Proceedings of the 1995 Winter Simulation Conference
ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman

ESTIMATING THE BENEFIT OF THE PARALLELISATION OF DISCRETE EVENT SIMULATION

Simon J.E. Taylor
Farshad Fatin

Dept. of Computer Science and Information Systems
at St. John's
Brunel University, West London
Uxbridge, Middx UB8 3PH, U.K.

Thierry Delaitre

Centre for Parallel Computing
University of Westminster
115 New Cavendish Street
London W1M 8JS, U.K.

ABSTRACT

This paper presents a technique which attempts to aid the simulationist in the decision as to whether or not a simulation should be implemented on a multiprocessing computer. The proposed technique has been used to estimate the performance of parallel discrete event simulations. This employs *critical path analysis* to determine the lower bound of the execution time of a parallelised simulation and has been used by other authors to study the effect that process scheduling and causality maintenance protocols have on performance. The contribution of this paper is the extension of this technique to a class of common simulation models which demand resource sharing. This forces a parallel implementation to use some kind of information exchange protocol. A case study is presented which illustrates the potential usefulness of this technique. This employs a performance analysis tool developed for this purpose which has been implemented using the simulation package SES/Workbench.

1 INTRODUCTION

The techniques used to implement a discrete event simulation on a multiprocessing computer form the field of Parallel Discrete Event Simulation (PDES). The dual reasons for parallelising a discrete event simulation are reduced execution time (by sharing work across many processors) and increased problem size (in terms of memory required and completion time). In these terms, reports of many simulations that have been successfully implemented can be found in Fujimoto (1993). The same paper indicates that success cannot be guaranteed due to the many complex considerations of PDES. This lack of guarantee makes it difficult for a user to commit to what could be a high expenditure in terms of hardware purchase and software development.

Ultimately, the potential success of parallelising a simulation depends on the amount of parallelism, in terms of causal and data dependencies, within the model being simulated. It would be advantageous to be able to predict the potential speedup that could be expected by implementing the simulation on a multiprocessing computer. The question is, "*Is the analysis of model parallelism enough to justify the decision to parallelise?*"

This is then the ultimate goal of work being performed by the authors of this paper. This paper presents the current state of work on the approach being taken to analyse the parallelism of simulation models. This focuses on simulations which model physical systems that share resources. An artificial case study is presented to illustrate these techniques.

It will be assumed that the reader is familiar with the general approaches used in PDES. Excellent introductions to the field can be found in Fujimoto (1993) and Nicol and Fujimoto (1993).

2 PARALLELISM ANALYSIS OF PDES

A useful technique to analyse the parallelism of a model is *critical path analysis* (CPA). This technique has been used to determine a lower bound on the completion time of a PDES (Berry and Jefferson 1985, Livny 1985, Berry 1986, Jefferson and Reiher 1991, Lin 1992 and Srinivasan and Reynolds 1993). All apart from Lin (1992) have studied the effect that the ordering protocols used in PDES have on the lower bound completion time. Lin (1992) has also used CPA to study the effect that LP scheduling policies have on performance. For a complete derivation of the CPA method, the reader should consult the treatment given in Lin (1992) or Taylor (1993).

Operationally, the CPA technique performs a simulation of a PDES and records each LP's progress through time. In a PDES, let event e be scheduled at LP_i. Let $e.\alpha$ be the real time that event e is scheduled

at LP_i (the time that the timestamped message scheduling event e arrives at LP_i) and T_i be the real time that a process LP_i has reached. Let $e.E$ be the set of events scheduled due to the execution of e and let $\delta(e)$ be the time taken for a timestamped event message representing an event in the set $e.E$ to be sent from LP_i to another LP. Figure 1 shows the algorithm to find the critical path execution time. The calculation in line 9 updates the real time clock T_i of LP_i to the maximum of the current value of T_i and $e.\alpha$ plus the real execution time of event e, $\eta(e)$. The calculation in line 12 determines the time at which event messages generated by LP_i arrive at their destination LPs. The calculation in line 14 gives the equivalent sequential execution time.

```
/* initialisation */
1.  FOR ALL i DO
2.     T_i := 0
3.  ENDFOR;
4.  FOR ALL e pre-scheduled in the event list DO
5.     e.α := 0
6.  ENDFOR;
    /* main loop */
7.  WHILE NOT end of simulation DO
       /* Let event e be the next event to be executed in
       the sequential simulation */
8.     EXECUTE e;
       /* Update the real time clock of LP_i */
9.     T_i := max(T_i, e.α) + η(e) ;
10.    FOR ALL e' ∈ e.E DO
11.       SCHEDULE e'
12.       e'.α := T_i + δ(e')
13.    ENDFOR;
14.    T_S = T_S + η(e)
15. ENDWHILE.
```

Figure 1: Critical Path Analysis Algorithm (Lin 1992)

Adding computation and communication times, this algorithm will give an estimate of the parallelism within the model in terms of estimated speedup (the estimated sequential run time T_S divided by the estimated run time of the PDES (ie. the LP with the largest T_i)).

A criticism of this is that the estimated speedup is not an estimate of the parallelism within a given model but an evaluation of the decomposition of the model into LPs. This is a valid point. However, be it an evaluation of the parallelism within a simulation model or an evaluation of the decomposition strategy, this approach still provides a starting point in the study of justifying whether or not to go parallel.

3 PERFORMANCE ANALYSIS OF PDES: A CASE STUDY

To introduce how performance estimates can be determined, we consider a modelling technique and a parallelisation strategy. Petri Nets (PNs) provide a method for conceptualising a problem in terms of the logical flow of objects in the system (Peterson 1981, Marsan 1995). The model is represented as a bipartite multigraph with two types of nodes; *places and transitions.*

Places are usually drawn as circles and transitions as bars (straight lines). Each transition can have *input* and *output* functions that are defined by directed arcs. The net is *executed* by defining a *marking* for the places and then *firing* transitions. A marking is a distribution of *tokens* to the places of the PN, where tokens can under certain assumptions represent entities in a physical system. The marking represents the state of the system.

A transition is *enabled* when all of its input places have one or more tokens. A transition fires by removing one token from each of its input places and adding one token to each of its output places. The state of the system (the marking) changes as a result of the occurrence of events (transition firing).

For our purposes we will use an individual-token net to illustrate our problem. In individual-token nets every token represent an *entity*. The arcs of the net are labelled with the type of entities they can carry (coloured tokens).

Coloured tokens can represent the system's objects or entities in a simulation model. An entity is any component of the model that can be imagined to retain its identity through time. Entities are either idle, represented as places, or engaged in time consuming activities (represented as timed transitions). This sequence of idle and active states constitutes the life cycle of the entity. The model is then created by combining all of the individual life cycles. Obviously, these models cannot support token split and merger as tokens represent entities and not conditions.

Consider the partial PN model of Figure 2. This models a physical system where two entities must spend time together, co-operating in some activity or with some shared resource. In terms of the PN, an entity must be in place P_1 and an entity must be in place P_2 for the transition T_1 to be fired.

Conceptually, once this transition has fired the two entities are consumed. After the simulated time for the transition to complete has passed, the transition produces two entity copies that appear in the places P_3 and P_4 following the transition.

Decomposing this partial PN model into a PDES presents several possibilities. For purposes of this case study, we will use the technique put forward in Taylor

(1991) and Taylor (1993). The choice is arbitrary as we are concerned with illustrating the method by which the potential performance of a PDES can be estimated rather than evaluating a given PDES technique.

Decomposing the model of Figure 2 in terms of the event orientation (Derrick, Balci and Nance 1989), we associate LPs with each place in the model. This generates four LPs; LP_1, LP_2, LP_3 and LP_4 representing P_1, P_2, P_3 and P_4 respectively. Transitions are represented as event messages; an event message is sent from one LP to another indicating the time at which a transition will finish and produce tokens for another place. For example, LP_1 will send an event message to LP_3 to indicate the finish time of transition T_1 and the arrival of a token at P_3, LP_2 will send an event message to LP_4 to indicate the finish time of transition T_1 and the arrival of a token at P_4.

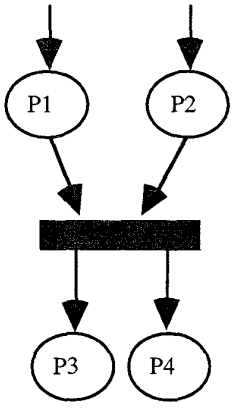

Figure 2: Model Illustrating Shared Resources

A problem arises when either LP_1 or LP_2 receives an event message from any transitions that precede them. Given that an event message represents the arrival of an entity in the place represented by an LP, if LP_1 receives an event message, it must determine if the arriving entity can be consumed by transition T_1. As the firing of T_1 depends on the contents of both P_1 and P_2 containing an entity, LP_1 cannot make the decision by itself; it needs information about the contents of P_2. As PDES dictates that information can only be shared between LPs by message exchange, to get this information LP_1 must query LP_2 about its contents, make the decision as to whether or not T_1 can fire and inform LP_2 about its decision. An information sharing protocol is therefore required to resolve this problem. Using the protocol discussed in Taylor (1991) and Taylor (1993) LP_1 will send a timestamped *query* message to LP_2 to request current state information. LP_1 will then wait for LP_2 to respond. LP_2 responds by sending a *reply* message to LP_1 containing information about its current contents. With this information about the combined state of LP_1 and LP_2, LP_1 can now determine if transition T_1 can fire. If T_1 *can* fire, LP_1 has the task of performing the simulation of T_1 (determining the entities that transition T_1 will produce and the time at which such entities will enter the places following T_1). Once LP_1 has done this it deletes entities consumed by T_1 from its own state, sends an *update* message back to LP_2 and sends a timestamped event message to LP_3 informing the LP that an entity is scheduled to arrive in place P_3 at the time at which transition T_1 finishes. On receipt of the update message, LP_2 updates its state (by removing entities consumed by the transition T_1) and sends a timestamped event message to LP_4 informing the LP that an entity is scheduled to arrive in place P_4 at the time at which transition T_1 finishes. If T_1 did *not* fire, then the *update* message must still be sent to complete the synchronous round of message exchange. However, no event messages will be sent as the transition did not fire. Finally, if LP_2 had received the event message, then the roles of the two LPs would be reversed.

The decomposed form of the partial PN model is shown in Figure 3 and the above discussion of the QRU-protocol is illustrated in Figure 4.

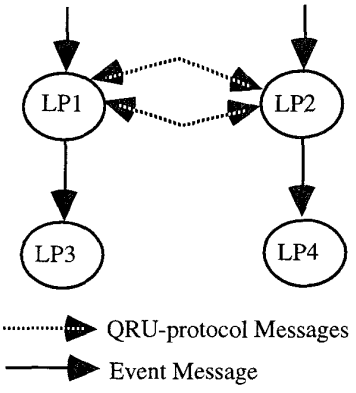

Figure 3: Decomposed Model Illustrating Shared Resources

Note that the design approach taken to this protocol postpones PDES ordering requirements. For purposes of this study, we will assume that messages will arrive in the correct order.

Remaining consistent with the terminology of the QRU-protocol, the requirement to share information between two or more LPs is the consequence of a *distributed event.*

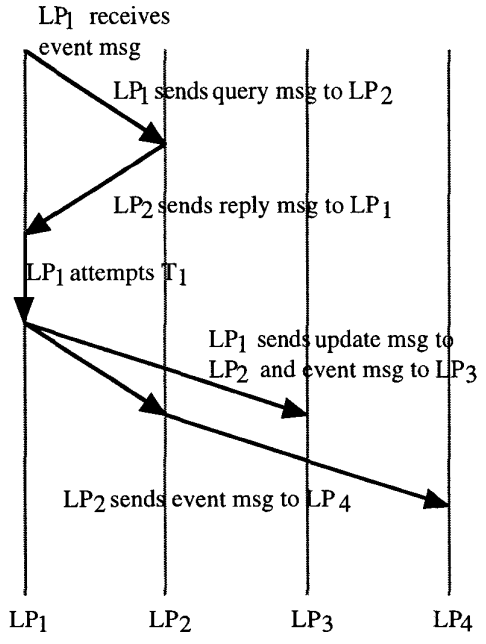

Figure 4: Illustration of the QRU-Protocol

A distributed event represents a synchronisation barrier across the LPs participating in that event. Further, boundaries can be identified which group together LPs participating in a series of distributed events. Such a tightly coupled group of LPs has been termed a *partition*. The extent of a partition is defined by the extent of the distributed events executed within the partition. Event messages are received by, and sent from, the processes in a partition across the partition border. Actions within a partition are entirely dependent on messages received across the partition border. For example, LP_1 and LP_2 form a partition.

3.1 A Performance Model

In a PN model there will be a set of places P, consisting of the queues $P_1, P_2, ..., P_x$ and a set of transitions T consisting of the transitions $T_1, T_2, ..., T_y$, where x and y are the number of places and transitions respectively. Decomposing such a model into a parallel simulation under the previously discussed approach will yield a set of processes LP, such that $LP_1 = P_1, LP_2 = P_2, ..., LP_x = P_x$. Each process participates in distributed events as dictated by the PN model. On the basis of mutually exclusive sets of transitions the parallel simulation will have a set of partitions M, consisting of the partitions $M_1, M_2, ..., M_z$, where z is the number of partitions.

Each partition P_i can potentially contain two types of LP.

- Instigant LP, *I*. A LP receiving an event message which then instigates execution in the partition.
- Non-instigant LP, *NI*. These are all the other LPs in the partition.

The LP type has bearing on the real time that a given LP sends event messages and finishes execution in a partition. For a partition M_i, let $\tau(M_i)$ be the earliest real time partition M_i receives an event message. Let $\overline{\tau}(M_i)_I$ and $\overline{\tau}(M_i)_{NI}$ be the earliest real times when LPs of type I and NI finish execution. We assume that a transition will only fire once for any set of markings. Let $\eta(T_i)$ be the execution cost for a transition T_i. Let E_{TiI}, and E_{TiNI} be the set of event messages sent by LPs of type I and NI for transition T_i. Let $E_{TiI}.\alpha$ and $E_{TiNI}.\alpha$ be the real times at which the event messages are due to arrive at their destination processes. Let $\delta(Q)$, $\delta(R)$, $\delta(U)$ and $\delta(E)$ be the communication times for *query*, *reply*, *update* and *event* messages sent from one LP to another when evaluating a transition. Note that if any transition does not fire, then the communication cost due to the QRU protocol is still incurred. If it is assumed that each LP executes in parallel then, for a partition with one transition and one place,

$$\overline{\tau}(M_i)_I = \tau(M_i) + \sum_{i}^{1} \eta(T_i)$$

and

$$E_{TiI}.\alpha = \tau(M_i) + \sum_{i}^{1} \eta(T_i) + \delta(E)$$

There are no NI processes in such a partition. For partitions with many transitions and one place,

$$\overline{\tau}(M_i)_I = \tau(M_i) + \sum_{i}^{i'} \eta(T_i)$$

where i' is the number of transitions in the partition, and

$$E_{TiI}.\alpha = \tau(P_i) + \sum_{i}^{i''} \eta(T_i) + \delta(E)$$

where i'' is the index of the successfully firing partition. There are no NI processes in such a partition.

For purposes of this case study, when there are many places and transitions in a partition, we will assume two possible interconnections. The first is *fully sequential* and the second *fully parallel*. In the first case, all the

places are connected to all of the transitions in a partition. This sequentialises the parallel execution in such a partition as each transition is dependent on *all* of the places in the partition; the LPs of the partition must continually exchange information. The second case assumes that all places share one and only one transition in the partition (a necessary condition). All places then have equal numbers of transitions connected to them. In this allocation, each place therefore shares a common transition and then has exclusive connection to zero or more transitions. The effect of this is that once the common transition has been evaluated, each LP can then evaluate its own transitions independently (and in parallel) with the other LPs of the partition. As these other partitions are effectively shared out amongst the places of the partition, the parallelism in a partition is maximised.

As will be seen, the choice of this model leads to fairly expected results. We take this opportunity to emphasise that the purpose of this paper is to introduce the reader to the technique rather than to perform evaluations of realistic models or protocols.

For fully sequential partitions we therefore have

$$\overline{\tau}(M_i)_I = \tau(M_i) + \sum_{i}^{i'} \eta(T_i) + i'\delta(Q) + i'\delta(R) + (i'-1)\delta(U)$$

$$\overline{\tau}(M_i)_{NI} = \tau(M_i) + \sum_{i}^{i'} \eta(T_i) + i'(Q) + i'\delta(R) + i'\delta(U)$$

and

$$E_{TiI}.\alpha = \tau(M_i) + \sum_{i}^{i''} \eta(T_i) + i''\delta(Q) + i''\delta(R) + (i''-1)\delta(U) + \delta(E)$$

$$E_{TiI}.\alpha = \tau(M_i) + \sum_{i}^{i''} \eta(T_i) + i''\delta(Q) + i''\delta(R) + i''\delta(U) + \delta(E)$$

where i' is the number of transitions in the partition and i'' represents the index of the successfully firing transition. For fully parallel partitions we have,

$$\overline{\tau}(M_i)_I = \tau(M_i) + \sum_{i}^{\mathrm{int}((i'-1)/j)+1} \eta(T_i) + (\mathrm{int}((i'-1)/j)+1)\delta(Q) + (\mathrm{int}((i'-1)/j)+1)\delta(R) + (\mathrm{int}((i'-1)/j))\delta(U)$$

where j is the number of places in the partition, and

$$\overline{\tau}(M_i)_{NI} = \tau(M_i) + \sum_{i}^{\mathrm{int}((i'-1)/j)+1} \eta(T_i) + (\mathrm{int}((i'-1)/j)+1)\delta(Q) + (\mathrm{int}((i'-1)/j)+1)\delta(R) + (\mathrm{int}((i'-1)/j)+1)\delta(U)$$

and

$$E_{TiI}.\alpha = \tau(M_i) + \sum_{i}^{i''} \eta(T_i) + i''\delta(Q) + i''\delta(R) + (i''-1)\delta(U) + \delta(E)$$

$$E_{TiNI}.\alpha = \tau(P_i) + \sum_{i}^{i''} \eta(T_i) + i''\delta(Q) + i''\delta(R) + i''\delta(U) + \delta(E)$$

To determine the estimated execution time, we modify the algorithm of Figure 1, by replacing line 9 and line 12 with appropriate execution time and communication time calculations.

3.2 Results

The modified algorithm in Figure 1 has been implemented for the fully connected model using a performance analysis tool implemented under the simulation package SES/Workbench (Scientific Engineering Software Inc. 1992). For purposes of this case study, several simulation runs were made for different model parameters. These parameters assumed uniform partitions, with the number of transitions equal to the number of places.

Two strategies were employed. The results to one experiment are shown here; the investigation of the effect of reducing the communications overhead within a partition by merging LPs. It was assumed that communication time remained constant and that the transition firing cost changed proportionately with the

number of LPs in the partition.

As can be seen in the graphs in Figure 5 and Figure 6, for the fully connected model, under conditions where partitions are forced into completely sequential execution (fully sequential), the worst speedup is obtained when the overall partition granularity is small (in the case of two LPs per partition). Increasing granularity by first increasing the number of LPs and then merging, generally increases performance. The difference becomes marginal between 16 LPs/Partition (E = 8) and 2 LPs/Partition (E = 64). This indicates that if a partition has largely sequential execution, LPs should be merged to reduce communications overhead and to increase LP granularity. When partitions are fully parallel, the best performances can be expected when there are many LPs that can be executed simultaneously. This is an intuitive result and is indeed the case, the best performance being when there are 128 LPs/Partition. The worst performances are obtained when there are few LPs/Partition.

4 CONCLUSIONS

This paper has presented a potentially powerful performance analysis tool which could possibly be used to estimate the worth of parallelising a discrete event simulation model and the structure thereof. A case study has been presented which deals with the performance analysis of a simulation model with distributed events. The results obtained show that for a fully connected model speedups are possible under some fully sequential and fully parallel conditions. At this level of analysis, it is implied that PN models such as those discussed in the paper should only be parallelised when good speed up is expected (in this case, when there are many fully parallel partitions).

So, on the basis of the case study, can we answer the question *"Is the analysis of model parallelism enough to justify the decision to parallelise?"* Of course not. The case study is far too simple to serve as anything else but an example of how we *could* go about the analysis of parallelism in a model. On the basis of this method, our research is currently formulating a series of more realistic models which take into account factors such as aggregation when there are more LPs than processors. For each model we will obtain the estimate of performance and then compare this against an actual implementation of such. The results of this exercise will be available at the time of the conference.

We expect the likely conclusion of this work to be that although the estimation of parallelism by simulation gives an indication of expected performance, the actual figure is misleading. The reason for this is that the current method does not take into account actual processing and communication times, LP to processor mapping and scheduling and the effect that the ordering protocol has on performance. It is considered that for this technique to be successful, these factors must be taken into account.

We must therefore emphasise that the contents of this paper represents the first step on a very long road.

REFERENCES

Berry, O. and D.R. Jefferson. 1985. Critical path analysis of distributed simulation. In *Proceedings of the 1985 SCS Conference on Distributed Simulation*,, 57-60. Society for Computer Simulation, San Diego, California.

Berry, O. 1986 Performance evaluation of the Time Warp distributed simulation mechanism. Ph.D. dissertation, University of Southern California.

Derrick, E.J., Balci, B. and Nance R.E. 1989. A Comparison of Selected Conceptual Frameworks for simulation Modelling, The Implementation of Four Conceptual Frameworks for Simulation Modelling in High Level Languages. In *Proceedings of the 1989 Winter Simulation Conference*, 711-717. Institute of Electrical and Electronics Engineers, San Francisco, California.

Fujimoto, R. 1993. Parallel Discrete Event Simulation: Will the Field Survive?. *ORSA Journal on Computing* 5(3).

Jefferson, D.R. and P. Reiher. 1991. Supercritical speedup. In *Proceedings of the 24th Annual Simulation Symposium*, 159-168.

Lin, Y-B. 1992. Parallelism analysers for parallel discrete event simulation. *ACM Transactions on Modeling and Computer Simulation* 2(3):239-264.

Livny, M. 1985. A study of parallelism in distributed simulation. In *Proceedings of the 1985 SCS Conference on Distributed Simulation*, 94-98. Society for Computer Simulation, San Diego, California.

Marsan, M.A. 1995. *Modeling with General Stochastic Petri Nets*. Chichester: John Wiley and Sons, to appear.

Nicol, D. and R.M. Fujimoto. 1995. Parallel Simulation Today. *Annals of Operations Research*, to appear.

Peterson J.L. 1981. *Petri Net Theory and the Modelling of Systems*, Englewood Cliffs: Prentice-Hall.

Scientific Engineering Software Inc. 1992. *SES/Workbench User's Manual, Release 2.1.* Scientific Engineering Software Inc.

Srinivasan, S. and P.F. Reynolds, Jr. 1993. On critical path analysis of parallel discrete event simulations. Computer Science Technical Report No. TR-93-29, School of Engineering and Applied Science,

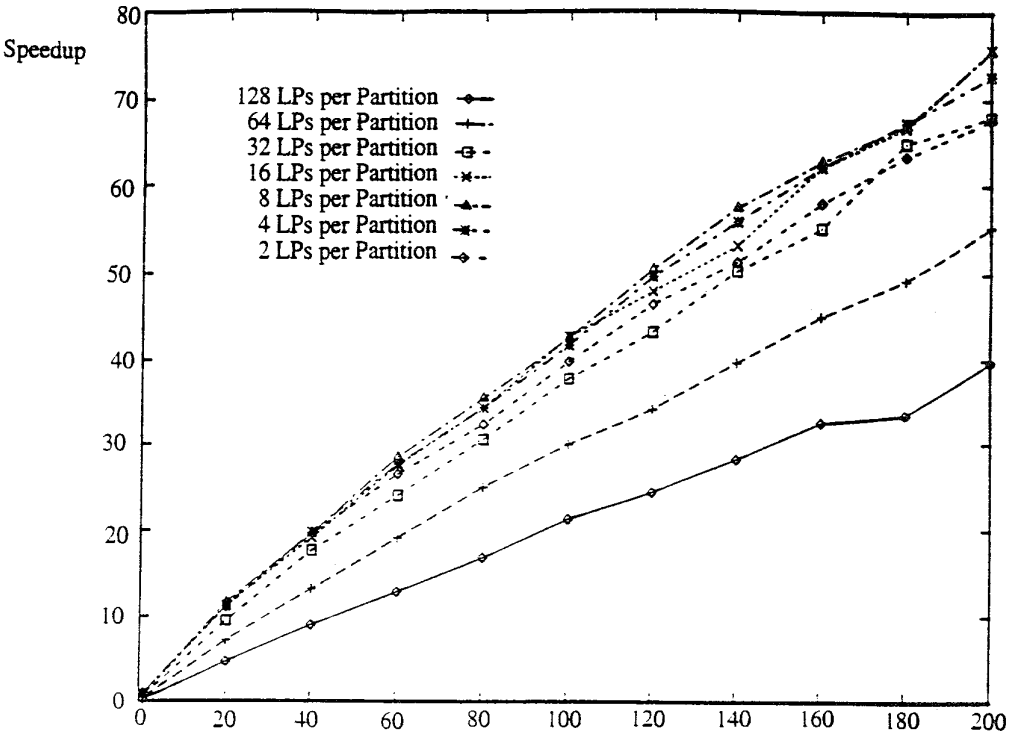

Figure 5: Fully Sequential

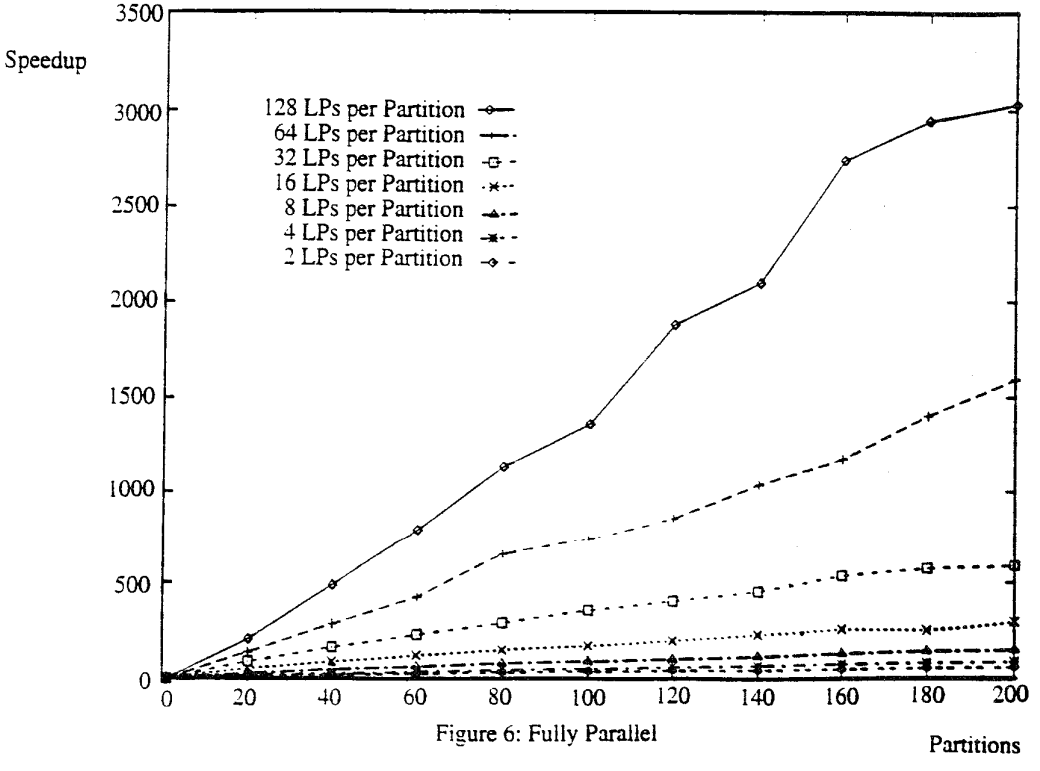

Figure 6: Fully Parallel

University of Virginia.
Taylor, S.J.E. 1991. The Development of a Parallel Activity-based Discrete Event Simulation. In *1991 European Simulation Multiconference*. Society for Computer Simulation, San Diego, California.
Taylor, S.J.E. 1993. The Application of Parallel Processing to Manufacturing Systems Simulation. Ph.D. thesis, Leeds Metropolitan University, Leeds, United Kingdom.

AUTHOR BIOGRAPHIES

SIMON TAYLOR is a lecturer in the Department of Computer Science and Information Systems at St. John's in Brunel University. He received his Ph.D in Parallel Simulation from Leeds Metropolitan University in 1993. His main research interests are automatic parallelisation within which parallel simulation forms part of a larger research effort. He is also interested in the application of simulation for problem solving, be the implementation parallel or sequential.

FARSHAD FATIN is a research student in the same Department. He is conducting his Ph.D. programme of study and is developing novel protocols for parallel simulation.

THIERRY DELAITRE is a research assistant at the Centre for Parallel Computing in the University of Westminster. He is registered for a Ph.D and is currently studying simulation techniques pertaining to the decomposition of parallel programs. At the time of writing he is implementing a simulator for parallel programs utilising PVM.