Proceedings of the 1991 Winter Simulation Conference
Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)

PARALLEL INSTANCE DISCRETE-EVENT SIMULATION USING A VECTOR UNIPROCESSOR *

James F. Ohi
Bruno R. Preiss

Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario N2L 3G1

ABSTRACT

This paper examines the possibility of running N simulations in parallel on a vector processor. In such a system each instance of execution runs identical code but with a different input data set. The main problem which is addressed is the choice of block selection policy, that is, the choice of which indivisible block of code to execute next. This paper investigates four block selection policies by simulating the execution of such a system. A stochastic flow-graph representation was chosen to model the execution of a simulation. A two-level block selection policy was found to have the best potential speedup of the four block selection policies. The speedup levels achieved were not large, and decreased when there were a large number of unique event types (and therefore handlers) in the simulated system.

keywords - parallel simulation, vector processors, SIMD processors, stochastic modelling.

1 INTRODUCTION

This paper examines the feasibility of using a vector processor to provide speedup for the discrete event simulation (DES) problem. Although the discussion herein is presented in terms of vector processors such as the Cray (Russell 1978), the analysis applies also to single instruction multiple data (SIMD) parallel machines such as the Connection Machine (Hillis 1985). The approach taken is to run multiple instances of the same program simultaneously on a vector processor to achieve speedup. Each instance of the program executes with a different input dataset. The goal is to exploit a vector processor by running N datasets simultaneously in less time than it takes to run N sequential simulations one after another.

During parallel execution of program P, one indivisible block of code is selected to run at a time and all instances scheduled to run that block execute until the block execution is complete, at which point another block is selected. This process is repeated until all instances have reached a termination state. To use such a method, an automated transformation that converts an existing sequential algorithm to a parallel algorithm could be applied.

This method of parallelizing a sequential application was investigated by Rego and Mathur (Rego and Mathur 1990) for general, non-regressive program graphs. In this paper the method of Rego and Mathur is applied to program graphs which model discrete event simulation problems. These graphs tend to be highly regressive and thus direct application of the Rego and Mathur results is not practical.

The primary advantage of the vector approach over autonomous MIMD implementations is simplicity of implementation. An automated transformation technique can directly convert a sequential algorithm into a parallel algorithm in a vector parallel system (Rego and Mathur 1990). Currently this is not practical with MIMD approaches. Additionally, synchronization problems associated with MIMD implementations do not exist since each simulation operates as if on a sequential machine, the only difference being that many simulations are being executed simultaneously.

A disadvantage of the vector parallel approach is that there is only one processor controller for all instances of execution. Although all instances use the same program code, input data differs. It is evident that although the code is identical, each parallel program instance may traverse the code via a different path, depending on the input data. When data dependent computation occurs, the controller must make a decision as to which code segment to execute next, since the instances are no longer executing the same code. This problem is called the code block

*This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

selection problem. The code block selection policy is critical for efficient utilization of the processor. In this study, three block selection policies from the Rego and Mathur study are compared on a DES program graph along with a new, two-level selection policy.

Another possible limitation of the vector implementation is that one must have a good reason to execute N runs of the same program at once. In the case of simulation problems this is not unlikely, since it is often the case that the same experiment is repeated several times using different random seeds. Alternatively, one may simulate a given system in many different scenarios to fully characterize its behaviour. The availability of efficient vector parallel algorithms for simulation execution coupled with automated sequential program transformation makes these possibilities more practical.

1.1 Performance Evaluation

In this section we address the question of determining the potential performance gains associated with the parallel instance technique. Does the level of speedup achieved justify the effort of developing a system which automates this parallel to serial conversion?

One method of approaching this problem is to select a benchmark simulation that is representative of the types of simulations that are to be run. One could then implement a vectorized version of the simulation by hand and observe the resulting speedups on the actual machine. The difficulty comes in selecting a program which is general enough to give useful results which can be applied to a variety of simulation programs.

The approach taken in this paper is to formulate a stochastic model of a representative simulation using a flow graph representation. In this case the difficulty is finding a stochastic model which is truly representative of the problems to be solved. An advantage of this approach is that it easily allows studying the effects of varying simulation structure characteristics. Through these studies it can be determined what types of simulations are expected to run well using this parallel technique.

This paper builds upon Rego and Mathur's research which investigated general, non-regressive stochastic program graphs and dealt with a specific program structure. This paper focuses on the potential parallelism amongst indivisible blocks of code and also models the sequential overhead of the parallel algorithm with specific cases.

This paper is divided into 5 sections. In section 2, the Rego/Mathur model is discussed and their findings are briefly presented. In section 3, a stochastic model of the discrete event simulation problem is presented and specific block selection policies are discussed. In section 4, the test procedure is discussed and results are presented. Following these are conclusions in Section 5.

2 THE REGO/MATHUR MODEL

This section describes a model for representing vector execution of a program graph and outlines Rego and Mathur's basic approach to the block selection problem. The results of their research are then summarized since they have relevance to the discrete event simulation problem.

During the parallel-instance execution of a program P, one indivisible program code block at a time is selected to run and all instances which are scheduled to run that block are allowed to execute in parallel until the block's completion. At this point another block is selected to run and the process is repeated until all program instances terminated.

Rego and Mathur define four block selection techniques:

1. Complete First Policy - select the block with the lowest "index" or sequence number,
2. Move Forward Policy - select the block with highest index,
3. Majority Rules Policy - select the block that the majority of instances wish to execute, and
4. Random - choose a block at random from the list of blocks which need to be executed.

2.1 Definition of Speedup

Rego and Mathur define parallel speedup, γ, as the time required to perform N runs of a simulation over the time required to run N parallel instances of the simulation at the same time on a vector machine. This is shown in Equation (1):

$$\gamma = \frac{N * \sum_{j=1}^{K} m_j t_j}{\sum_{j=1}^{K} n_j t_j \alpha_{j,N}} \quad (1)$$

where N is the machine vector length, m_j is the number of times serial block j is executed using a standard sequential computer, n_j is the number of times parallel block j is executed (although all instances do not necessarily execute it at once), t_j is the time to execute serial block j and K is the number of blocks in the program. $\alpha_{j,N}$, defined in Equation (2), is the block speedup coefficient for block j when executing with a vector length of N.

$$\alpha_{j,N} = \frac{t_i^p}{N * t_i^s} \tag{2}$$

In this equation, t_i^p represents the time it takes to execute the parallel block i using vector parallelism and associated overhead, and t_i^s represents the time to execute block i serially.

To simplify the model, assume block execution time is constant, i.e. $t_j = t$, and $\alpha_{j,N} = \alpha_N$. Then Equation (1) becomes Equation (3):

$$\gamma = \frac{\sum_{j=1}^{K} m_j}{\alpha_N \sum_{j=1}^{K} n_j} \tag{3}$$

Values for α_N were obtained through simulation by Rego and Mathur on an Alliant FX/80 over various block types. The values in this study are obtained from the data given in (Rego and Mathur 1990). Note that $\alpha_N = \frac{1}{N}$ for the case of no parallel block overhead.

2.2 Stochastic Program Execution Representation

Rego and Mathur model a program graph using a stochastic, flow-graph representation. Each node in the graph represents a program block which takes a constant time to execute. Weighted, directed arcs connect nodes in the program graph. The weights of the arcs represent the branching probability of moving from one block to another. The arc weights are dependent on the application and can be obtained from user-supplied branching frequencies or from the results of testing using a large number of input data sets.

This type of model gives an idea of P's *average* transition behaviour. With this model, repeated tests can be performed on the program to acquire statistically valid results without the need to generate a large number of random input data sets for each test case.

Rego and Mathur proved analytically that the complete first block selection policy is optimal for non-regressive graphs, that is, graphs without backward jumps. The program flow graph used for analysis and simulation is shown in Figure 1. They also verified their results through simulation using a constant branch factor $\beta_k = \beta$, $\forall k = 1, 2, ..., K$ with K being the total number of program blocks. Simulation studies also showed that random choice policy performed next best, followed by majority rule and move forward. The fact that majority rule and

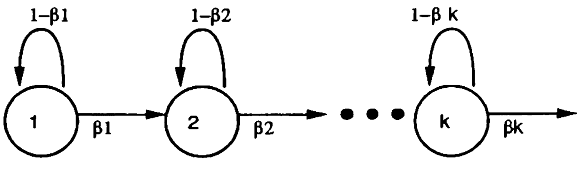

Figure 1: Non-Regressive Graph

move forward performed worse than random would indicate that these policies actually discourage parallelism compared to complete randomness in a non-regressive graph.

Rego and Mathur also repeated these experiments on a regressive graph. In this case majority rules had the best performance followed by random choice, then move forward and finally complete first. Note that the complete first policy which had performed well in the first example had the worst performance in the second example.

Rego and Mathur also demonstrated through simulation that block selection overhead caused significant speedup reduction when the selection policy execution time is significant compared to the execution time of individual blocks. They measured this by setting the overhead to be various percentages of the block execution time and evaluating speedup degradation effects.

3 A STOCHASTIC DES MODEL

This section presents a flow-graph for the discrete event simulation problem and discusses modelling assumptions. The block selection policies used for testing are also introduced.

One of the attractive features of this vectorization technique is in the simple manner by which much of the sequential algorithm can be systematically converted into parallel vectorized code. In light of this fact the most common discrete event simulation sequential algorithm was chosen for these experiments. Figure 2 shows the general structure. All instances start in an event dispatcher which fetches the next event from an event queue and calls the appropriate event handler to process it. The processing of an event leads to the creation of new events until the simulation terminates. This basic approach is discussed in (Misra 1986).

A non-deterministic representation of this algorithm involves modelling the dispatcher as a node with outdegree E. E is the total number of unique event types in the system. As a simplifying assumption, each event type has a uniform probability of occurring. Therefore the weight on each outgoing arc

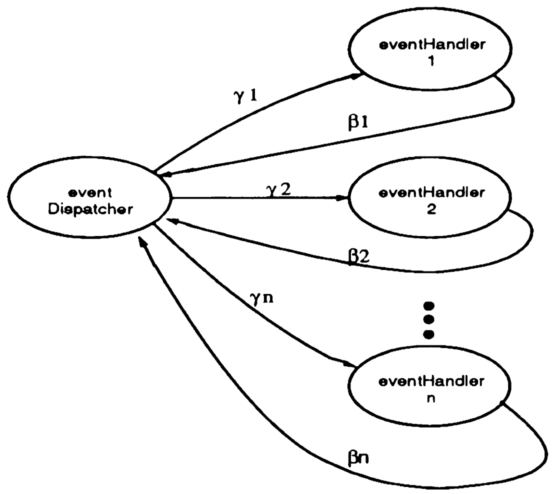

Figure 2: DES-like Flow Graph

is $\frac{1}{E}$.

As can be seen in Figure 2, the stochastic DES flow graph is simple but highly regressive. The simple structure of the DES graph allows for the collection of statistical results which can be used to predict the performance of most DES programs. The results could also apply to a hierarchal simulation structure, where each event branch breaks up into sub-branches representing sub-event types within the event.

In addition to the vector length N, the parameters of the model include the move forward or branch factor, β, which specifies the tendency of event graphs to move forward. Other parameters include the average number of blocks per event, as well as the number of unique events in the system. The length of the simulation run is also a control input affecting the influence of transients in the experiments.

In this study, the subprogram graph within each event is non-regressive. Thus, the simple graph representation shown in Figure 1 is used for performance modelling. More complex graphs could be used to express arbitrary forward conditional branching, but such forms would be difficult to implement in a "general" way and would require more control parameters, complicating the test procedure. Additionally, simulation-specific structures could be introduced into the graph specification. One example of this would be an event queueing routine which is a typical function in simulations. Including these application-specific structures is difficult to do in a general and easily controlled way in stochastic graphs. The graph shown in Figure 1 has only one control parameter to allow adjusting the event graph's tendency to move forward. This model can account for variable length loop structures, and also accounts somewhat for conditional branches, modelling their tendency to jump ahead or stay behind during execution. The emphasis of this paper is on modelling the global structure of simulation programs and studying the effects of various characteristics.

The results from Rego and Mathur's research are not directly applicable to the DES problem since their intention was to model "general" program graphs rather than "general simulation" program graphs. They therefore selected non-regressive graphs for testing to make the problem analytically solvable within a reasonable time.

3.1 Selection Policies

Since we have chosen to model the event sub-graphs in Figure 2 as being non-regressive, we can apply the results from the Rego/Mathur experiments for block selection within each event. Therefore we use the optimal complete first block selection policy while execution is within an event handler. However the overall DES graph is also highly regressive and therefore may not perform well under a global complete first policy.

In order to exploit the optimal use of complete first policy within an event graph, a two-level hierarchical block selection policy is proposed. Event type selection constitutes the first level in the selection hierarchy, while block selection within event subgraphs constitutes the second level. Once a suitable event handler has been selected for execution, the complete first policy is used within the event handler to make block selections until termination of the event. The choice of the best event selection policy is not intuitively nor analytically obvious. Rego and Mathur showed through simulation that the majority rule selection policy appeared to perform comparatively well for regressive graphs and thus it will be used in the two-level scheme.

The disadvantage of two-level block selection policies compared to single-level ones is that synchronization of all instances at the event level must occur before the next event type can be chosen. This means all instances must wait for each other to complete the event subgraph before they can go on. This incurs a loss of processor utilization since some processors, due to their input streams, may finish event processing long before others do. However this can also be considered an advantage over the other selection policies, since it can be postulated that single level selection policies being applied on regressive graphs do not encourage occasional "synchronization" of the instances. Therefore instances can potentially become "spread out" over the entire program graph and have little temporal block synchronization resulting in a loss of parallelism.

A comparison of the two-level block selection technique's average performance compared to single level approaches will be shown, as well as the speedup obtained against an equivalent sequential implementation. One point to note is that the complete first algorithm is not easily applied to DES graphs since it relies on selecting the lowest block index numbers to make a selection for the next block to execute. Numbering of the blocks in a regressive graph is somewhat arbitrary since the set of indexes of one event branch is arbitrarily chosen to have lower index numbers than that of another branch. In this study event 0 has block numbers 0 to $K[0]-1$, while event 1 has block numbers $K[0]$ to $K[1]-1$ and so forth, where $K[i]$ represents the number of blocks in event i.

A simplifying assumption in this model is that all blocks have the same execution time. This is not true in reality, but it is assumed it will not affect results to a large degree.

The accuracy of this stochastic model of discrete event simulations is questionable at best due to the simplifying assumptions in the program graph. Nevertheless it provides an upper bound on the speedups that could be achieved in a real system, as well as giving insight into the effects varying graph parameters. The one advantage of using this abstract modelling technique is that it can be easily modified to cover a larger spectrum of cases.

4 TEST PROCEDURE AND RESULTS

An investigation was performed into the effects of four different parameters on speedup. The parameters of interest were the vector length N, the number of unique event classes E, the average number of blocks per event B and the branch factor β. In addition, tests were performed to determine the impact of simulation startup and termination transients. These effects are present because the termination criterion for the simulation was to terminate when all of the simulation instances had executed a given number of blocks. Using this technique, some instances which were lucky enough to have their blocks selected more often than others terminated much earlier, resulting in a decrease in utilization since a vector component was left idle. Tests were performed to determine the minimum simulation length in order to minimize startup and termination effects since this project seeks steady state results.

Graphs which are plotted with respect to N account for sequential overhead (using α_N) while the other graphs only show block-level speedup, which is valid for comparing the relative performance of the block selection policies.

Instead of specifying exact values for graph characteristics during test runs, average or range values were specified and a random DES graph was generated based on these values. It was then executed using each of the block selection policies and the speedup was calculated using Equation (3).

During the flow graph generation phase, a graph was randomly created according to command-line supplied specifications. The number of blocks in each event was generated using a normal distribution with a specified variance.

During simulation, event-types are generated using a uniform distribution. For purposes of generality, it is assumed that the choice of the next event generated is independent of the previous events processed. Block transition probabilities within events also use a uniform variate. Once again this assumes that the probability of successful block transitions is independent of previous block transitions and also independent of event type.

4.1 The Effect of E

Intuitively, increasing the number of unique event types will reduce parallelism for all policies since it serves to spread out all of the instances over more event subgraphs. Consequently, they have less chance of occupying the same event as another instance. Figure 3 shows results with $\beta = 0.5$, $B = 40$ and $N = 100$.

As can be seen in Figure 3, all four policies exhibit the same behaviour. For low E (< 20), the two-level scheme exhibits a significant speedup advantage over the single level schemes. However for large E, the differences between all of policies is very small although the two-level scheme does show a negligible advantage. Clearly for large E the speedup for all of the policies is not large (around 2).

4.2 The Effect of B

B represents the average number of blocks in an event subgraph. Recalling that the complete first policy was optimal for non-regressive graphs (Rego and Mathur 1990), it is expected that as the number of blocks per event increases relative to the number of events, the speedup associated with the complete first policy would improve. This is because the graph begins to look less regressive due to the presence of lengthy, non-regressive chains. Accordingly, the performance of the random and majority-rules policies should deteriorate with increasing number of blocks per event since the graph looks more and more like a non-regressive graph. For very low numbers of blocks

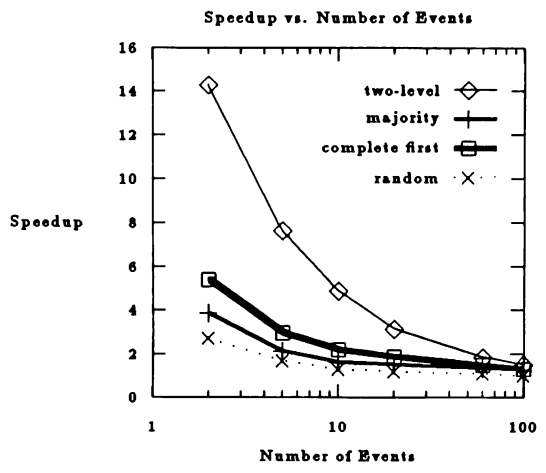

Figure 3: Log Graph $\beta = 0.5, B = 40, N = 100$

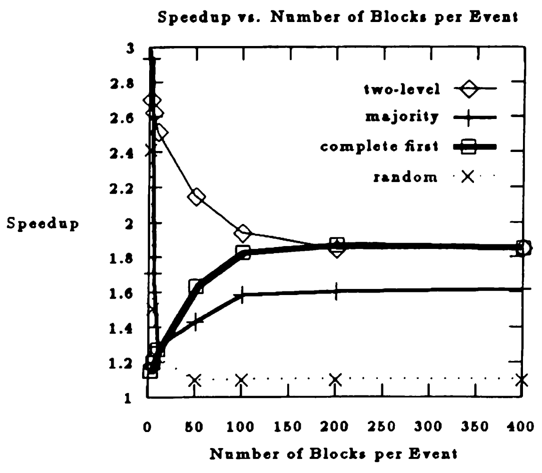

Figure 5: $\beta = 0.5, E = 40, N = 100$

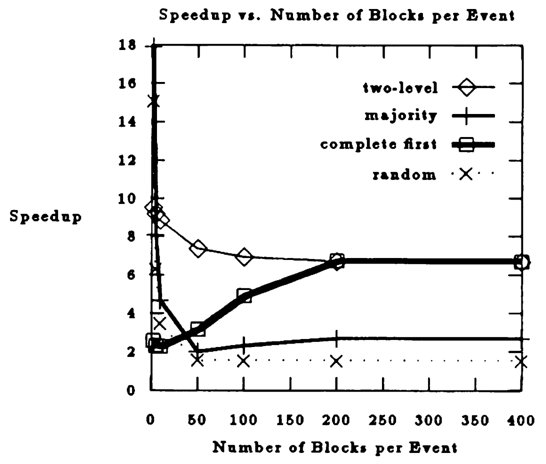

Figure 4: $\beta = 0.5, E = 5, N = 100$

per event, the graph appears highly regressive with very little non-regressive content. Thus, it is expected that the majority-rules and random choice policies should perform better than complete-first.

The two-level selection policy is expected to work well in both situations since it combines the complete-first and majority-rules policies in one. Figures 4 and 5 show the expected behaviour. Figure 4 shows results with moderate β ($\beta = 0.5$), a small number of event types ($E = 5$) and $N = 100$. Figure 5 shows a similar case but with a large number of event types ($E = 40$).

The crossover point where complete first overtakes majority rules and random selection lies between 30 and 50 blocks per event. This crossover position is affected by the number of unique event types in the system. The two-level selection policy always performs well. When B is high (> 100 blocks), complete first and the two level policy had almost equal speedup, depending on the number of event types in the system. In summary, these effects are observed because it appears that the ratio of B to E affects the degree to which a graph appears non-regressive or regressive.

4.3 The Effect of β

β represents the tendency for the program to move forward in the program flow graph. The parameter β was varied between 0.1 and 0.9 to characterize its effect on overall speedup and the results are shown in Figures 6 and 7. Figure 6 shows results with $B = 40$, $E = 5$ and $N = 100$. Figure 7 shows results with larger numbers of event types and blocks per event ($B = 100$ and $E = 10$) to see the difference. As can be seen in the graphs, the two-level block selection policy had the best speedup and increases monotonically with increasing β. It is believed that this performance is due to the fact that two-level policy enforces synchronization of the instances between events, improving parallelism.

The single level policies do not perform the periodic synchronization that the two level approach performs and therefore do not encourage parallelism. As a result, the speedup achieved is lower. Only the complete first policy forces synchronization because instances with high index blocks wait for instances with low index blocks. However, complete first does not consider which of the events has the most instances and therefore does not maximize parallelism.

Speedup generally decreases with decreasing β for all policies except for complete first. A small β yields programs that take a longer time to execute since they have a smaller tendency to move forward. This causes increased conflict rather than synchronization

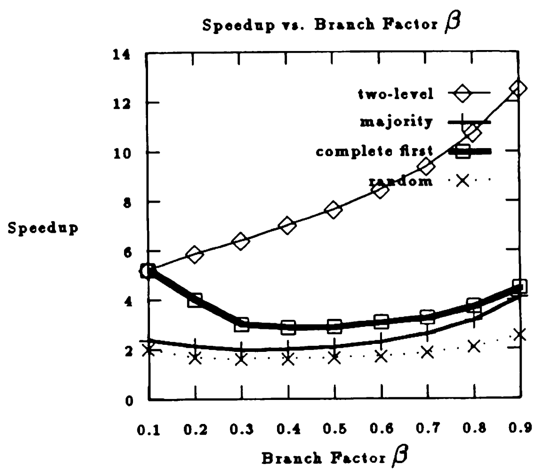

Figure 6: $B = 40, E = 5, N = 100$

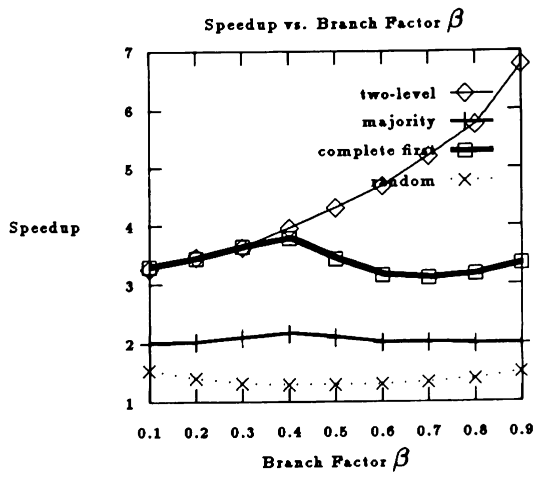

Figure 7: $B = 100, E = 10, N = 100$

and therefore reduces parallelism.

As β increases, the probability for instances to move through blocks in events *together* increases causing the slight increase parallelism and speedup. The two-level approach is preferable to the others when β is high.

An interesting property emerges in Figure 7. When B is high ($= 100$), the complete first policy speedup improves. It appears that the two level policy seems to serve as an upper bound for the complete first policy speedup for very large B. The complete first plot therefore will look more and more like the two level plot as B increases.

4.4 The Effect of N

N represents the vector length of the vector processor being used. Estimates of the overhead associated with the parallel implementation have been obtained from the results in Rego and Mathur (1990). In all cases, the two-level block selection policies have the highest relative speedup compared to the single-level schemes. However, the absolute speedup is not always significant.

Figure 8 shows an optimistic speedup graph with parameter values selected so that the two-level block selection policy performs well. The parameters are $\beta = 0.5$, $B = 40$ and $E = 2$. Referring to Figure 3, it can be seen that speedup decreases rapidly until about 20 unique events, after which it levels off with speedups in the range of 3 – 4. Figure 9 shows speedup in the lower part of that graph with $E = 40$.

Figure 10 shows the case where B is high. As in the case of Figure 7, the complete first policy plot looks more and more like the two-level plot for very large B.

Figure 11 shows a scenario in which the two level policy performs poorly, specifically, $\beta = 0.3$ (low), $B = 100$ (high) and $E = 30$ (relatively high). As can be seen speedups are lower and the complete first policy almost has the same speedup as the two level approach.

5 CONCLUSIONS

Clearly the accuracy of the model limits the scope over which any conclusions can be made. In order to generalize the conclusions to real simulation programs in general, a validity test would be necessary to prove the model.

Assumptions were made to simplify the model and make it controllable through parameters for study. As discussed in previous sections, the speedups calculated can be thought of as theoretical upper bounds

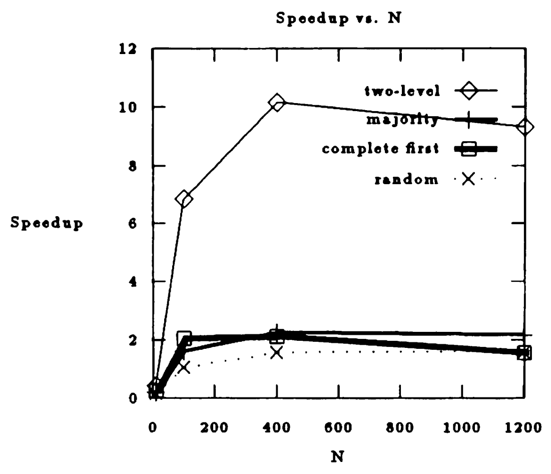

Figure 8: $\beta = 0.5, N = 40, E = 2$

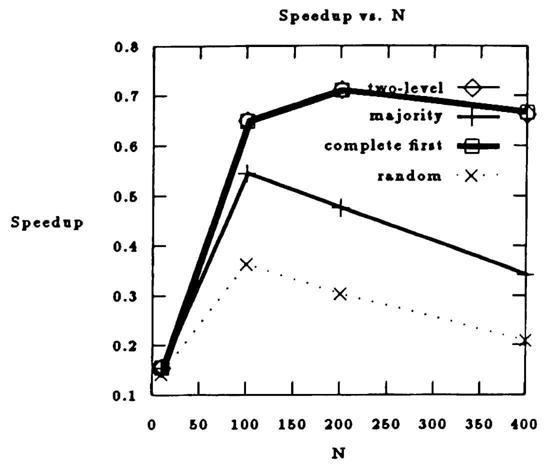

Figure 11: β = 0.3 (low), B = 100 (high), E = 30 (relatively high)

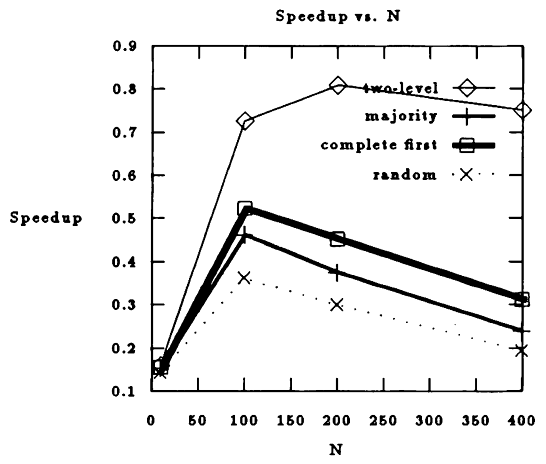

Figure 9: $\beta = 0.5, N = 40, E = 40$

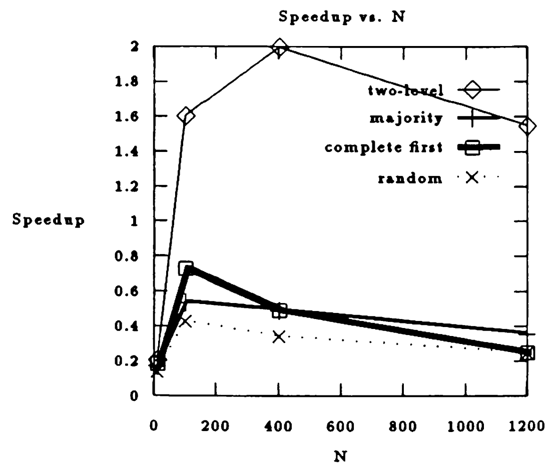

Figure 12: $\beta = 30, B = 40, E = 10$

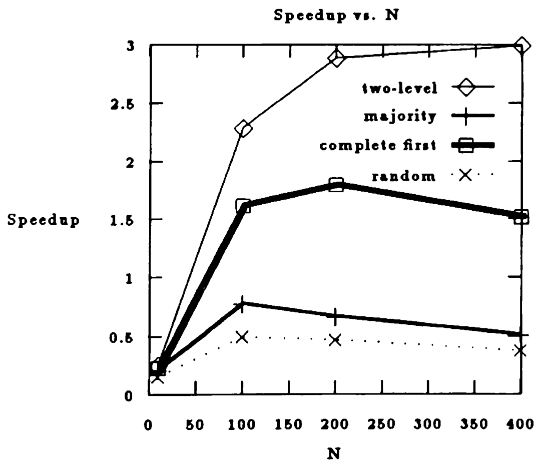

Figure 10: High B

since they omit certain details of an actual implementation but rather concentrate on the global structure. It is also clear that for graph structures different from the simulation graph structure presented here, the results will change in a manner that is not easily predictable.

- The single-level approaches described in Rego and Mathur's research are not in general suitable for the DES flow graph model presented here. This is because DES flow graphs tend to have highly-regressive parts as well as non-regressive parts.
- The two-level block selection policy proved to perform relatively better than the single-level strategies under the conditions in this project. Therefore given a program with a comparable structure to that in Figure 2, the two-level strategy is the method of choice.
- When B is high, the complete first policy performs comparably to the two-level policy.
- Speedup using the two-level selection policy increases with β.
- Speedup with all policies monotonically decreases with increasing E.
- Speedup with all policies except complete first decreases with increasing B. The two-level strategy always has the highest speedup, but with large B, complete first is comparable. This is because with large B, the graph appears less regressive.
- Improved speedup occurred with increasing N until around $N = 150$, at which point the speedup improvements became very small.
- Overall, the speedup is not large. This is partially attributed to the limitations of a single controller vector processor. As seen in the graphs which included overhead, speedup levels do not exceed 4 on a 100 processor system for the cases presented. The usefulness of this technique is dependent on the "need for speed" versus the expense. However, Rego and Mathur demonstrate that conversion from serial to parallel vector algorithms using this technique can be automated and thus is seductively simple.

REFERENCES

Hillis, W. Daniel, *The Connection Machine*, The MIT Press, 1985.

Misra, J., Distributed Discrete-Event Simulation. *Computing Surveys*, Vol. 18., No.1, March 1986

Rego, Vernon and A.P. Mathur, Concurrency Enhancement through Program Unification: A Performance Analysis, *Journal of Parallel and Distributed Comp.*, Vol. 8.,1990, pp.201-207.

Rego, Vernon and A.P. Mathur, Exploiting Parallelism Across Program Execution: A Unification Technique and Its Analysis, *IEEE Trans. on Parallel and Distributed Systems*, Vol. 1, No. 4., October 1990, pp.399-414.

Russell, R.M., The CRAY-1 Computer System, *Communications of the ACM*, Volume 21, Number 1, January 1978, pp. 63-72.

AUTHOR BIOGRAPHIES

JAMES F. OHI is a graduate student at the University of Waterloo, Waterloo, Ontario, Canada, where he completed a B.A.Sc degree in Computer Engineering in 1990. His research interests include parallel discrete-event simulation, system modelling and parallel architectures. He is a member of the Institute of Electrical and Electronics Engineers.

BRUNO R. PREISS received the B.A.Sc degree in Engineering Science (Electrical Engineering Option) in 1982, the M.A.Sc degree in Electrical Engineering in 1984, and the Ph.D. degree in Electrical Engineering from the University of Toronto, Toronto, Ontario, Canada. He is an Assistant Professor in the Department of Electrical and Computer Engineering at the University of Waterloo, Waterloo, Ontario, Canada. He is a member of the Computer Communications Networks Group and the VLSI Research Group. His current research interests include parallel discrete-event simulation and multiprocessor and parallel processor computer architectures.

Bruno R. Preiss is licenced as a Professional Engineer in the Province of Ontario, Canada. He is a member of the Canadian Society for Electrical and Computer Engineering, the Institute of Electrical and Electronics Engineers, and the Association for Computing Machinery.