Washington University in St. Louis
Washington University Open Scholarship

All Computer Science and Engineering Research | Computer Science and Engineering

Report Number: WUCS-88-08

1988-03-01

Hierarchical Discrete-Event Simulation on Hypercube Architecture

Roger D. Chamberlain and Mark A. Franklin

This paper presents model of hierarchical discrete-event simulation algorithm running on a hypercube architecture. We assume a static allocation of system components to processors in the hypercube. We also assume a global clock algorithm, with an event-based time increment. Following development of the performance model, we describe an application of the model in the area of digital systems simulation. Hierarchical levels included are gate level (NAND, NOR, and NOT gates) and MSI level (multiplexors, shift registers, etc.). Example values (gathered from simulations running on standard von Neumann architectures) are provided at the model inputs to show the effect of... Read complete abstract on page 2.

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

 Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation
Chamberlain, Roger D. and Franklin, Mark A., "Hierarchical Discrete-Event Simulation on Hypercube Architecture" Report Number: WUCS-88-08 (1988). *All Computer Science and Engineering Research.*
https://openscholarship.wustl.edu/cse_research/765

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

This technical report is available at Washington University Open Scholarship: https://openscholarship.wustl.edu/cse_research/765

Hierarchical Discrete-Event Simulation on Hypercube Architecture

Roger D. Chamberlain and Mark A. Franklin

Complete Abstract:

This paper presents model of hierarchical discrete-event simulation algorithm running on a hypercube architecture. We assume a static allocation of system components to processors in the hypercube. We also assume a global clock algorithm, with an event-based time increment. Following development of the performance model, we describe an application of the model in the area of digital systems simulation. Hierarchical levels included are gate level (NAND, NOR, and NOT gates) and MSI level (multiplexors, shift registers, etc.). Example values (gathered from simulations running on standard von Neumann architectures) are provided at the model inputs to show the effect of different model parameters and partitioning strategies on the simulation performance.

HIERARCHICAL DISCRETE-EVENT SIMULATION ON HYPERCUBE ARCHITECTURES

Roger D. Chamberlain and Mark A. Franklin

WUCS-88-08

March 1988

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

Hierarchical Discrete-Event Simulation on Hypercube Architectures

by

Roger D. Chamberlain and Mark A. Franklin

Computer and Communications Research Center
Campus Box 1115
Washington University
St. Louis, Missouri 63130
(314) 889-6106

Abstract:

This paper presents a performance model of a hierarchical discrete-event simulation algorithm running on a hypercube architecture. We assume a static allocation of system components to processors in the hypercube. We also assume a global clock algorithm, with an event-based time increment. Following development of the performance model, we describe an application of the model in the area of digital systems simulation. Hierarchical levels included are gate level (NAND, NOR, and NOT gates) and MSI level (multiplexors, shift registers, etc.). Example values (gathered from simulations running on standard von Neumann architectures) are provided at the model inputs to show the effects of different model parameters and partitioning strategies on the simulation performance.

Keywords:

discrete-event simulation, hypercube architecture

This research was sponsored in part by funding from NSF Grant DCR-8417709 and an NSF Graduate Fellowship.

Hierarchical Discrete-Event Simulation on Hypercube Architectures

Roger D. Chamberlain and Mark A. Franklin

Computer and Communications Research Center
Campus Box 1115
Washington University
St. Louis, Missouri 63130

1. Introduction

Systems to be investigated using discrete-event simulation techniques often can be described at varying levels of complexity and detail. At the highest levels of abstraction, system components are few in number, but each component may embody a complete subsystem that can be quite complex in itself. At lower levels of abstraction, the individual system components are relatively simple in nature, but many components are required to specify the complete system. Simulation of systems that include components at varying levels of abstraction, that is, hierarchically described systems, is the focus of this paper.

A hierarchical system model has several advantages when compared to models that are restricted to only one level of abstraction. Top down design techniques lead naturally to hierarchical descriptions when portions of a design have progressed to different stages in the design process. For example, in digital systems simulation a large circuit might be specified as a set of functional blocks, each of which has a behavioral description. If the design of one of the functional blocks is complete, its behavioral description can be replaced with a lower level description (i.e., at the gate level) in order to verify the design of that functional block. The behavioral descriptions of the other blocks provide the proper environment for the functional block of interest, but cannot be modeled at the gate level if their designs are not yet complete.

This research was sponsored in part by funding from NSF Grant DCR-8417709 and an NSF Graduate Fellowship.

In the cases where a complete description of a system component exists at both high and low levels of abstraction, it is not immediately clear that the low level model is to be preferred for simulation purposes. There are often tradeoffs associated with modeling various system components at different levels, and a hierarchical description allows the tradeoffs to be tailored to varying user needs for information on the performance of different system components. A lower level description often has the property of providing more information than a higher level description but often also has the penalty of increased simulation costs.

The need for high performance discrete event simulation continues to grow, particularly, with the advent of VLSI, in the computer design process, where simulation has become more important and the number of components to be simulated in a typical system has increased dramatically. One approach to providing improved performance is through the use of parallel architectures. Clearly, to take advantage of parallel architectures, parallel versions of the simulation algorithm must be developed. Partitioning the workload associated with the algorithm among a set of interconnected processors can be done in three ways.

1. Across the simulation algorithm: Different subtasks of the algorithm (e.g., event queue management, component evaluation) can be allocated to different processors and the data can be pipelined between the processors.

2. Across the simulated system: The data itself (i.e., system components) can be partitioned among the processors with each processor handling all subtasks of the algorithm for the data in its partition.

3. A combination of the above two methods.

Since there are a small number of subtasks that must be performed in the simulation algorithm, and the systems of interest are large systems, the second approach appears to have the most potential parallelism to be exploited. Franklin et al. [1] analyze the simulation problem and provide a taxonomy of simulation architectures. Wong et al. [2] have shown that considerable parallelism of type 2 above exists in gate-level logic simulation.

We assume a static allocation of system components to processors. That is, a component is allocated to a processor previous to initiating the simulation and does not migrate from one processor to another. We also assume a global clock algorithm, with an event-based time increment. Each processor in the parallel system evaluates all the events that come off its local event queue at a particular simulated time, sends a done signal to a single master processor also indicating when in future simulated time the processor has more work to do, and then waits for a message from the master processor telling how far to advance simulated time. The master processor is responsible for determining (using response information from the other processors) the next point in simulated time that has activity. After this has been determined, the master processor sends a start message to each of the other processors communicating with the start message the next time point to be simulated.

This paper presents a performance model of the discrete-event simulation algorithm running on a hypercube architecture [3]. The hypercube architecture has several features that make it a reasonable target for performing parallel simulation. First, it has high connectivity between the various processors. In a P processor system, a message must traverse no more than $\log_2 P$ links before reaching its destination, and on average will travel less than that distance [4]. Second, it is an MIMD machine, allowing the different processors to implement component models that might be greatly different from one another. This is often the case in hierarchical simulation where system components are modeled at differing levels of abstraction. Third, the hypercube architecture has good scaling properties. As the number of processors grows, the number of links required at each processor increases only logarithmically, and the overall communications capability in the system grows with the number of processors.

This paper is divided into five sections. The section to follow develops a model for the performance of a hierarchical discrete-event simulation algorithm running on a hypercube machine. The third section describes several partitioning strategies for allocating system components in the simulation to processors on the hypercube. The fourth section describes the

application of the model in the area of digital systems simulation, and the final section presents a summary and conclusions.

2. Model Development

This section describes a performance model for hierarchical discrete-event simulation running on a hypercube architecture. The model is based on work by Wong and Franklin [5] that considered a special purpose architecture dedicated to logic simulation. This paper extends their model to hierarchical discrete-event simulation and assumes a hypercube architecture rather than a special purpose simulation machine.

Model variables are listed in Table 1. Along with each variable and its definition, a type is indicated. The variable types are one of "output," "input," "design," and "auxiliary." Output

Table 1. Variable Definitions.

Var.	Type	Definition
R_P	Output	Simulation run time using P processors
C	Input	Number of system components to be simulated
L	Input	Number of levels in hierarchical system description
B	Input	Number of busy ticks in the simulation
E	Input	Number of event/component evaluations
M_∞	Input	Number of messages when $P \rightarrow \infty$
k_l	Input	Fraction of event/component evaluations at level l
α	Input	Work distribution across communications links
β	Input	Work distribution across processors
P	Design	Number of processors
t_{El}	Design	Single event/component evaluation time at level l
t_{CF}	Design	CPU time for single message formulation
t_{CT}	Design	CPU time for single message transmission
t_{CR}	Design	CPU time for single message reception
t_{LM}	Design	Link time for single message transmission
t_{LV}	Design	Link time for single message protocol overhead
M_P	Aux.	Number of messages with P processors
H	Aux.	Average number of hops required per message
W	Aux.	Average communications width
t_{CPU}	Aux.	Total CPU time per busy tick
t_{COMM}	Aux.	Total communications link time per busy tick
t_{EVAL}	Aux.	Total component evaluation time per busy tick
t_E	Aux.	Average single event/component evaluation time
t_{COMCPU}	Aux.	Total communications overhead for CPU per busy tick

variables are the dependent variables, input variables are the independent variables, design variables are design parameters of the architecture, and auxiliary variables are intermediate values (i.e., functions of input and design variables) used to simplify the expressions.

If the simulation runs for B busy ticks (simulation time points that have one or more events that need to be evaluated), the simulation run time for P processors can be expressed as

$$R_P = B \max(t_{CPU}, t_{COMM}) \tag{1}$$

where t_{CPU} is the average processor time per busy tick and t_{COMM} is the average communications link time per busy tick. Communications is assumed to occur concurrently with processing, hence the maximum of processor time or communications time will determine the total run time. The processor time can be further divided into component evaluation time and overhead required to process messages (formulate outgoing messages, handle communications protocols, etc.).

$$t_{CPU} = t_{EVAL} + t_{COMCPU} \tag{2}$$

The component evaluation time can be expressed in terms of the average time to perform a component evaluation, t_E, the average number of events per busy tick per processor, $E/(BP)$, and the computation imbalance factor β.

$$t_{EVAL} = \beta \frac{E}{BP} t_E \tag{3}$$

E is the number of event/component evaluations performed over the entire simulation. The imbalance factor is intended to account for the fact that the computational workload may not be perfectly balanced across the processors at each busy tick. During each busy tick, the most heavily loaded processor takes, on average, β times longer to perform its evaluations that it would if the evaluations were evenly distributed across all processors. Thus, $\beta = 1$ implies a perfect balance and $\beta > 1$ indicates the degree of imbalance. The average time to perform a component evaluation, t_E, can be expressed in terms of the number of levels in the hierarchical description, L, the fraction of component evaluations at each level, k_l, and the time to perform a component evaluation at each level, t_{El}.

$$t_E = \sum_{l=1}^{L} k_l t_{El} \tag{4}$$

At each busy tick, there are M_P/B messages to be transmitted over the communications channels, where M_P is the number of messages with P processors over the entire simulation. The start and done messages used for time synchronization account for an additional $2(P-1)$ messages at each busy tick, giving $\frac{(M_P/B)+2(P-1)}{P}$ as the average number of messages originating at each processor at each busy tick. Each processor receives, on average, the same number of messages from other processors. If each message must traverse, on average, H communications links before it reaches its final destination, $H-1$ intermediate processors must receive and then resend the message, giving $\frac{(M_P/B)+2(P-1)}{P}(H-1)$ as the average number of pass-through messages seen by each processor at each busy tick. Assuming t_{CF}, t_{CT}, and t_{CR} as the CPU time to formulate, transmit, and receive a message, the expression below gives the time spent by each processor at each time point processing communications at each busy tick.

$$t_{COMCPU} = \alpha\left[\frac{\frac{M_P}{B}+2(P-1)}{P}(t_{CF}+t_{CT}+t_{CR}) + \frac{\frac{M_P}{B}+2(P-1)}{P}(H-1)(t_{CT}+t_{CR})\right] \tag{5}$$

Note that a communications imbalance factor α has been introduced to account for an unequal distribution of message volume across processors. This can be rearranged to the following,

$$t_{COMCPU} = \alpha\left[H\left(\frac{M_P}{B}+2P-2\right)\frac{(t_{CT}+t_{CR})}{P} + \left(\frac{M_P}{B}+2P-2\right)\frac{t_{CF}}{P}\right] \tag{6}$$

where the first term in the square brackets, [ ], accounts for the work that must be performed on all messages, either original or pass-through, and the second term accounts for the work that must be performed when a processor originates a message (e.g., formulation of the message itself).

With appropriate substitutions, the processor time equation, (2), can be rewritten as follows.

$$t_{CPU} = \beta\frac{E}{BP}\sum_{l=1}^{L} k_l t_{El} + \alpha\left[H\left(\frac{M_P}{B}+2P-2\right)\frac{(t_{CT}+t_{CR})}{P} + \left(\frac{M_P}{B}+2P-2\right)\frac{t_{CF}}{P}\right] \tag{7}$$

The average communications link time per busy tick is related to the first term in equation (6). The second term in (6) is not included in the link equation because the link does not distinguish between original messages and pass-through messages. The volume of individual message hops (represented by the expression $\alpha H\left\lceil \frac{M_P}{B}+2P-2\right\rceil$) clearly is the same. The CPU times t_{CT} and t_{CR} are replaced by the transmission time t_{LM} and the link overhead for the message protocol t_{LV}. The factor of P in the denominator is replaced by a W, the average communications width, or number of simultaneous messages that can be transmitted concurrently.

$$t_{COMM} = \alpha H\left\lceil \frac{M_P}{B}+2P-2\right\rceil \frac{(t_{LM}+t_{LV})}{W} \tag{8}$$

The communications width, W, is bounded by the number of bidirectional links at each processor, and therefore is less than or equal to $P\lg P$. In existing commercial hypercubes, W is further limited by the memory bandwidth at each processor (i.e., even though there are $\lg P$ incoming message links and $\lg P$ outgoing message links at each processor, the memory bandwidth is not sufficient for each of these $2\lg P$ links to be active simultaneously).

The intermediate variables that have not yet been evaluated are M_P, the number of messages that are needed to perform the simulation on P processors, and H, the average number of hops required by each message. Both of these values are dependent upon the component partitioning and allocation schemes utilized, as well as characteristics of the hypercube interconnection network, and are described in the following section.

3. Partitioning Strategies

Two static component partitioning strategies are considered in this paper. The first, a random partitioning, is used as a benchmark against which other strategies can be compared. The second strategy is heuristic based, and is an attempt to minimize communication volume while maintaining a high degree of computational parallelism. Finding an optimum partitioning is an NP-hard problem [6], and therefore leads to algorithms with exponential cost, not a realistic

alternative for the systems of interest (i.e., those with large numbers of components).

The first partitioning strategy to be considered is random partitioning. In this technique, C components are each randomly placed on one of P processors, yielding an average of C/P components per processor. A message is generated whenever a component on processor i wishes to communicate some information to a component on a different processor j $(i \neq j)$. With C/P components per processor, the component receiving the message could be any of the other $(C/P)-1$ components on processor i or the $C-(C/P)$ components on other processors, all with equal probability. With M_∞ being the number of messages which would occur as $P \rightarrow \infty$, the number of messages for a P processor system is:

$$M_P = M_\infty \frac{(C-C/P)}{C-1} \approx M_\infty(1-1/P), \quad C \gg 1 \tag{9}$$

In this equation the number of messages equals zero when there is only one processor and increases with increasing P until it is equal to M_∞ when each system component is on a separate processor.

Note that on an N-dimensional hypercube $(N = \lg P)$ the number of processors reachable in i hops is given by the binomial coefficient $\left[\begin{matrix} N \\ i \end{matrix}\right]$, with the maximum number of hops required equal to N. The average number of hops that a message (destined for one of $P-1$ other processors) must travel is therefore given by the following expression.

$$H = \frac{1}{P-1} \sum_{i=1}^{N} i \left[\begin{matrix} N \\ i \end{matrix}\right] = \frac{P}{P-1} \frac{N}{2} = \frac{P}{P-1} \frac{\lg P}{2} \tag{10}$$

The second partitioning strategy to be considered is based on a heuristic that attempts to minimize communications requirements while maintaining a high degree of processor load balancing [7]. The system components and their interconnections are first represented by a graph, with each component corresponding to a vertex and an edge connecting two vertices if the corresponding system components communicate with each other during the simulation. A seed vertex is then chosen for each of P partitions such that no two seed vertices are less than some

minimum distance, D, from one another in the graph (i.e, the shortest path from any seed node to any other seed node is of length greater than or equal to D). The partitions are then grown out from the seed vertices in a breadth-first manner, at each iteration adding a vertex to each partition ensuring that the vertex chosen has not already been allocated to some other partition. Once the partitions have been formed, they are randomly allocated to the P processors in the hypercube. The specification of the algorithm used to select the seed vertices is given in Appendix A.

Given the random allocation of partitions to processors, the expression for the average number of hops that a message must travel is still given by equation (10). The improvement gained by using the heuristic is obtained by lowering the message volume M_P.

Unfortunately, the heuristic partitioning algorithm does not lend itself to a simple expression for M_P, so we therefore resort to more pragmatic modeling methods. In our experience [7], the following expression has been successful in matching experimental results,

$$M_P = M_\infty (a_0 - \frac{a_1}{P} + a_2 P) \tag{11}$$

where the constants a_0, a_1, and a_2 depend upon the connectivity of the system under simulation. In the example that follows, the values for the constants will be determined by measuring communications volume after partitioning a system using the heuristic described above.

4. Digital Systems Simulation Example

This section describes the application of the performance model developed above to digital systems simulation. For this example, we will consider the system components to be at one of two hierarchical levels. The lower level is the gate/switch level (e.g., system components such as NAND, NOR, NOT gates, pass transistors, etc.) and the higher level is the MSI level (e.g., system components such as multiplexors, shift registers, counters, etc.). Example values for the input variables (gathered from simulations of representative circuits running on a standard von Neumann architecture [8]) are provided at the model inputs to show the effects of different design

parameters and partitioning strategies on the simulation performance. These results are given as a set of curves (Figures 1 - 3) showing the predicted speedup over a single processor implementation, R_1/R_P, versus the number of processors, P.

The circuits used to illustrate the use of the performance model are a 330 component hierarchical description of a stopwatch chip scaled to 33,000 components, a 1200 component description of a priority queue chip scaled to 12,000 components, and a 3100 component radiation treatment planning chip scaled to 31,000 components. The circuits were exercised, using a unit delay timing model with random input and initial state vectors, and the following data was collected:

input parameter	value stopw	value pqueue	value rtp	meaning
B	3,080	1199	5741	busy ticks
E	9,687,100	750,390	2,535,930	event/component evaluations
M_∞	10,108,800	1,294,670	2,593,950	potential message volume
k_1	0.883	0.97	0.998	fraction of evaluations at SSI level
k_2	0.117	0.03	0.002	fraction of evaluations at MSI level
a_0	0.554	0.565	0.366	constants for equation (11)
a_1	0.827	0.778	0.414	when using heuristic partitioning
a_2	0.0043	0.0019	0.0016	

Although not measured directly, for the purposes of this example we will assume the communications and processor imbalance factors, α and β, range from 1.1 to 2.2, increasing linearly with the number of processors.

The number of processors, P, is varied from 2 to 64, and representative values for the other design parameters are given below:

design parm.	value	meaning
t_{E1}	150 µs	evaluation time at gate/switch level
t_{E2}	450 µs	evaluation time at MSI level
t_{CF}	40 µs	CPU message formulation time
t_{CT}	20 µs	CPU message transmission time
t_{CR}	20 µs	CPU message reception time
t_{LM}	32 µs	Link message transmission time [9]
t_{LV}	320 µs	Link message protocol overhead time [10]

Figure 1 gives the stopwatch speedup for both the random partitioning case and the heuristic partitioning case under three different sets of design parameters. The middle pair of curves uses the design parameters given in the table above, the top pair of curves results when the cost of communications is decreased by a factor of four, and the bottom pair of curves results when the cost of component evaluation is decreased by a factor of three. In all three cases, the random partitioning curve is the lower of the two, as would be expected. Figure 2 provides the same set of curves for the priority queue, and Figure 3 privides the same set of curves for the RTP chip.

There are several things to note about the curves. First, as communications cost goes down relative to CPU cost, the speedup is improved. This is due to the fact that the simulation run time is now due primarily to processor workload and can therefore more closely approximate an ideal linear speedup. Achieving linear speedup is prevented by the processor imbalance factor, β, being greater than one. Second, as communications cost goes up relative to CPU cost, the relative difference between random and heuristic partitioning increases. This is attributable to the fact that when communications costs are a significant contributor to the simulation run time, the lower message volume experienced with the heuristic partitioning algorithm can be of benefit. This is especially true for low processor populations, where the heuristic seems to work quite well.

5. Summary and Conclusions

This paper has presented an analytical model for the performance of a hierarchical discrete-event simulation on a hypercube architecture. Model inputs include independent variables (determined by the simulation being performed) and design parameters (determined by the technology). The model predicts the run time of a simulation taking into consideration the processor workload due to system component evaluation, processor workload due to required message processing, and communications workload.

An example of the use of the model was given for the application area of digital systems simulation. Example input parameters were gathered from a sequential simulation of a pair of digital circuits, and the predicted speedup over a sequential simulation was plotted. The curves indicated the importance of communications workload relative to communications workload, as well as showed the improvement in speedup due to the heuristic partitioning algorithm.

Topics currently under investigation include the relaxation of several of the assumptions mentioned at the beginning of the paper (e.g., static allocation relaxed to allow migration of system components to different processors during the simulation, and global clock algorithm relaxed to allow different processors to simultaneously be at different points in simulated time) and the use of simulated annealing to partition the system components for allocation onto processors.

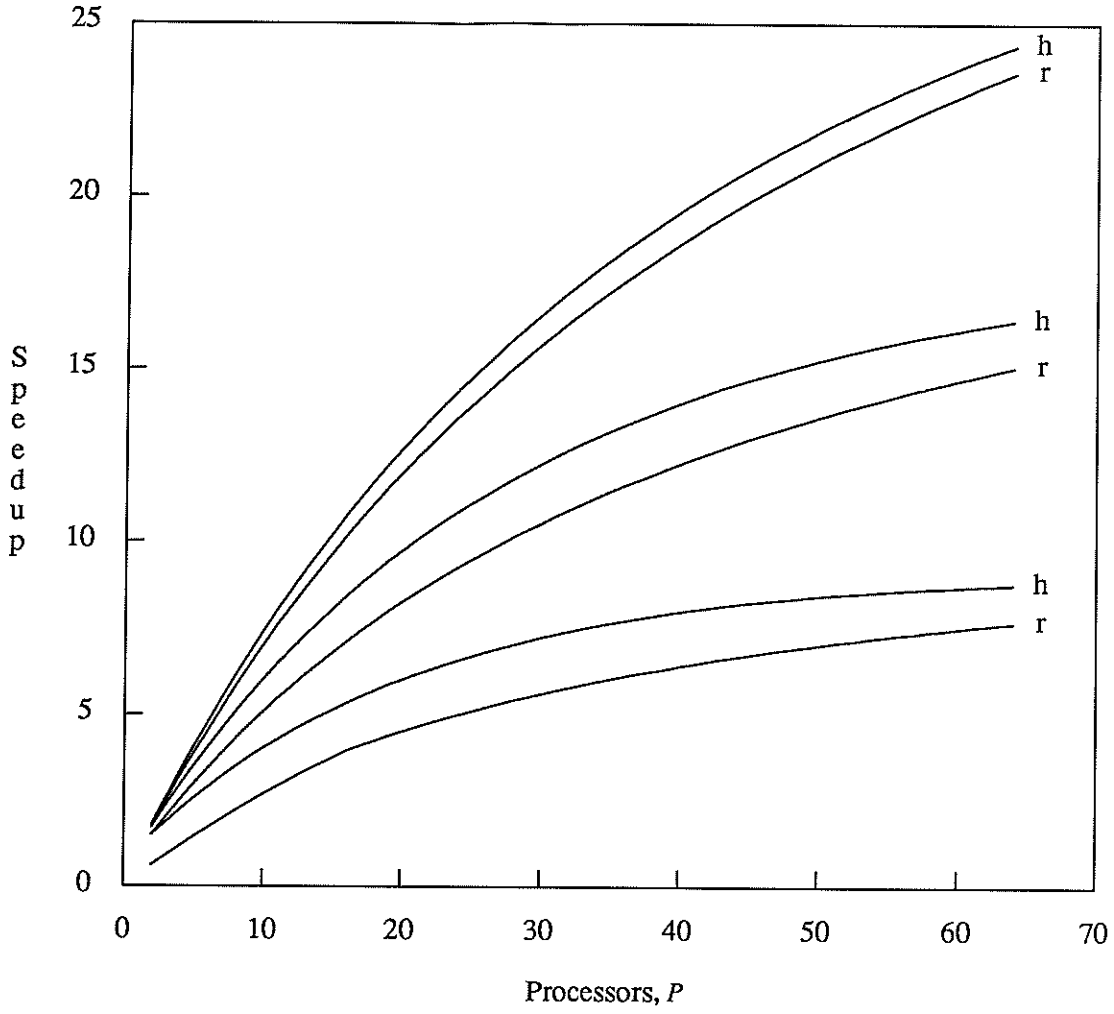

Figure 1. Stopwatch Speedup

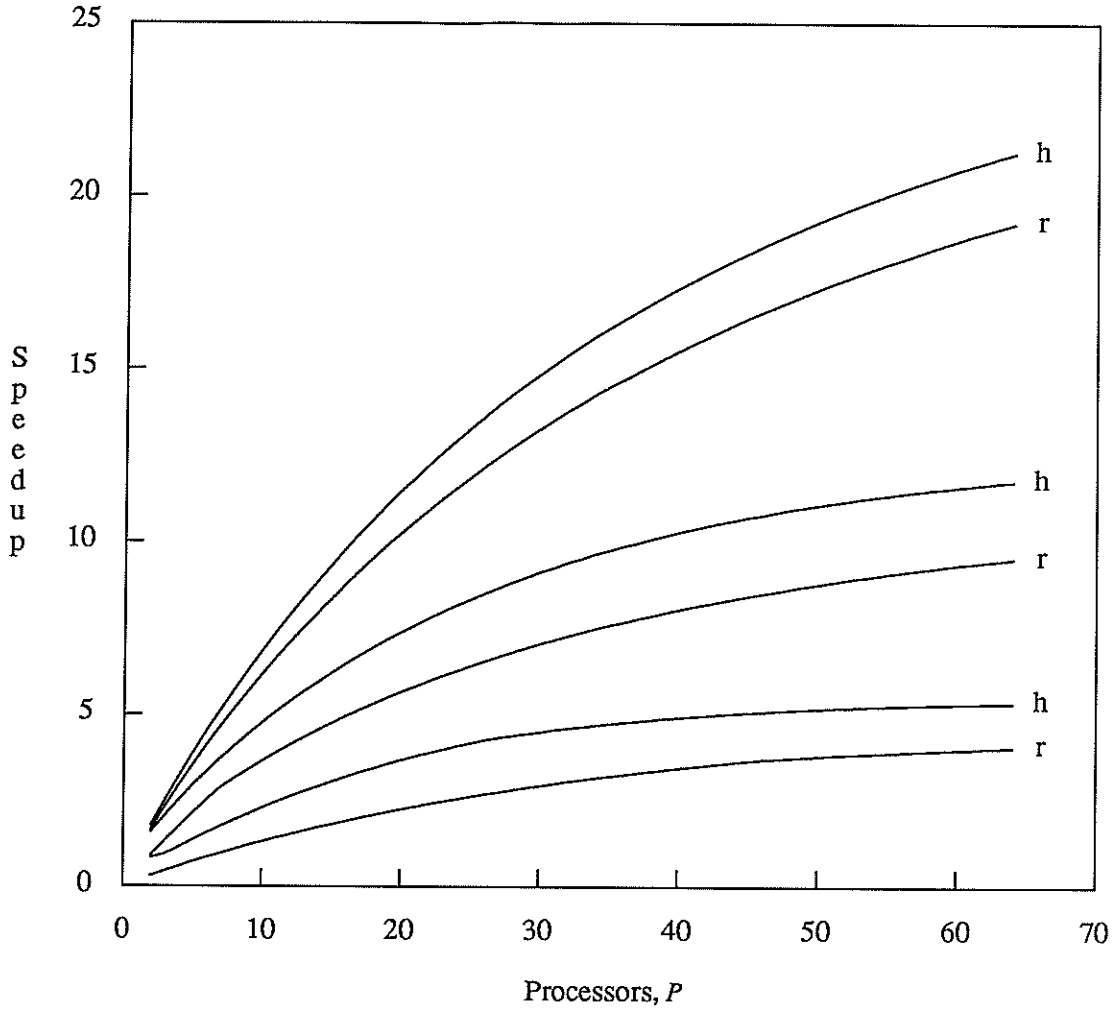

Figure 2. Priority Queue Speedup

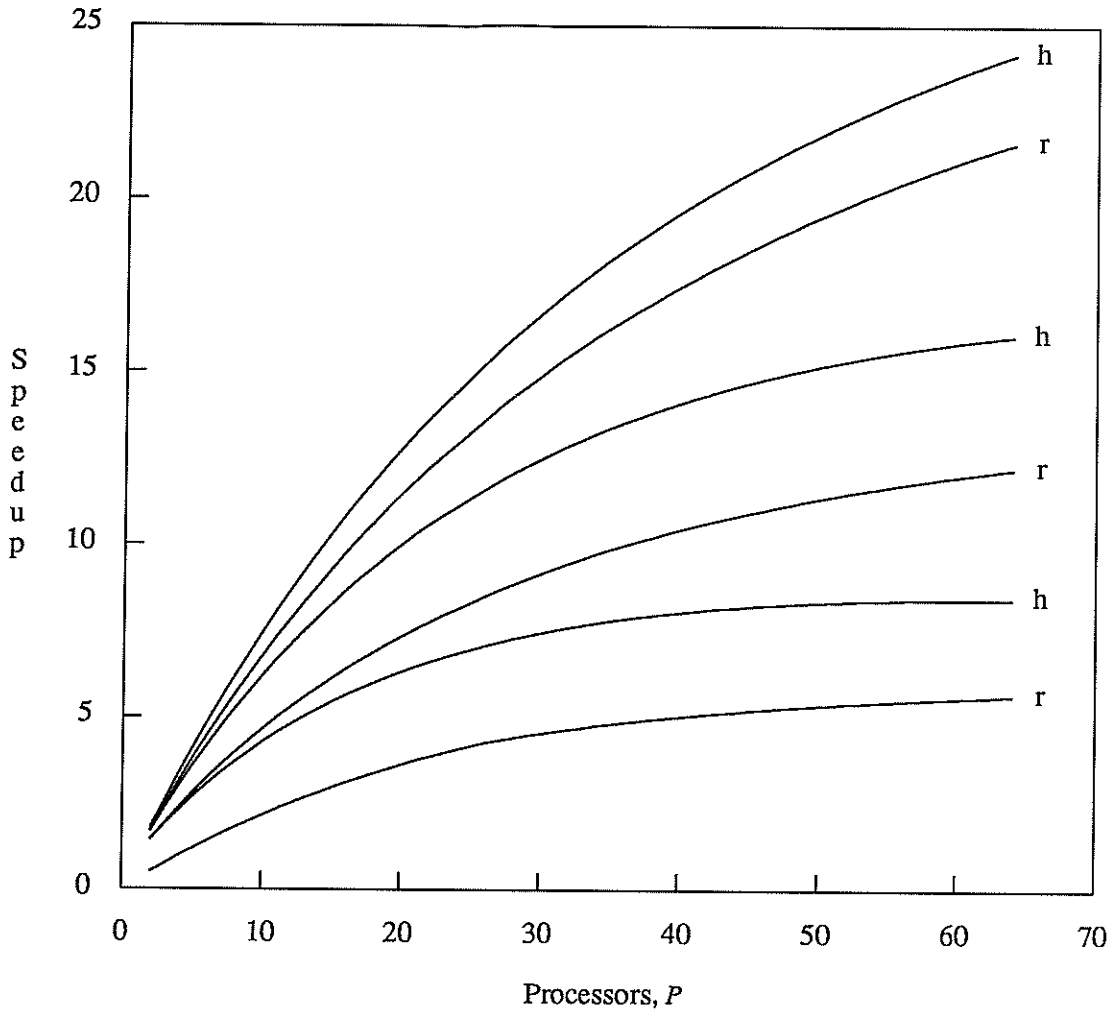

Figure 3. RTP Speedup

Appendix A

Seed Vertex Selection Algorithm

We wish to select P seed vertices such that all P vertices are at least a distance D away from each other. The heuristic begins by choosing the first vertex randomly. Then, given a seed vertex s_i, it moves out in a breadth-first manner marking vertices out to distance D. The next seed vertex is chosen randomly from the perimeter of the set of marked vertices. The *perimeter* consists of those vertices which are at distance D from the marking seed vertex and are marked by only one seed vertex. In the following algorithm, s_i is the i^{th} seed vertex and $M(s_i)$ is the set of vertices marked by s_i.

```
Randomly select s_1;
Assign s_1 to partition 1;
for i:=2 to P do
        M(s_{i-1}) := {s_{i-1}};
        for d:=1 to D do
                M(s_{i-1}) := M(s_{i-1}) ∪ {vertices of distance d from s_{i-1}};
        endfor
        perimeter := marked vertices M(s_{i-1}) with distance D from s_{i-1} not otherwise marked;
        if (perimeter is not empty) then
                s_i := randomly selected vertex from perimeter;
        else
                s_i := randomly selected unassigned vertex;
        endif
        Assign s_i to partition i;
endfor
```

References

[1] Mark A. Franklin, Donald F. Wann, and Kenneth F. Wong, "Parallel Machines and Algorithms for Discrete-Event Simulation," *Proceedings of the 1984 International Conference on Parallel Processing*, August 1984, pp. 449–458.

[2] Kenneth F. Wong, Mark A. Franklin, Roger D. Chamberlain, and Brian L. Shing, "Statistics on Logic Simulation," *Proceedings of the 23rd Design Automation Conference*, Las Vegas, Nevada, July 1986, pp. 13–19.

[3] C. L. Seitz, "The Cosmic Cube," *Communications of the ACM*, Vol. 28, No. 1, January 1985, pp. 22–33.

[4] Youcef Saad and Martin H. Schultz, "Topological Properties of Hypercubes," Research Report YALEU/DCS/RR–389, Dept. of Computer Science, Yale Univ., New Haven, Connecticut, June 1985.

[5] Kenneth F. Wong and Mark A. Franklin, "Performance Analysis and Design of a Logic Simulation Machine," *Proc. of the 14th Annual International Symposium on Computer Architecture*, Pittsburgh, Pennsylvania, June 1987, pp. 46–55.

[6] V. M. Lo, "Task Assignment in Distributed Systems," Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, Champaign, Illinois, 1983.

[7] Kenneth F. Wong and Mark A. Franklin, "Load and Communications Balancing on Multiprocessor Logic Simulation Engines," *Proceedings of the International Workshop in Hardware Accelerators*, Oxford, England, October 1987.

[8] Roger D. Chamberlain and Mark A. Franklin, "Collecting Data About Logic Simulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD–5, No. 3, July 1986, pp. 405–412.

[9] John P. Hayes, Trevor N. Mudge, Quentin F. Stout, Stephen Colley, and John Palmer, "A Microprocessor-based Hypercube Supercomputer," *IEEE Micro*, October 1986, pp. 6–17.

[10] Trevor N. Mudge, G. D. Buzzard, and T. S. Abdel-Rahman, "A High Performance Operating System for the NCUBE," *Hypercube Multiprocessors 1987*, Mike Heath, ed., 1987, pp. 90–99.