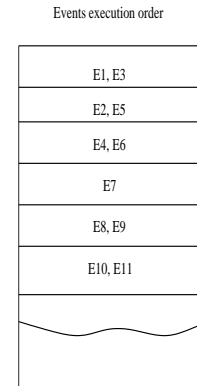


Fig. 9. Event execution order (group level)

V. DISCUSSION

At the event level in addition to the events belonging to different nodes, the events belonging to different layers within the nodes can be executed in parallel. Thus, the number of events executed in parallel could be more at the event level when compared to the number of events executed at the node or group level. However, the computational overhead to compute the set \mathcal{P} at the event level could be high.

At the node level, a higher value of lookahead can be extracted between nodes when compared to the lookahead

Number of Nodes	Total # of Events	Total # of Events in Parallel	% of Reduction in Execution Time	% of Events in Parallel
4 Node Network	2199	1420	32.3	64
10 Node Network	4855	1554	16	32

TABLE I
EXPERIMENT RESULTS.

value extracted at the group level. However the computational overhead of modeling each node as a logical process is higher when compared to modeling each group as a logical process.

VI. SIMULATION EXPERIMENTS

We conducted simulation experiments to evaluate the performance in terms of the amount of parallelism that can be extracted. We ran the experiments on a sequential simulator and logged all the events that occurred during the sequential run. In this section we show what percentage of total events can be executed in parallel at the group level. The network is divided into two groups. A node in one group sends a stream of packets to a node in another group. This is a constant bit-rate traffic with packets sent every 0.5 seconds. The packet size is 512 bytes. We conducted experiments for a 4 node and a 10 node network. The Table I shows the percentage of events that can be executed in parallel over a dual processor system. The table also shows the total number of events that were simulated. The percentage of events that are executed in a 4 node network is seen to be higher than a 10 node network. This is because, at the group level, the number of events extracted for parallel execution mainly depends on the number of partitions. In both the experiments, we had partitioned the network in two partitions, hence for a four node network we get larger percentage for parallel execution.

VII. CONCLUSION

With unprecedented growth in the usage of military wireless communication devices, there is an increasing need for simulating high fidelity, large-scale wireless networks in shorter time. In this paper, we presented ways of extracting parallelism available in wireless network simulations at (i) event level, (ii) node level, and (iii) group level. Using the terrain information, distance information, medium characteristics, and cross-layer interaction, a set of non-conflicting events can be extracted and executed in parallel without

violating causality constraints. In the future we intend to evaluate each of the techniques presented in this paper by conducting simulation experiments.

REFERENCES

- [1] Fujimoto, R.M. Parallel discrete-event simulation. In *Communications of the ACM*, October 1990.
- [2] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla. GloMoSim: A scalable network simulation environment. Technical Report 990027, UCLA, Computer Science Department, 1999.
- [3] Meyer, R.A and Bagrodia, R.L. Path Lookahead: A Data Flow View of PDES Models. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation (PADS'99)*, 1999.
- [4] OPNET. *OPNET Modeler Documentation* <http://www.opnet.com/support>.
- [5] Qualnet. *Qualnet Parallel Developer Documentation* <http://www.scalable-networks.com/>.
- [6] Zhengrong Ji, Junlan Zhou, Mineo Takai, Jay Martin, and Rajiv Bagrodia. Optimizing Parallel Execution of Detailed Wireless Network Simulation. In *18th Workshop on Parallel and Distributed Simulation (PADS'2004)*, 2004.