*Proceedings of the 1999 Winter Simulation Conference*
*P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds.*

# A GLOBAL SYNCHRONIZATION NETWORK FOR A NON-DETERMINISTIC SIMULATION ARCHITECTURE

Marc Bumble
Lee Coraor

Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16801, U.S.A.

## ABSTRACT

Our previous work presented methods of accelerating non-deterministic discrete event simulation at the processing element level. Here, two algorithms are proposed for synchronizing a network of processing elements according to the next network minimum event timestamp. One method has an expected running time of $O(k)$ while the second has an expected running time of $O(k \log(k))$. A network architecture is developed and simulation results of the time expected to locate and broadcast the next network minimum timestamp are reported.

## 1 INTRODUCTION

Methods of speeding up the runtime of Non-Deterministic Discrete Event Simulation are presented with the intent of developing an architecture capable of running a simulation fast enough to be practical for dealing with emergencies. One scenario which could utilize our results would be a traffic model of a city, such as Atlanta, during a large event, like the Olympic games. Assuming that the traffic model was already in place, if an emergency situation, for example, a terrorist bomb, required the re-routing of traffic, an operator should be able to add detour adjustments to the model, and quickly rerun the simulation to test proposed detours before their implementation. The intent here is to present parts of a machine architecture which are accelerated beyond the simulation speed possible with a normal general purpose computer. The result is a simulator capable of providing detailed accurate traffic simulations fast enough to be useful to traffic engineers and police during emergencies. Many deterministic simulation machines have been built as logic simulators. Yet there may be an even wider audience for non-deterministic machines.

In a simulation network, generalized by Figure 1, nodes are synchronized using either a time-driven or event-driven simulation approach. Time-driven simulation models step through every time increment whether or not an event occurs. This model is simple and easy to implement. However, processing empty simulation cycles detracts from an optimum simulation speed. Event-driven simulation models jump from simulation event to event, skipping the empty cycles; however, the cost of determining the next event time in a network of nodes and synchronizing the nodes to jump

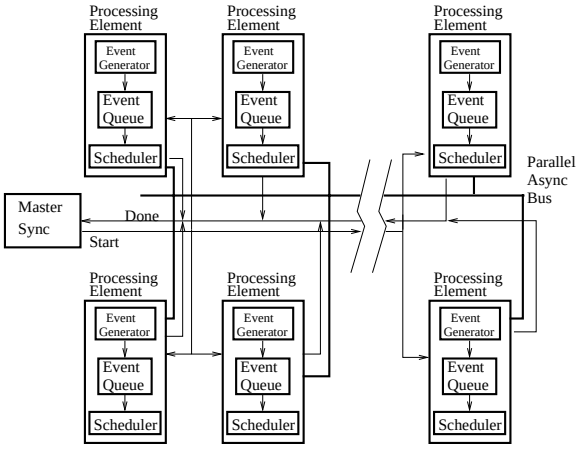


Figure 1: **A Network of Processing Elements.** A simulation network consists of event sources, sinks, and way-points. Each must be synchronized to the global system time clock. Two common methods of synchronization are time-driven and event-driven synchronization. Arrival statistics of the various sources and sinks can be used to gauge which method is faster. The time-driven simulator illustrated uses a master/slave approach similar to Levendel (Levendel 1982). The Master Synchronizer asserts the Start line at the beginning of each time cycle. Each network processor signals it is ready for the next time cycle by asserting its Done line. The Start and Done lines are configured as reduction network lines illustrated in Figure 4.

simultaneously may be higher than simply stepping through all the empty intermediate time intervals.

A single architecture can be constructed which allows a simulation to run as either a time or event-driven model. For the non-deterministic simulation architecture proposed, the decision between the two models is made at the beginning of the simulation and the selected model is used for the simulation duration. Hence, this paper also examines criteria that can be used to select the best alternative. Time-driven synchronization is relatively trivial, so the paper concentrates on event-driven systems.

A simulator is composed of individual nodes joined in a network. Each node consists of an event generator, local event queues, and a scheduler as depicted in Figure 2. To prevent causality errors, individual nodes are synchronized so that all process the same simulation cycle simultaneously. In conservative event-driven simulation, individual nodes all jump to the simulation cycle which coincides with the smallest timestamped event held within the network. Logistical difficulties occur in both the communication and sorting of the timestamps. Each node's local minimum timestamp must be compared against all of the local minimum timestamps in the global network.

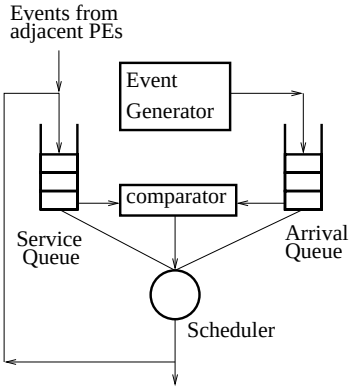


Figure 2: **The local processing element design** The local processing element (PE) design uses two queues for each server. The arrival queue holds the sorted list of arrival events from the Event Generator. Service events, which are created from processing successful arrival events, are stored in the Service Queue along with events from adjacent network processing elements. A comparator samples the heads of both queues and indicates where the next minimum local timestamped event resides.

This research also demonstrates that having both time and event-driven simulation capability can be worth the cost. A runtime comparison of event-driven versus time-driven simulation is presented. Also, the additional cost of the event-driven simulation capability is estimated. The proposed solution consists of a parallel bus and a reduction network, similar to the control network combination found in the CM-5 (Hord 1993). The mathematics presented are not intended as new contributions. Rather, the math results are used solely to assist in evaluating the expected speed of the new proposed synchronization methods.

### 1.1 Related Work

Discrete event-driven simulation is inherently serial in nature due to its *causality* (Nicol 1996) constraints. In 1987, Reed et al reported that the parallel implementation [of simulations] rarely completes more quickly than the sequential implementation [when distributed simulations are run with a central server network] (Reed 1987). In an effort to overcome similar obstacles, Fujimoto developed the Rollback Chip (Fujimoto 1992) to accelerate both state saving and simulation checkpoint restoration which are required by optimistic simulations when straggler events are encountered. Reynolds (1993) further established the requirement for high speed network computation of the Global Virtual Time (Jefferson 1985) and introduced the Parallel Reduction Network as hardware support to attain that goal. Reynold's method passes a vector of simulation values through a computation reduction tree composed of Arithmetic Logic Units (ALUs). Beaumont et al (1994) employ hardware support for parallel discrete event simulation through the use of Field Programmable Gate Arrays (FPGAs). A ring of interconnected FPGAs determines which events contain the smallest common timestamp.

Our previous work presented methods of accelerating two processor level phases of a conservative discrete event simulation, the event generation phase (Bumble 1997) and the storage phase within an event list (Bumble 1998a,b). This paper contributes two new algorithms which synchronize a network of nodes according to the smallest timestamped event held by the aggregation of processing elements. Unlike Reynolds, the synchronization methods presented here are not based on a message or vector passing scheme. The presented network also differs from Reynold's approach in that it is not a tree. The proposed network interconnection is more geometrically scalable and it avoids incurring message passing time penalties. The initial step in the proposed network algorithms is purely a logic reduction step, and should therefore also be much faster than the network proposed in Beaumont.

## 2 BUS ARCHITECTURE

Traditional approaches in multi-processor simulation search for the next smallest timestamp in a network of N processing elements. The simulation model may have n active model nodes distributed across the N processing elements in a balanced fashion, but each processing element will have one minimum timestamp for the model nodes it handles.

Each processor minimum timestamp must be compared against the other processor minimum timestamps found in the network. Some of the more commonly encountered network search algorithms include network structures constructed as k-ary trees depicted in Figure 3 and found in Reynolds (1993). To determine the minimum timestamp in such a network requires $\log_k(n)$ communications steps. The smallest timestamp is filtered to the root of the tree, and from there the result must be distributed to the rest of the network. One disadvantage of this system is that the wire interconnect run lengths may become progressively longer approaching the root of the tree. This method requires $O(\log_k(n))$ communications steps. If all processing element minimum timestamps were required to be compared against each other, $O(\log_k(n))$ steps would be the fastest synchronization rate a simulator could hope to achieve.

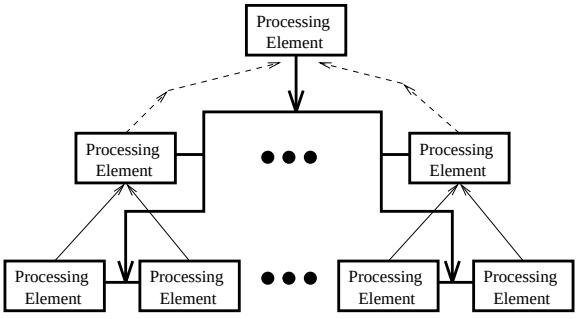


Figure 3: **K-ary Search Tree Network** The K-ary search network topology allows N processing elements in a network to compare individual local minimum timestamp results to the winner of the K elements on the level below. Successive winners compete in tournament style comparisons.

Section 3 presents a method which avoids individual timestamp comparisons. For large mean arrival rates, the new method approaches $O(k)$. Multiple transmitters share the bus and are able to generate signals simultaneously. The bus architecture is composed of a bi-directional reduction logic network employing Emitter Coupled Logic (ECL) (Pickles 1997) which gives the interface reasonable transmission speed, and ECL hardware connects nicely with CMOS technology (Chappell 1988). ECL switching speed is accomplished by keeping transistors always biased in their active regions. This bias alleviates the delay required for the sufficient accumulation of charge to change the state of saturated gates. The gates switch much faster than conventional CMOS gates; however, they also burn significantly more power. OR or NOR logic is used to run buses in two directions as depicted in Figure 4. Reduction logic can be accomplished directly at the processing element I/O points without processor intervention.

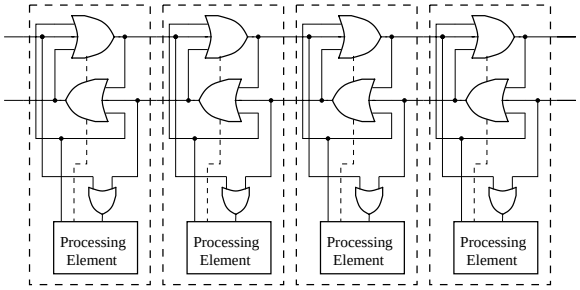


Figure 4: **The PE Interconnection Network** The interconnections consist of high speed Emitter-Coupled Logic (ECL) gates. ECL buses link the Processing Elements (PE) together, allowing a rapid and semi-parallel determination of the next smallest network timestamp. The OR network assists in the computation of the smallest timestamp and serves for both computation and bus signal driving. The PEs can both assert and read the bus signal values. A dashed control line is provided to allow the PE to prevent the propagation of signals to subsequent elements on the bus. The use of this control line is illustrated in Figure 5.

The bus illustrated in Figure 4 is used to form a consensus among the processing elements. If any element asserts a signal on the bus, the signal is propagated so that all elements in the network quadrant are aware that at least one element has asserted the signal. This broadcast mode is employed by the algorithm described in Section 3. Each gate also contains an additional control line which, when asserted, prevents signals from propagating further. The on/off switching capability is used to pair elements in Section 3.2.

## 3 SEARCH ALGORITHM

The algorithms for finding the network minimum timestamp proceed in two basic phases. The first phase consists of a general elimination which prunes processing elements having timestamps larger than $2^k$, the base 2 ceiling of the global minimum timestamp. The second phase of the algorithm then finds the minimum among the remaining nodes.

### 3.1 Phase 1 Elimination

First, all network processing elements (PEs) find their local minimum values. This search involves comparing the lead elements of the service and arrival queues from Figure 2 in $O(1)$ time. A hardware algorithm for maintaining the smallest event within a local processing element is presented in Bumble (1998b). Next, each PE computes the difference between the current global simulation time cycle and the

next local minimum timestamp, $t_{diff}$, in $O(1)$ time. The current global simulation time cycle is always available at each node. Each PE determines the number of bits, $b$, required to express $t_{diff}$. For example, 13, requires 4 bits, $1101_2$. The PEs simultaneously assert the signal line representing $b$ on the global parallel Data Bus illustrated in Figure 5. After all PE's have floated their $b$ values on the bus in $O(1)$, the PEs whose $b$ value is greater than the bus minimum signal line eliminate themselves from the search. The smallest asserted signal line of the parallel bus narrows the scope of the search to the limited range of numbers expressed in Equation 1:

$$2^b - 2^{b-1} = 2^{b-1}(2-1) = 2^{b-1} \quad (1)$$

All elements not eliminated in this first phase are referred to as active elements in the second phase.

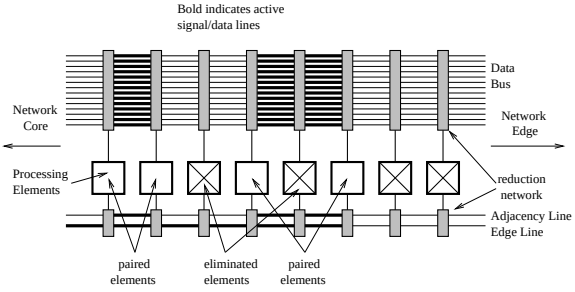


Figure 5: **Algorithm Phase 2 Method 2** Elements eliminated by the initial reduction step are illustrated inscribed with a cross. Signals flow unimpeded through the eliminated processing elements (PEs). The data signals are shown, in bold, traversing the upper bus. The lower two-signal bus represents the basic handshaking signals. The Edge signal indicates to each PE whether or not that PE is on the network edge. All PEs which have not self-eliminated during the first phase generate an active Edge signal and propagate the signal towards the network core. The Adjacency signal is used to pair PEs. Each active PE, which receives the Edge signal but not the corresponding Adjacency signal, propagates its own Adjacency signal towards the direction of the network core. When either another active PE or the core receives the Adjacency signal, that PE does not propagate the signal but instead compares its minimum local timestamp with the timestamp value received on the Data bus. The minimum value of the pair becomes the minimum value at the node closest to the core while the outer pair node is eliminated.

### 3.2 Phase 2 Selection

The second phase of the algorithm can proceed in either of two methods. Method one requires a 3-bit reduction network, and method two requires a 2-bit reduction network. The first method performs a binary search through the range of timestamps isolated in Phase 1. The second method performs binary eliminations, tournament style, among the remaining active nodes. The reduction network can also serve as the Start and Done lines for the Master Synchronizer described by Figure 1.

In the first method, a Bus Master begins a binary search through the remaining interval of time, described by Equation 1, to determine the minimum global timestamp. A reduction network is used to allow the PEs to signal whether their values are higher, equal, or lower than the value floated on the parallel bus. Using Equation 1, the global search can be completed in $O(\log_2(2^{b-1})) = b-1$, and the resulting global minimum timestamp range is visible to all PEs simultaneously.

The selection phase has several significant advantages over tree search methods. One advantage is the initial elimination step which occurs across the network at all PEs simultaneously. This advantage is opposed to a k-ary tree in which the first comparison happens at the lowest level only. Another significant advantage is that the network is somewhat more conducive to geometric element layout as opposed to a tree, where the interconnections between element levels get progressively longer. At [gate] logic speeds of two nanoseconds, an equivalent gate delay is introduced by every foot of the interconnecting lines (MECL 1989). The gate delays of the proposed devices are expected to be approximately one nanosecond (Pickles 1997). The 3-dimensional cubic array network illustrated in Figure 6 keeps all interconnection lengths uniformly short. The most important advantage is the significant reduction in the expected number of timestamp transmissions and coalitions. Its disadvantages include requirements for additional hardware and bus lines as illustrated in Figures 4 and 5. The event-driven simulation cost requires the extra Data Bus illustrated in Figure 5. The Edge and Adjacency lines along with each line of this Data Bus must be configured as illustrated in Figure 4 requiring three OR gates and two wires per signal.

The second method proposed for Phase 2 allows the processing elements which remain after the first phase to work in adjacent pairs. All active processing elements generate a signal which is passed towards the core along the network Edge signal line. Therefore any PE and the core receiving this signal know that there exists at least one active element on their network edge side. Next, the elements use their Adjacency signal line to form processor pairs. Active elements at the edge of the network propagate both the Edge and Adjacency signals. The next innermost

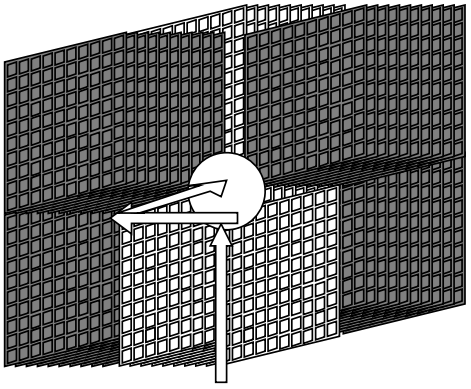

Figure 6: **The 3-Dimensional Network Structure** Although trees, as depicted in Figure 3, have a logarithmicly decreasing structure, they offer difficult geometric constraints for actual implementation. A linear parallel bus offers a more implementable structure. In the network illustrated, each bus is composed of reduction logic as shown in Figure 4. Much of the communications can be accomplished by the Process Element (PE) I/O cells. The length of each bus is a trade-off between communications element switching speed, bus signal propagation speed, and physical PE geometry constraints. In this figure, the PE's are arrayed along linear busses. If there are 10 elements on each bus, then the network may contain 8000 elements. The core may be composed of more than one processor, but for the purposes of this paper, the core is assumed to be one unit.

active element will receive both the Edge and Adjacency signals along with the value of the smallest timestamp on the data lines as shown in Figure 5. This inner core side element will propagate only the Edge signal towards the core. Having alternating elements propagate the Adjacency signal facilitates a pairing of the network elements. In each pair, the element closer to the network edge automatically self eliminates. The inner paired element compares its local minimum timestamp with the value received on the data-bus. The smaller value becomes the minimum used in the next cycle. The core retains the smallest value until all eight network quadrants have reported in, and then the core broadcasts the final result. In addition to the geometric network layout advantage of the first method, the second advantage of this mode is that as the number of nodes, N, increases, the expected time of the minimum timestamp becomes more isolated. The first elimination becomes the only step required.

## 4 EVENT VERSUS TIME-DRIVEN SIMULATION

This section derives equations which characterize the expected arrival time of the next event for two Independent and Identically Distributed (IID) networks. One network's processors generate exponentially distributed event arrivals. The other network arrival rates follow a Weibull distribution.

When confronted with a network of random event generators, the next expected event time to occur can be calculated using *Ordered Statistics* (Scheaffer 1990). We need to determine the shortest expected arrival time in order to evaluate which simulation approach is the most appropriate.

Let $X_1, X_2, \ldots, X_n$ denote independent continuous random variables which have distribution functions $F_1(x)$, $F_2(x)$, $\ldots, F_n(x)$ and density functions $f_1(x), f_2(x), \ldots, f_n(x)$ respectively. Ordered random variables, $X_i$, are denoted $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$ where $X_{(1)} \leq X_{(2)} \leq \ldots \leq X_{(n)}$. Continuous random variables allow the equality signs to be dropped. So the minimum value is

$$X_{(1)} = min(X_1, X_2, \ldots, X_n) \tag{2}$$

The density function of $X_{(1)}$, denoted $g_1(x)$ can be found as:

$$\begin{aligned} & P\left[X_{(1)} \leq x\right] = 1 - P\left[X_{(1)} > x\right] \\ & = 1 - P\left(X_1 > x, X_2 > x, \ldots, \ X_n > x\right) \\ & = 1 - [1 - F_1(x)][1 - F_2(x)] \cdots [1 - F_n(x)] \end{aligned} \tag{3}$$

Taking the derivative of both sides yields the density function,

$$\begin{aligned} g_1(x) \quad = \quad & f_1(x)[1 - F_2(x)] \cdots [1 - F_n(x)] + \\ & [1 - F_1(x)]\, f_2(x) \cdots [1 - F_n(x)] + \ldots \\ & [1 - F_1(x)][1 - F_2(x)] \cdots f_n(x) \end{aligned} \tag{4}$$

The expected time of the next arrival event can then be calculated by finding the expectation of $g_1(x)$ as follows:

$$E(x) = \int_0^\infty x g_1(x) dx \tag{5}$$

For an actual simulator, the computation of Equation 5 would be automated given that the user supplies the appropriate $F(x)$ and $f(x)$. For the purposes of this paper, the expected minimum timestamp is derived for 2 sample distributions, the Exponential and Weibull Distributions. These mathematical results are not intended as new contributions. The equations are used to derived the results of Section 5.

### 4.1 Exponentially Distributed Example

As a simple example, the next expected event time for a network of two exponentially distributed event generators is cal-

culated. The exponential density function is provided in Equation 6:

$$f(x) = \frac{1}{\theta}\ e^{-\frac{x}{\theta}} \tag{6}$$

The exponential cumulative distribution function can then be calculated as:

$$\begin{aligned} F(x) &= 0 && for\ t < 0 \\ F(x) &= P(X \le x) = \int_0^x \frac{1}{\theta}\ e^{-\frac{x}{\theta}} dt \\ &= -e^{-\frac{x}{\theta}}\Big|_0^x = 1 - e^{-\frac{x}{\theta}} && for\ t \ge 0 \end{aligned} \tag{7}$$

The density function for this example just contains two generators, so Equation 4 simplifies to:

$$g_1(x) = f_1(x)\,[1 - F_2(x)] + [1 - F_1(x)]\,f_2(x) \tag{8}$$

Substituting equations 6 and 7 into Equation 8 yields the following probability density function, $g_1(x)$:

$$\begin{aligned} g_1(x) &= \left(\tfrac{1}{\theta_1}\ e^{-\frac{x}{\theta_1}}\right)\left[1 - (1 - e^{-\frac{x}{\theta_2}})\right] + \\ &\quad \left[1 - (1 - e^{-\frac{x}{\theta_1}})\right]\left(\tfrac{1}{\theta_2}\ e^{-\frac{x}{\theta_2}}\right) \\ &= \left(\tfrac{1}{\theta_1} + \tfrac{1}{\theta_2}\right)\ e^{-\frac{(\theta_1+\theta_2)}{\theta_1\theta_2}x} \end{aligned} \tag{9}$$

Next, the development of the expected value of the minimum is derived by inserting Equation 9 into Equation 5 to derive:

$$\begin{aligned} E(x) &= \int_0^\infty x\left(\frac{1}{\theta_1} + \frac{1}{\theta_2}\right)\ e^{-\frac{(\theta_1+\theta_2)}{\theta_1\theta_2}x} dx \\ &= \left(\tfrac{1}{\theta_1} + \tfrac{1}{\theta_2}\right)\int_0^\infty x\ e^{-\alpha x} dx \\ &\quad where\ \alpha = \tfrac{(\theta_1+\theta_2)}{\theta_1\theta_2} \end{aligned} \tag{10}$$

Equation 10 can be simplified by applying the following $\Gamma(n)$ definition (Beyer 1978):

$$\Gamma(n) = \int_0^\infty x^{n-1}\ e^{-x} dx \tag{11}$$

The situation in Equation 10 is slightly different due to the $\alpha$ in the exponent. We can massage Equation 10 by using the substitution $y = \alpha x$ and $dy = \alpha dx$. Equation 12 is derived:

$$\begin{aligned} &\int_0^\infty x^{n-1}\ e^{-\alpha x} dx = \int_0^\infty \left(\frac{y}{\alpha}\right)^{n-1}\ e^{-y}\frac{1}{\alpha} dy \\ &= \left(\tfrac{1}{\alpha}\right)^n \int_0^\infty y^{n-1}\ e^{-y} dy = \left(\frac{1}{\alpha}\right)^n \Gamma(n) \end{aligned} \tag{12}$$

The results of Equation 12 and the substitution for $\alpha$ can be inserted back into Equation 10. For Equation 10, n = 2. Note that $\Gamma(n) = (n-1)!$.

$$\begin{aligned} E(x) &= \left(\tfrac{1}{\theta_1} + \tfrac{1}{\theta_2}\right)\Gamma(n)\left(\tfrac{1}{\alpha}\right)^n \\ E(x) &= \left(\tfrac{1}{\theta_1} + \tfrac{1}{\theta_2}\right)\Gamma(2)\left(\tfrac{1}{\alpha}\right)^2 \\ E(x) &= \left(\tfrac{1}{\theta_1} + \tfrac{1}{\theta_2}\right)(2-1)!\left(\tfrac{1}{\alpha}\right)^2 \\ E(x) &= \tfrac{\theta_1\theta_2}{\theta_1+\theta_2} \end{aligned} \tag{13}$$

The results of Equation 13 can be interpreted as follows. If the mean of the first process, $\theta_1$, is 5 seconds, and the mean of the second process, $\theta_2$, is 10 seconds, then the expected minimum arrival time of the two processes is given by Equation 13 as:

$$\begin{aligned} E(x) &= \tfrac{\theta_1\theta_2}{\theta_1+\theta_2} \\ E(x) &= \tfrac{5\cdot 10}{5+10} = 3.33\ seconds \end{aligned} \tag{14}$$

For three exponentially distributed event generators, the next expected event for x would occur at time:

$$E(x) = \tfrac{\theta_1\theta_2\theta_3}{\theta_2\theta_3+\theta_1\theta_3+\theta_1\theta_2} \tag{15}$$

The software simulations reported in Section 5 assume that all event generators create events with the same $\theta$ value. Therefore the expected value of the next event in Equation 15 can be generalized to Equation 16, where N is the number of processing elements.

$$E(x) = \tfrac{\theta}{N} \tag{16}$$

### 4.2 Weibull Distribution Example

As a second example, an IID Weibull distribution is calculated. This example is calculated for a 2 source experiment similar to Equation 14. The Weibull distribution has the density function found in Equation 17.

$$f(x) = \tfrac{\gamma}{\theta}x^{\gamma-1}e^{-x^\gamma/\theta} \quad for\ x > 0 \tag{17}$$

Equation 17 can be integrated directly to derive the cumulative distribution function in Equation 18.

$$F(x) = 1 - e^{-x^\gamma/\theta} \quad x > 0 \tag{18}$$

Equations 17 and 18 can be inserted into Equation 8. Also, if IIDs are assumed, then $\gamma_1 = \gamma_2$ and $\theta_1 = \theta_2$ allowing additional simplifications:

$$\begin{aligned} g_1(x) &= \left(\tfrac{\gamma_1}{\theta_1} x^{\gamma_1 - 1} e^{-x^{\gamma_1}/\theta_1}\right)\left(e^{-x^{\gamma_2}/\theta_2}\right) + \\ &\quad \left(e^{-x^{\gamma_1}/\theta_1}\right)\left(\tfrac{\gamma_2}{\theta_2} x^{\gamma_2 - 1} e^{-x^{\gamma_2}/\theta_2}\right) \\ &= \tfrac{2\gamma}{\theta} x^{\gamma - 1} e^{-2x^{\gamma}/\theta} \\ &\quad where\ \theta_1 = \theta_2\ and\ \gamma_1 = \gamma_2 \end{aligned} \tag{19}$$

To find the expected minimum, the results of Equation 19 are inserted into Equation 5:

$$E(x) = \int_0^{\infty} \frac{2\gamma}{\theta} x^{\gamma} e^{-2x^{\gamma}/\theta} dx \tag{20}$$

If we then let $y = x^{\gamma}$ so that $x = y^{1/\gamma}$ and $dx = \frac{1}{\gamma} y^{\frac{1-\gamma}{\gamma}} dy$, we can substitute these results into 20 to obtain:

$$\begin{aligned} E(x) &= \tfrac{2\gamma}{\theta} \int_0^{\infty} y e^{-2y/\theta} \frac{1}{\gamma} y^{\frac{1-\gamma}{\gamma}} dy \\ &= \tfrac{2}{\theta} \int_0^{\infty} y^{1/\gamma} e^{-2y/\theta} dy \end{aligned} \tag{21}$$

Employing the Gamma function substitution (Beyer 1978) listed in Equation 22, where $a = \frac{2}{\theta}$ and $n = \frac{1}{\gamma}$ we find a solution for Equation 21.

$$\int_0^{\infty} x^n e^{-ax} dx = \frac{\Gamma(n+1)}{a^{n+1}} \tag{22}$$

$$\begin{aligned} E(x) &= \tfrac{2}{\theta} \int_0^{\infty} y^{1/\gamma} e^{-2y/\theta} dy \\ &= \left(\tfrac{2}{\theta}\right)\left(\frac{\Gamma\left(\frac{1}{\gamma}+1\right)}{\left(\frac{2}{\theta}\right)^{\frac{1}{\gamma}+1}}\right) = \left(\tfrac{2}{\theta}\right)^{-\frac{1}{\gamma}} \Gamma\left(\tfrac{1}{\gamma}+1\right) \end{aligned} \tag{23}$$

Equation 23 has a nice solution for $\gamma = 1$ or $\gamma = 2$. For the latter case, the formula yields:

$$\begin{aligned} E(x) &= \left(\tfrac{2}{\theta}\right)^{-\frac{1}{2}} \Gamma\left(\tfrac{1}{2}+1\right) = \left(\tfrac{2}{\theta}\right)^{-\frac{1}{2}} \tfrac{1}{2}\Gamma\left(\tfrac{1}{2}\right) \\ &= \sqrt{\left(\tfrac{\theta}{2}\right)} \tfrac{1}{2}\sqrt{\pi} \end{aligned} \tag{24}$$

Equation 23 can be generalized for N different IID generators as:

$$E(x) = \left(\tfrac{N}{\theta}\right)^{-\frac{1}{\gamma}} \Gamma\left(\tfrac{1}{\gamma}+1\right) \tag{25}$$

Note that the larger the number of event generators which exist in the system, the shorter the expected time to the next event, $E(x)$. Although these examples use homogeneous distributions, it is assumed that the trend holds for independent heterogenous distributions as well. So the larger the number of event generators in the simulation, the faster the events will arrive, and the smaller the mean time between events grows.

## 5 RESULTS

Networks of processing elements deployed in three dimensional cubic arrays were simulated for both time and event-driven mechanisms. The event-driven synchronization time was computed assuming that the worst case signal propagation delay was required for all steps. The signal propagation delay is assumed to be composed of the time required to propagate a signal through each reduction gate in the array as depicted in Figure 4. The gates are assumed to be high speed Emmiter-Coupled Logic (ECL) one nanosecond delay gates (Pickles 1997). Processing elements are assumed to be deployed along linear buses whose lengths are determined by the number of processors in the simulation. The processing element connections to the bus are assumed to be spaced 10 centimeters apart, well within the run lengths listed for properly terminated ECL transmission lines (MECL 1989). Propagation delay along the bus is assumed to be 5 nanoseconds per meter excluding the time through the OR gate drivers. The peripheral buses of processing elements are assumed to connect to the middle level of linear buses. Gates at the end of the buses bridge onto the next bus layer. There are three interconnected bus layers illustrated in Figure 6.

The arrival distributions determine the time required to locate the next smallest timestamp in the network. For the time-driven simulation mode, the time required is composed of two components. The first component tallies the propagation delay as the signals are passed through each repeater gate illustrated in Figure 4. The second component is the propagation time in the wire runs between each gate. The signal must pass through the entire network from its edge elements to its core. The delay of each signal through the network is approximated in Equation 26.

$$\begin{aligned} delay &= ((array\_dim)(array\_length) + \\ &\quad array\_dim - 1)(switching\_time) + \\ &\quad (array\_length)(element\_spacing) \\ &\quad (prop\_delay) \end{aligned} \tag{26}$$

The event-driven simulation time was computed by using the same pair of components described above. One bus propagation/elimination step is always necessary. Then the power of 2, k, which is a logarithmic ceiling of the expected minimum network timestamp value is computed. Next, the number of events, E, which could be expected to arrive before $2^k$ is computed. Each comparison is assumed to require about 4ns. Finally, $log_2(E)$ determines the number

of comparisons needed to calculate the network minimum timestamp. Each communication through the network is assumed to be from edge to core, the worst case scenario.

In Figures 7-10, the two event-driven methods described in Section 3, are illustrated. The two methods are labeled *Event-Driven Range* and *Event-Driven Elements*. The Event-Driven Range algorithm, the first method described in Section 3.2, performs a binary search by dividing the range of possible time values isolated in the first elimination phase. Alternatively, in the second method from Section 3.2, the Event-Driven Element method, the algorithm takes advantage of the fact that as the distribution means increase, the remaining elements become more isolated at the extreme edges of the distribution curves. The first method must step through the remaining binary range of numbers, searching for the minimum. The second method tends to jump directly to the correct element in O(1) as the distribution means increase. All plotted time units are in nanoseconds.

Figure 7 plots the time required by the network minimum timestamp search algorithm as the number of processing elements and the event generator Exponential means vary. The Exponential event generators used in the distributions are IID. Figure 8 shows a slice of Figure 7 at the 1000 processing element mark. The graphs indicate a clear gain which can be harvested if the time and event-driven methods are used in conjunction. The second method in the second phase of the event-driven algorithm clearly yields significant gains for exponential distributions with large means. A scenario which would benefit from this algorithm might be one where the simulation has distribution means in the millisecond range but requires nanosecond resolution.

Figure 9 illustrates the results of a simulation using IID Weibull distributed sources. The plot varies the number of processing elements and the arrival rates of those elements. Time-driven simulation works well with smaller mean arrival rates, and the second proposed event-driven method, the Event-Driven Element method, works best with higher distribution means. Figure 10 clearly illustrates the greater potential range of benefit to be gained by a machine which can proceed using either a time or an event-driven approach.

The criteria used to select between the best simulation method is based on calculations determining which alternative requires less time. If the expected time to the next event can be computed using Equation 5, the rest of the process is pure accounting. The results of the computation yield the number of expected time-driven steps required to advance to each new event.

Using Equation 26, the time-driven estimate is approximated by Equation 27.

$$\textit{Time-Driven duration} \quad = \quad E(x)(\textit{delay}) \qquad (27)$$

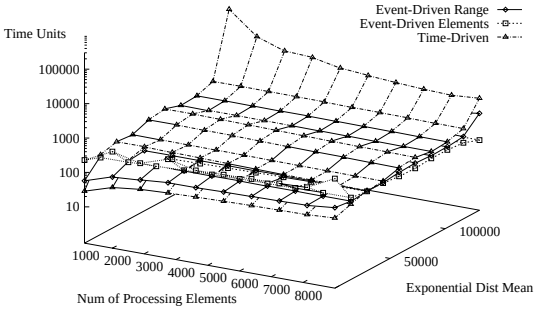


Figure 7: **Exponential Distribution in Event vs Time-Driven Simulation** The graph illustrates a network of IID nodes in network sizes ranging from 125 to 8000 nodes. Each node is generating arrival events according to an Exponential distribution with *mean* arrival times ranging from 1 to $2^{14}$. The graph illustrates that for an exponential arrival rate, the mean arrival time offers the most significant impact to network synchronization. The time required for the event-driven model is computed by counting the longest signal run from the edge to the center of the network multiplied by the propagation delay per unit length. One nanosecond is added for each OR gate, see Figure 4, encountered in traversing the path to the network core. The time-driven delay through the network is the duration of the number of time steps required.

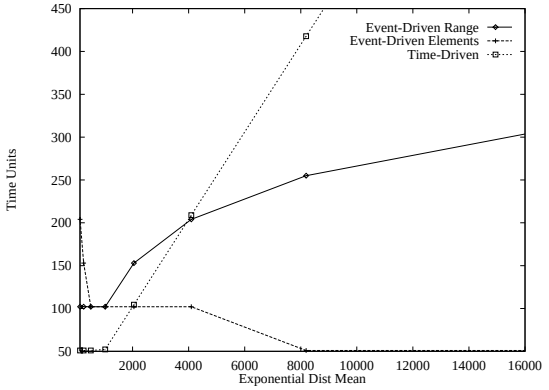


Figure 8: **Exponential Distribution slice of Figure 7** Illustrates a slice taken from Figure 7 where the simulation contains 1000 processing elements. The first method from Section 3.1 is labeled as the Event-Driven Range, and the second method from the same section is labeled Event-Driven Elements. The graph illustrates that a time-driven approach used in conjunction with the Event-Driven Element method provides the fastest network search approach.

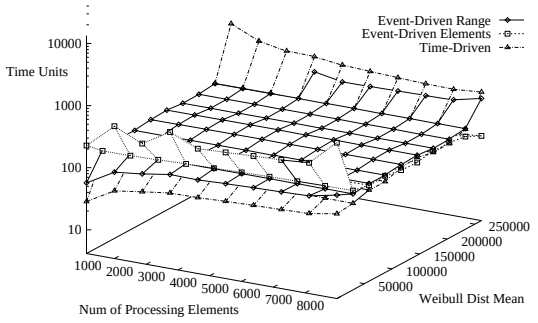


Figure 9: **Weibull Distribution in Event vs Time-Driven Simulation** The Weibull distribution results are similar to the Exponential. The optimum cross over point from the time-driven to the event-driven method allows a wider speedup gain to be derived.

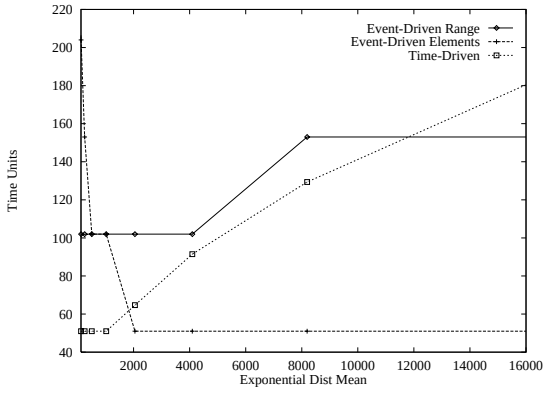


Figure 10: **Weibull Distribution slice of Figure 9** The graph is a slice of Figure 9 at the 1000 processing element mark. The relative simulation search times are displayed. Clearly, the optimal solution would be a simulator which could select between the Time and Event-Driven Element approaches.

The event-driven time estimate for Section 3.2 is computed by estimating the number of active elements left in the range which exists after the initial elimination step. The calculation shown in Equation 28 is composed of the delays required for the first phase elimination and the additional expected number of eliminations required by the second phase.

$$\textit{Event-Driven duration} = \textit{delay}(1 + \log_2(F(x) * N)) \tag{28}$$

The selection between event-driven and time-driven simulation is then a choice between the smallest estimate, provided by Equations 27 or 28.

## BIBLIOGRAPHY AND REFERENCES

Beaumont, C., Boronat, P., and Champeau, J. et al. Reconfigurable technology: An innovative solution for parallel discrete event simulation support. In *8th Workshop on Parallel and Distributed Simulation (PADS '94). Proceedings of the 1994 Workshop on Parallel and Distributed Simulation*, pages 160–163, Edinburgh, UK, July 1994. IEEE, SCS, San Diego, CA, USA.

Beyer, William H., editor. *CRC Standard Mathematical Tables*. CRC Press, Inc., Boca Raton, FL, 25 edition, 1978.

Bumble, Marc, and Coraor, Lee. Introducing parallelism to event-driven simulation. In *Proceedings of the IASTED International Conference–Applied Simulation and Modelling, ASM '97, Banff, Canada, July 27-August 1, 1997*, 1997.

Bumble, Marc, and Coraor, Lee. Architecture for a non-deterministic simulation machine. In *1998 Winter Simulation Conference Proceedings*, volume 2, pages 1599–1606, December 1998a.

Bumble, Marc, and Coraor, Lee. Implementing parallelism in random discrete event-driven simulation. In *Lecture Notes in Computer Science 1388, Parallel and Distributed Processing*, pages 418–427. IEEE Computer Society, Springer, March 1998b.

Chappell, Barbara A., Chappell, Terry I., Schuster, Stanley E., Segmuller, Herman M., Allan, James W., Franch, Robert L., and Restle, Phillip J. Fast cmos ecl receivers with 100-mv worst-case sensitivity. *IEEE Journal of Solid-State Circuits*, 23(1):59–67, February 1988.

Fujimoto, Richard M., Tsai, Jya-Jang, and Gopalakrishman, Ganesh C. Design and evaluation of the rollback chip: Special purpose hardware for time warp. *IEEE Transactions on Computers*, 41(1):68–82, January 1992.

Hord, R. Micheal. *Parallel Supercomputing in MIMD Architectures*. CRC Press, Inc., Boca Raton, Florida, 1993.

Jefferson, David R. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.

Levendel, Y. H., Menon, P. R., and Patel, S. H. Special-purpose computer for logic simulation using distributed processing. *The Bell System Technical Journal*, 61(10): 2873–2909, December 1982.

Motorola. *MECL Device Data*, 1989.

Nicol, David M. Principles of conservative parallel simulation. In *Proceedings of the 1996 Winter Simulation Conference*.

Pickles, Neil S., and Lefebvre, Martin C. ECL I/O buffers for BiCMOS integrated systems: A tutorial overview. *IEEE Transactions on Education*, 40(4):229–241, November 1997.

Reed, Daniel A., Molony, Allen D., and McCredie, Bradley D. Parallel discrete event simulation: A shared memory approach. *Proc of the 1987 ACM SIGMETRICS Conf on Meas and Model of Comput Syst*, 15(1):36–38, May 1987.

Reynolds, Jr., Paul F., Pancerella, Carmen M., and Srinivasan, Sudhir. Design and performance analysis of hardware support for parallel simulations. *Journal Of Parallel And Distributed Computing*, 18(4):435–453, August 1993.

Scheaffer, Richard L. *Introduction to Probability and its Applications*. The Duxbury Advanced Series in Statistics and Decision Sciences. PWS-KENT Publishing Company, Boston, USA, 1990.

## AUTHOR BIOGRAPHIES

**MARC BUMBLE** is a graduate student in the Computer Science and Engineering department at the Pennsylvania State University in University Park, PA. He received his B.S. and M.S. degrees in Electrical Engineering from the University of Pennsylvania in Philadelphia. His current research investigates architectures for accelerating non-deterministic simulation, including the application of reconfigurable logic.

**LEE CORAOR** is an Associate Professor of Computer Science and Engineering at the Pennsylvania State University in University Park, PA. He received his Ph.D. in Electrical Engineering from the University of Iowa. Dr. Coraor has worked on the design, implementation and performance evaluation of decoupled architectures and is currently investigating FPGA architectures and applications.