Report Number: WUCS-87-27

1987-10-02

Load and Communications Balancing on Multiprocessor Logic Simulation Engines

Ken Wong and Mark A. Franklin

The problem considered in this paper is to find an assignment of logic components to processors which will achieve logic simulation speed-ups approaching the ideal for large processor populations. This problem becomes particularly important when a significant portion of the speed-up expected from logic simulation engines is attributed to load sharing (as opposed to obtaining speed-up by employing specialized hardware to carry out specific tasks associated with the simulation process such as event queue manipulation or function evaluation). Our research considers this problem for a particular multiprocessor simulation architecture for which a performance model has bene developed. The model... Read complete abstract on page 2.

Load and Communications Balancing on Multiprocessor Logic Simulation Engines

Ken Wong and Mark A. Franklin

Complete Abstract:

The problem considered in this paper is to find an assignment of logic components to processors which will achieve logic simulation speed-ups approaching the ideal for large processor populations. This problem becomes particularly important when a significant portion of the speed-up expected from logic simulation engines is attributed to load sharing (as opposed to obtaining speed-up by employing specialized hardware to carry out specific tasks associated with the simulation process such as event queue manipulation or function evaluation). Our research considers this problem for a particular multiprocessor simulation architecture for which a performance model has bene developed. The model is parameterized on the basis of workload characteristics, architecture design parameters, and load (processing and communication) distribution. Workload characteristics were derived from actual VLSI circuit simulations. An approach to the component assignment problem is presented and evaluated using this data. The circuits that we are considering have component populations in the tens and hundreds of thousands. Since the optimal component assignment problem is NP-complete, a heuristic assignment is considered and its performance compared against a simple random assignment algorithm (i.e. components are assigned to processors such that each processor is chosen with equal probability). The random assignment method, though simple, is shown analytically to have large communication requirements even for the small number of processors (e.g. five). This approach therefore limits the number of processors which can be effectively applied towards speeding up the simulation process through exploitation of computational parallelism. The heuristic examined in this paper attempts to improve on the random assignment method by reducing the communication volume while maintaining the degree of parallelism in the random method. Results show that for tens of processors and circuit sizes in the tens of thousands, the heuristic can reduce the message volume in the benchmark circuits by a factor of between 2 and 4.

LOAD AND COMMUNICATIONS BALANCING ON MULTIPROCESSOR LOGIC SIMULATION ENGINES

Kenneth F. Wong and Mark A. Franklin

WUCS-87-27

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Presented at the International Workshop on Hardware Accelorators, September 30 - October 2, 1987, University of Oxford, Oxford, England.

This research was sponsored in part by funding from NSF Grant DCR-8417709.

Load and Communications Balancing on Multiprocessor Logic Simulation Engines

Ken Wong and Mark A. Franklin
Computer and Communications Research Center
Washington University
St. Louis, Missouri 63130
USA
(314) 889-6106

The problem considered in this paper is to find an assignment of logic components to processors which will achieve logic simulation speed-ups approaching the ideal for large processor populations. This problem becomes particularly important when a significant portion of the speed-up expected from logic simulation engines is attributed to load sharing (as opposed to obtaining speed-up by employing specialized hardware to carry out specific tasks associated with the simulation process such as event queue manipulation or function evaluation). Our research considers this problem for a particular multiprocessor simulation architecture for which a performance model has been developed. The model is parameterized on the basis of workload characteristics, architecture design parameters, and load (processing and communication) distribution. Workload characteristics were derived from actual VLSI circuit simulations. An approach to the component assignment problem is presented and evaluated using this data.

The circuits that we are considering have component populations in the tens and hundreds of thousands. Since the optimal component assignment problem is NP-complete, a heuristic assignment is considered and its performance compared against a simple random assignment algorithm (i.e., components are assigned to processors such that each processor is chosen with equal probability). The random assignment method, though simple, is shown analytically to have large communication requirements even for a small number of processors (e.g., five). This approach therefore limits the number of processors which can be effectively applied towards speeding up the simulation process through explotiation of computational parallelism. The heuristic examined in this paper attempts to improve on the random assignment method by reducing the communication volume while maintaining the degree of parallelism in the random method. Results show that for tens of processors and circuit sizes in the tens of thousands, the heuristic can reduce the message volume in the benchmark circuits by a factor of between 2 and 4.

* This research was sponsored in part by funding from NSF Grant DCR-8417709.

LOAD AND COMMUNICATIONS BALANCING ON MULTIPROCESSOR LOGIC SIMULATION ENGINES*

KEN WONG and MARK A. FRANKLIN

INTRODUCTION

Simulation plays a prominent role in both validating new VLSI system designs, and in obtaining test vectors for validating the operation of resulting fabricated chips. The high costs associated with such simulations has led to proposals for (Franklin *et al* 1984, Ashok *et al* 1985, Hahn and Fischer 1985) and the development of (Denneau 1982, Howard *et al* 1983, Zycad 1983, Valid 1984, Hefferan *et al* 1985, Silicon 1985, Xcelerated 1986) computers tailored to the logic simulation process. These simulation engines perform simulations at speeds ranging from 10 to 1000 times the speed of standard, general-purpose computers and employ techniques ranging from microcoding of the simulation algorithms to the development of special-purpose multiprocessors.

Our concern in this paper is with a specific type of multiprocessor simulation engine and with the problem of allocating the circuit components to be simulated to the individual processors in the system. An improper allocation can lead to two types of performance degradation. On the one hand an imbalance of computational loads across the processors may lead to processor bottlenecks where one or more processors remain busy for long periods of time while the remainder of the processors are idle. On the other hand an imbalance of communications loads may lead to either inadvertent saturation of the interprocessor communications network, or excessive communications loads for a small subset of the processors. In either case the result will be the creation of a communications bottleneck. Note that the two problems are interrelated in that a circuit component allocation has both an associated computational and communications impact.

This paper considers the component allocation problem for a particular multiprocessor simulation architecture. A performance model of the architecture is given. The model is parameterized on the basis of workload characteristics which were derived from actual VLSI circuit simulations (Chamberlain 1985, Chamberlain and Franklin 1986a, Wong *et al* 1986) and architecture design parameters such as number of processors. Parameters reflecting load and communications balancing are included in the overall performance measure and the impact of improper load and communications balancing is determined. Two approaches to allocating components to processors are presented and evaluated using data collected from simulations of the circuit workload.

The next section presents the architecture considered in this paper. Section 3 develops a run-time performance model for this class of simulators. Section 4 defines elements of the run-time model in terms of a graph model of the circuit to be simulated. A random and a graph-based heuristic component allocation scheme are presented. Section 5 presents the results of an experiment

* This research was sponsored in part by funding from NSF Grant DCR-8417709.

involving several circuits which were simulated. Data collected from these simulations allows us to parameterize the run-time model and compare alternative component allocation schemes. In Section 6 the model and experimental results are applied to the analysis of a particular logic simulation architecture. The final section presents a summary and conclusions.

LOGIC SIMULATION ARCHITECTURES

This paper considers the architecture shown in Figure 1 . A master processor maintains the global time T and signals slave processors through a START signal to begin processing events for the current time point T. When the master receives DONE signals from all slaves, it increments the global time ($k \leftarrow k+1$), and starts the next simulation cycle by sending another START signal to the slaves. Simulation cycles are repeated until the clock reaches the termination time. Note that even when there are no events scheduled for a time point, the simulation cycle is still executed but consists of only the sending of START and DONE signals.

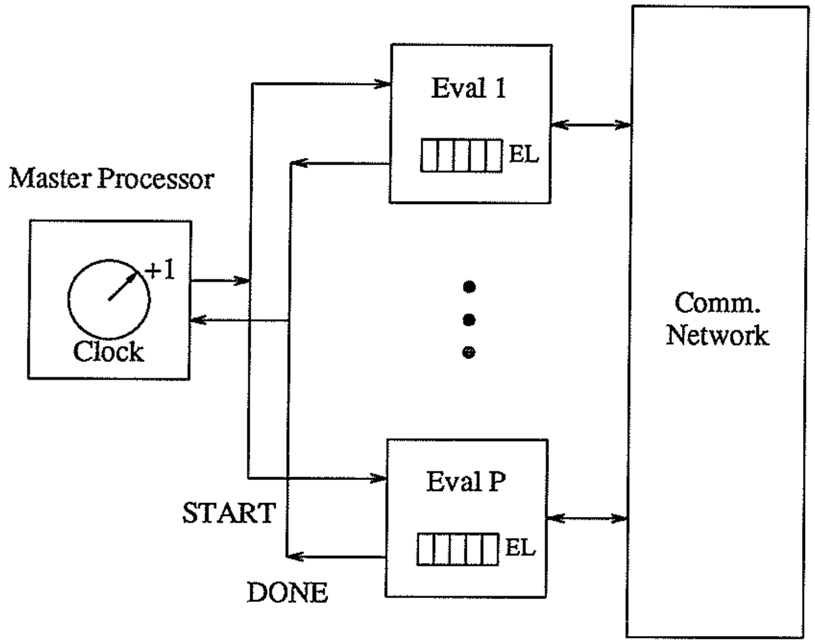

Figure 1. A Multiprocessor Logic Simulator.

Each slave processor evaluates events scheduled for time T, communicates state information to other slave processors over a communication network, and sends a DONE signal to the master processor when it has processed all of its events scheduled for T. Communication buffers between the slave processors and the communication network allow message transmission to proceed in parallel with event/function evaluation.

This general architecture was also analyzed by Levendel *et al* (1983). Our model resembles theirs but considers a broader range of parameters and uses a workload model based on real data. Furthermore, in this paper we specifically explore circuit partitioning strategies which were not considered in their analysis.

THE MODEL

This section describes a simple performance model for the simulation engine in Figure 1. The model can be used to compute upper bounds on architecture performance. The model parameters are first introduced and then the overall model is developed. The partitioning problem will be considered in the context of this performance model.

Model Parameters

The output (dependent) variables, input (independent) variables, and design parameters of the model are shown in Table 1 below. The output variables are our performance measures of interest and are functions of the input variables. Values for the input variables can be obtained from measurements of a conventional, sequential simulator. The design parameters included in this model are the number of processors (P), the effective width of the communication network (degree of communication concurrency) (W), and the times for event/function evaluation (t_E), communication of one event message (t_M), and clock synchronization (t_S, t_D).

Table 1. Variable Definitions.

Var.	Type	Definition
R_P	Output	Simulation run-time using P processors
B	Input	Number of busy ticks
I	Input	Number of idle ticks
E	Input	Number of event/function evaluations
M_∞	Input	Number of messages when $P \rightarrow \infty$
β	Input	Work distribution across processors (β=1: even distribution; β>1: uneven distribution)
P	Design	Number of processors (event/func. evaluators)
W	Design	Average communication width (number of concurrent messages)
A	Design	Component-to-processor allocation
t_E	Design	Average single-event/func. evaluation time
t_M	Design	Single message transmission time
t_S	Design	Time to send a START signal
t_D	Design	Time to send a DONE signal

In a machine with a unit increment clock, the simulation clock runs for B+I simulation ticks or cycles during which at least one event evaluation is performed in each of B (Busy) ticks and none during I (Idle) ticks. There are a total of E event evaluations which are distributed over the B busy ticks and P processors. When there are multiple processors, the number of event evaluations may be distributed unevenly across the P processors resulting in one processor having to perform more event evaluations than the other processors. The most heavily loaded processor has β times more evaluations to perform than the average number of evaluations.

The average event/function evaluation time is t_E. Although some evaluations may take more time than others, we assume that all event/function evaluations take the average event/function evaluation time and that this time is invariant with the number of processors. During the I idle ticks, the system must still spend some time skipping over the idle ticks. This involves the transmission of the START signal by the master processor (t_S) and the processing of the subsequent DONE reply from the slave processors (t_D). The sum of these two times is referred to as the synchronization time, $t_{SYNC} = t_S + t_D$.

A change at the output of a component is an event (representing a signal change) which needs to be propagated to the inputs of each fanout component (those receiving the new signal value).

During the course of the simulation, there are M_∞ such propagations. M_∞ is the maximum number of messages which would be transmitted over the communication network. The propagation of the event will take, on average, t_M time units for each message transmission. Note that for simplicity, we have ignored approaches which transmit multiple events in a single message transmission. The component-to-processor allocation scheme and the circuit characteristics will determine how many of the M_∞ signal propagations will be transmitted over the communication network.

The communications network enters the model in two main ways. First, when there is more than one processor trying to communicate across the network at the same time, only W messages are accepted by the network. Thus, W is the effective width of the network. All other messages are queued until the network can accept more messages. Second, the speed of the network is captured in the parameter t_M which is the mean time required to transmit one message from source to destination processor.

Model Expressions

A simple mean-value model of the run-time of a P-processor system can be derived under the assumption that the ratio of the event/function evaluation time to the communication time is fairly constant over all busy ticks and is given by:

$$R_P = \max\left(\beta \frac{E}{P} t_E, \frac{M_P}{W} t_M\right) + (B+I)t_{SYNC}, \quad P \leq \frac{E}{B} \qquad [1]$$

The term $(B+I)t_{SYNC}$ accounts for those START and DONE signal transmission times which are not overlapped by either event/function evaluation or event-message communication. The term $(Et_E)/P$ represents the total event/function evaluation time when the events are evenly spread over the P processors. The β factor accounts for the degree of load imbalance and will be one when perfectly balanced and P when all events are on one processor. The term $(M_P t_M)/W$ represents the total communication time during the entire simulation. M_P is the number of messages in a P-processor system, t_M is the time required to transmit a single message, and W is the maximum number of parallel message transmissions permitted by the communication network. The max function accounts for the overlap of communication and event/function evaluation assuming that there can be complete overlap and the larger time component in each busy tick is always evaluation or communication.

The model shows that when event/function evaluation is the dominant time component, the run-time decreases as the number of processors P increases (i.e., linear speed-up) assuming β remains constant. When communication is the dominant component, the run-time will depend on the number of message transmissions M_P, the communication concurrency W, and the message transmission time t_M.

Though not pursued here, our model can be extended to account for pipelining both within the slave processors (Wong and Franklin 1987) and the communication network. That is, the simulation algorithm can be divided into stages which must be executed in sequence for each event. Similarly, the communication network may contain stages which a message must traverse to reach its destination. For this study we assume that both the event evaluation pipeline length and the network depth are 1.

THE PARTITIONING PROBLEM

In this section, two partitioning strategies are evaluated with respect to the effects on the simulation run-time. First we discuss the general circuit partitioning problem. Next, a partitioning scheme where components are randomly assigned to processors is analyzed. The message volume associated with random partitioning can be taken as an upper bound for acceptable partitioning strategies. Finally, a more intelligent partitioning scheme based on the idea of aggregating closely connected components on the same processor while maintaining computational balance is examined.

A circuit with C components can be represented as an undirected graph $G = < V,E_c >$ with vertex set V having elements {1,2, ... ,C} and edge set E_c which is a subset of the set of vertex pairs V x V. Verticies correspond to circuit components and edges to connections between components.

A particular allocation of the C circuit components to the set of P processors is represented by a C-by-P *allocation matrix* A with entries a_{ij} which is 1 if vertex i is assigned to processor j and 0 otherwise. The optimization problem of interest is to determine the component allocation (i.e., the elements of A) which minimizes R_P subject to the constraint that each component (or vertex) is assigned to only one processor. While this problem can be solved exactly using branch and bound techniques, problems of this kind have been shown to be NP-hard (Lo 1983) and thus for circuit sizes of interest (i.e., several hundred thousand components) heuristic allocation approaches must be utilized.

In the random partitioning scheme, C/P components are chosen randomly without replacement for each processor. A message arises when the output of a component located on processor i must be propagated to a fanout component located on a different processor j ($i \neq j$) . Since there are C/P components on each processor, the fanout component could be any of the other (C/P) – 1 components on processor i or the C – (C/P) components on the other processors, all with equal probability. Since there are a total of M_∞ signal propagations, the number of transmitted messages is given by:

$$M_P = M_\infty \frac{(C-(C/P))}{(C-1)} \approx M_\infty(1 - 1/P) \qquad [2]$$

Note that the number of messages transmitted across the communication network is 0 when there is one processor and increases with increasing P until it is equal to M_∞ when each component is placed on a separate processor. The above expression can be rewritten as

$$M_P = EFf(P) = EF(1 - 1/P) \qquad [3]$$

where $f(P) = (1 - 1/P)$ is the fraction of messages which must be transmitted over the network, and M_∞=EF.

The partitioning heuristic presented below is based on attempting to allocate components to processors so that two conditions are met. First, the number of components on each processor should be roughly equal to ensure that the computational load on each processor is about equal when events are randomly distributed over the processors. Second, components which are closely connected to each other should be placed on the same processor to reduce the communication volume since closely connected components tend to communicate heavily with each other.

Define the distance D_{ij} between two verticies i and j as the minimum number of edges required to establish a path in G between the two nodes. The heuristic for P processors and the graph $G=<V,E_c>$ is as follows:

1. **Initial Component Allocation:** Choose P vertices such that:
   a) Each vertex is chosen randomly without replacement from V.
   b) The distance between all selected vertex pairs is greater than some D^* (i.e., $D_{ij} > D^*$, for all i,j, $i \neq j$). The selected vertices form the starting or *seed* nodes for the P sets V_p, $1 \leq p \leq P$.

2. **Subsequent Component Allocation:** Add one vertex to each set of vertices $V_{p,}$ $1 \le p \le P$, such that:
   a) Each vertex is chosen without replacement from the set of unassigned vertices.
   b) The vertex is distance 1 from at least one vertex in V_p.
   c) If there is at least one vertex satisfying conditions a and b, choose one vertex according to the vertex selection procedure discussed below. Otherwise, randomly choose an unassigned vertex.
3. **Procedure Iteration:** Repeat step 2 until there are no vertices left to assign.

Denote the set of vertices defined by conditions a and b in step 2 by $U(V_p)$, the set of unassigned vertices with distance 1 from some vertex in V_p. The approach studied in this paper in selecting a vertex to assign to processor p is to randomly select a vertex from $U(V_p)$ in a *breadth-first* manner (i.e., whose distance from the seed node is minimum). The value of M_P will depend on the topology of the circuit being simulated, the number of processors present, the partitioning algorithm and the set of test vectors employed. The next section presents experimentally determined curves for f(P) and thus M_P.

EXPERIMENTAL RESULTS

Data for our model was collected using the *lsim* gate/switch-level logic simulator running on a VAX 11/750[†]. *Lsim* is a UNIX*/C-based simulator which was designed with data collection on the simulation process in mind (Chamberlain 1985, Chamberlain and Franklin 1986a). It can simulate systems containing both traditional unidirectional logic gates, and bidirectional MOS switches. Although *lsim* supports three types of delay models, the data presented in this paper were from the fixed delay model in which component delays are modeled by fixed low-to-high and high-to-low propagation times.

Our benchmark consisted of the four circuits shown in Table 2 (stop watch, associative memory, priority queue, and a radiation treatment planning chip). These circuits reflect a mix of characteristics (Table 3) and are the product of four graduate student design teams. The test circuits were kept small enough to insure that simulation run lengths were reasonable and disk storage availability was adequate. The Switches and Gates columns in Table 3 indicate the number of *lsim* bidirectional switches and unidirectional gates used in defining the circuit. The right column reflects the total number of transistors in each circuit.

† VAX is a trademark of Digital Equipment Corporation.

* UNIX is a trademark of AT&T Bell Laboratories.

Table 3: Circuit Characteristics.

Circuit	Tech.*	Type*	Switches	Gates	Total	Approx. Trans.*
Stop Watch	nmos	sync	216	131	347	650
Assoc Mem	nmos	async	296	454	750	1,700
Priority Q	cmos	sync	2,960	720	3,680	5,100
RTP Chip	nmos	sync	1,422	1,746	3,169	6,100
Average			1,224	763	1,987	3,388

* *Tech*nology, *sync*hronous, *async*hronous, *Approx*imate number of *trans*istors

The partitioning heuristic described earlier was applied to the circuits for processor populations ranging from P=2 to P=128. Then, random test vectors were applied to the circuits using *lsim*, and the message volumes M_P and M_∞ were recorded. Next, the ratio $M(P)/M_\infty$ for different values of P were calculated to obtain data points for estimating a theoretical expression for f(P). Finally, the coefficients of f(P) were obtained using the least-squares method and normalized with respect to the problem size.

Unlike random partitioning, f(P) for heuristic partitioning does depend on the circuit characteristics and the problem size (number of components C). In all four circuits, the following function with circuit dependent constants a_0, a_1, and a_2 fit the heuristic partitioning data well:

$$f(P) = a_0 - \frac{a_1}{P} + a_2 P \qquad [4]$$

Figure 2 shows f(P) for the heuristic (solid curves) and random (dashed curve) partitioning.

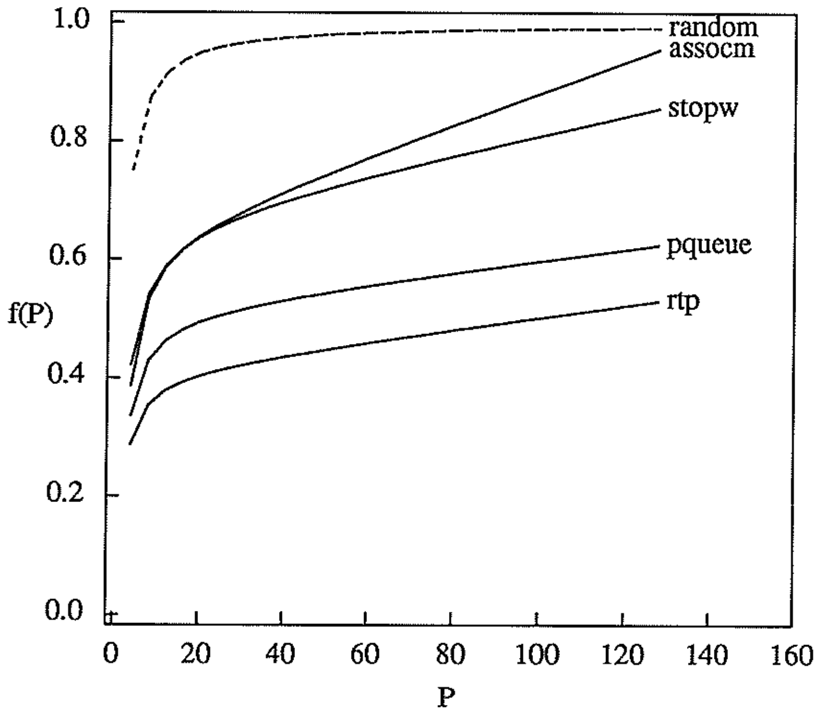

Figure 2. Unnormalized Message Function f(P).

However, the heuristic partitioning curves are for the circuit sizes shown in Table 3 and will be different for different circuit sizes. In order to express f() so that it is independent of circuit size, we rewrite Equation [4] by replacing P by C/C_P where C is the number of components shown in

Table 3 and C_P is the number of components assigned to each processor (=C/P):

$$f(C_P) = b_0 - b_1 C_P + \frac{b_2}{C_P} \qquad [5]$$

where $b_0=a_0$, $b_1=a_1/C$, and $b_2=a_2C$. Table 4 lists the coefficients b_i which were obtained using a least-squares fit and the root-mean-squared (RMS) error for the four circuits.

Table 4. Least-Squares Coefficients.

Circuit	b_0	b_1	b_2	RMS error
Stop Watch	.631	.00254	0.90	.0056
Assoc Mem	.661	.00150	1.19	.0089
Priority Q	.511	.00020	3.44	.0033
RTP Chip	.408	.00016	3.12	.0029
Average	.553	.0011	2.16	.0052

To convert back to the form in [4] where f() is a function of P, the constants a_i are computed from $a_0=b_0$, $a_1=b_1C$, and $a_2=b_2/C$. to illustrate the calculation of f(P) from $f(C_P)$, consider a 40,000 component circuit equivalent to the priority queue. Since C=40,000, b_0=.511, b_1=.00020, and b_2=3.44, we compute $a_0=b_0$=.511, $a_1=b_1C$=8, and $a_2=b_2/C=8.6\text{x}10^{-5}$. So,

$$f(P) = .511 - \frac{8}{P} + 8.6\text{x}10^{-5}P \qquad [6]$$

One note of caution is that the theoretical equations for f(P) can only be applied to processor populations which yield C_P-values which fall in the range of measured values. Furthermore C_P-values at the extreme parts of the range may be suspect. For example, the priority queue has C=3,680 components. Since our measurements were for the range of P=2 to 128, the data is at best valid for the range C_P=C/P=29 to 1840 components. This means that for a circuit which is 10 times as large (36,800 components), the data may be applied to processor populations in the range from 20 to 1,280 at best.

EFFECT OF PARTITIONING ON SPEED-UP

This section illustrates the impact of the heuristic partitioning algorithm on the performance of the simulation engine shown in Figure 1. The effects of the heuristic partitioning algorithm for other architectures can be examined in a similar manner. We consider the priority queue circuit scaled to 40,000 components as an example. An idealized speed-up equation is used to compare the effects of the two partitioning algorithms.

An idealized speed-up S^* can be derived by dividing the single-processor run-time Et_E by the P-processor run-time given in [1]. Since we are focusing on the effect of the partitioning strategy, we omit the synchronization term (which is independent of the partitioning) from the run-time equation [1]. After substituting the expression EFf(P) for M_P, omitting the synchronization time component, and expressing t_M as $t_M=K_Mt_E$, the speed-up equation becomes:

$$S^* = \min\left[\frac{P}{\beta}, \frac{W}{Ff(P)K_M}\right] \qquad [7]$$

K_M is the ratio of the single-message transmission time and the single-event/function evaluation time. The speed-up is idealized in the sense that we have ignored the clock synchronization time and used uniformity assumptions.

We consider the parameter values W=1, β=1.1, K_M=.05, and the f(P) derived in the preceding section. W=1 corresponds to a time-shared bus. K_M=.05 might correspond to a message time of t_M=400 ns and a single-event evaluation time of t_E=8 μs. The average fanout for the priority queue was F=1.5. Curves for idealized speed-up versus number of processors are shown in Figure 3 for the random and heuristic partitioning schemes. The speed-up curve for random partitioning indicates that the bus is already saturated and has limited the speed-up to about 14. Although not shown in the graph, the bus saturates when the number of processors reaches around 13 for random partitioning. However, heuristic partitioning has reduced the message volume so that the bus does not saturate until around 40 processors. The performance of the heuristic partitioning scheme on the RTP chip is very similar to the priority queue since its f(P) function is almost the same.

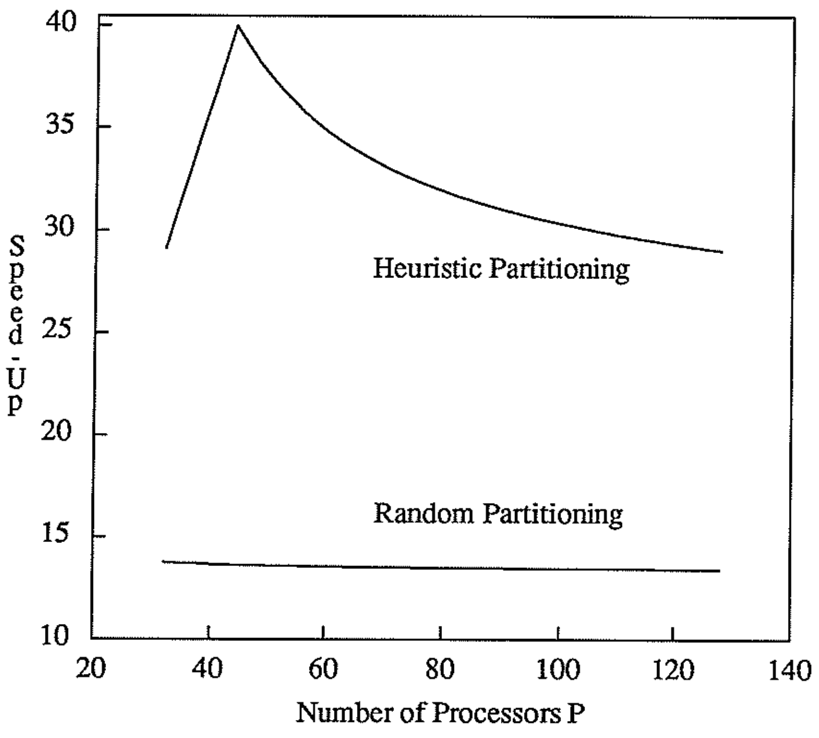

Figure 3. Effect of Partitioning on Speed-Up.

Thus, random partitioning may be appropriate for small processor populations. But the use of hundreds of processors may need better partitioning strategies.

SUMMARY AND CONCLUSIONS

This paper presented a model which could be used to estimate the effect of circuit partitioning on simulation machine performance and presented some preliminary results from some partitioning experiments. A simple heuristic partitioning scheme based on allocating circuit components based on seed vertices separated by a distance D* was compared to random partitioning. Results indicate that for processor populations in the tens and perhaps hundreds, a heuristic partitioning scheme can offer improvements of 2 to 4 over random partitioning if system performance is limited by the communication network.

The heuristic presented here is a simple heuristic which employs very little knowledge of the circuit. We are currently gathering data on other heuristics which are based on knowledge of either the location of registers or user-supplied data. These methods seem to offer yet further reduction in the message volume. We are also developing performance models of logic simulation on hypercube machines and redesigning the *lsim* simulator to handle mixed-mode simulation on a hypercube.

References

Ashok V, Costello R and Sadayappan P 1985 ''Distributed Discrete Event Simulation Using Dataflow,'' *Proc. Int. Conf. on Parallel Processing*, IEEE Comp. Soc. Press, pp. 503-510.

Chamberlain R D 1985 ''Lsim: A Gate-Switch Level Logic Simulator,'' M.S. Thesis, Dept. of Comp. Sci., Washington University, St. Louis, Missouri.

Chamberlain R D and Franklin M A 1986a ''Collecting Data About Logic Simulation,'' *IEEE Trans. on Computer-Aided Design* CAD-5:3, pp. 405-412.

Chamberlain R D and Franklin M A 1986 ''A Unified Approach to Mixed-Mode Simulation,'' Technical Report WUCS-86-20, Dept. Comp. Sci., Washington University, St. Louis, Missouri.

Denneau M M 1982 ''The Yorktown Simulation Engine'' *Proc. 19th Design Automation Conf.*, pp. 55-59.

Franklin M A, Wann D F and Wong K F 1984 ''Parallel Machines and Algorithms for Discrete-Event Simulation,'' *Proc. 1984 Int. Conf. on Parallel Processing*, pp. 449-458.

Hahn W and Fischer K 1985 ''MuSiC: An Event-Flow Computer for Fast Simulation of Digital Systems,'' *Proc. 22nd Design Automation Conf.*, pp. 338-344.

Hefferan P M 1985 *et al*, ''The STE-264 Accelerated Electronic CAD System,'' *Proc. 22nd Design Automation Conf.*, pp. 352-358.

Howard J K, Malm R L and Warren L M 1983 ''Introduction to the IBM Los Gatos Logic Simulation Machine,'' *Proc. IEEE Int. Conf. Comp. Design (ICCD'83)*, pp. 580-583.

Levendel Y H, Menon P R and Patel S H, ''Special-Purpose Computer for Logic Simulation Using Distributed Processing,'' *The Bell System Technical Journal*, Dec. 1983, pp. 2873-2909.

Lo V M 1983 *Task Assignment in Distributed Systems*, Dept. of Computer Science, University of Illinois, Ph.D. Thesis.

Misra J 1986 ''Distributed Discrete-Event Simulation,'' *ACM Computing Surveys* 18:1, pp. 39-65.

Pfister G F 1982 ''The Yorktown Simulation Engine: Introduction,'' *Proc. 19th Design Automation Conf.*, pp. 51-54.

Silicon Solutions Corp. 1985, ''The Mach 1000 Simulation Engine,'' Product Description.

Smith Robert J, II 1986 ''Fundamentals of Parallel Logic Simulation,'' *Proc. 23rd Design Automation Conf.*, pp. 2-12.

Valid Corp. 1984 ''Realfast Simulation Accelerator,'' Product Description.

Wong K F *et al* 1986 ''Statistics on Logic Simulation,'' *Proc. 23rd Design Automation Conference*, pp. 13-19.

Wong K and Franklin M A 1987 ''The Performance Analysis and Design of a Logic Simulation Machine,'' *Proc. 14th Annual Int'l Symp. on Computer Architecture*, pp. 46-55.

Xcelerated Computer Aided Technology Inc. 1986 ''MX and MXT Series,'' Product Description.

Zycad Corp. 1983 ''The Zycad Logic Evaluator,'' Product Description.