Towards Automated Performance Prediction in Bulk-Synchronous Parallel Discrete-Event Simulation

Mauricio Marín
Escuela de Computación
Universidad de Magallanes
Casilla 113-D, Punta Arenas, Chile
E-mail: `mmarin@ona.fi.umag.cl`

Abstract

This paper discusses the running time cost of performing discrete-event simulation on the bulk-synchronous parallel (BSP) model of computing [12, 16, 22]. The BSP model provides a general purpose framework for parallel computing which is independent of the architecture of the computer, and thereby it enables the development of portable software. In addition, the structure of BSP computations allows the accurate determination of the cost of parallel algorithms. We use this feature to devise a performance prediction methodology which enables the designer of parallel simulation models to predict in advance the systems which are amenable for efficient execution on a given BSP computer. The methodology is simple enough to be automated in parallel simulation languages.

1 Introduction

In BSP, any parallel computer is seen as composed of a set of P processor-local-memory components which communicate with each other through messages. The computation is organised as a sequence of *supersteps*. In each superstep, the processors may perform sequential computations on local data and/or send messages to other processors. The messages are available for processing at their destinations by the next superstep, and each superstep is ended with the barrier synchronisation of the processors. The total running time cost of a BSP program is the cumulative sum of the costs of its supersteps, and the cost of each superstep is the sum of three quantities: w, $h\,g$ and l, where **(i)** w is the maximum of the computations performed by each processor, **(ii)** h is the maximum number of words transmitted in messages sent/received by each processor with each one-word-transmission costing g units of running time, and **(iii)** l is the cost of barrier synchronising the processors. The effect of the computer architecture is cost by the parameters g and l, which are increasing functions of P. These values can be empirically determined [16].

Parallel discrete-event simulation (PDES) [3, 15], on the other hand, adopts the view of systems composed of a set of logical processes (LPs) that send messages to each other. The LPs encapsulate the state variables of the system and the messages contain events scheduled to occur at specific points of the simulation time. The events may modify the LP state variables and may schedule the occurrence of new events in other LPs. Once the LPs are placed onto the processors, one is faced with the non-trivial problem of synchronising the parallel occurrence of events. A number of *synchronisation protocols* have been proposed to solve this problem efficiently [3]. They can be classified into *conservative* and *optimistic* protocols, within each of which *asynchronous* and *synchronous* modes of operation can be identified. In most cases, synchronous protocols operate in a bulk-synchronous fashion which is very similar to that promoted by the BSP model. In addition, the efficient BSP implementation of asynchronous protocols has been investigated in [7].

It is widely accepted that the implementation of a parallel simulation is a quite involved and costly task. In contrast, it is much easier to quickly produce a prototypic sequential simulator for the same system. It would be desirable then to predict well in advance the feasible speedups to be achieved with a parallel simulator before doing the effort of implementing it. Moreover, the sequential simulator could be used to obtain an approximation of the behaviour of the actual parallel simulator. Note that an important advantage of the BSP model is that its cost model enables one to predict the performance of programs before their actual implementation. The parallel computer is characterised by the BSP parameters l and g, and the implied performance prediction methodology is simple: If for a given application we are able to estimate its cost of computation, communication and synchronisation, then the BSP parameters enable us to predict the running time of the application on different parallel computers.

In this paper we apply the ideas above to the PDES setting. We use the BSP cost model to obtain information about the relevant factors determining the efficiency of parallel simulations. We compute the speedup S considering a demanding case for the underlying parallel algorithm. We work with a few parameters representing classes of simulation models rather than specific models. To this end, in the expression for S we represent *any* simulation model by (mainly) three parameters. We use these parameters to determine the classes of systems for which a reasonable speedup can be achieved on a given parallel computer. This expression in combination with the augmentation of the sequential simulator to include an algorithm for simulating idealised synchronisation protocols, form the basis for the performance prediction methodology proposed in this work.

The rest of the paper is organised as follows. Section 2 derives the expression for the speed-up S. Section 3 presents an example which uses S to determine the range of simulation models which achieve $S > 1$ on two parallel computers. A validation of this predicted performance is also presented. Section 4 describes the basics on the sequential simulation of the operation of synchronous and asynchronous synchronisation protocols for PDES. The codes associated with these simulations are quite simple and general, which simplifies their automatic generation from a simulation language. Section 5 presents final comments and suggests further research.

2 Feasible speedups

In the following we use the term "parallel algorithm" in a generic sense because no specific synchronisation protocol is assumed. Moreover, we only consider an idealised situation where the overheads associated with the synchronisation protocol are not relevant to the computation of S. However, we believe that this is not a serious obstacle to predicting actual performance since these overheads can be made indeed small by designing efficient algorithms.

Note that we make some conservative assumptions in our cost model. We assume that the cost of processing an event in the parallel algorithm is identical to that of the sequential algorithm. We also consider that the cost of this event is the lowest (feasible) possible. These last two assumptions are pessimistic for the efficiency of the parallel algorithm since no advantage is taken regarding the partitioning of the system among the processors and the granularity of events is extremely small. In addition, we charge a rather high cost for the administration of messages within the processors. Thus, to some extent, these assumptions account for synchronisation protocol overheads.

Let us assume that the simulation goal is to process the occurrence of M_e events in the whole system. If the cost of processing every event e in a sequential algorithm is C_e^s, then the total cost C_S of the sequential simulation is given by

$$C_S = M_e\, C_e^s\, .$$

On the other hand, the total cost C_P of a parallel simulation is given by

$$C_P = (w + h\,g + l)\, N_{SP}\, ,$$

where N_{SP} is the total number of supersteps required to simulate the M_e events, and $(w + h\,g + l)$ the average cost of a superstep. On average a total of N_{EP} events are simulated in every superstep, namely

$$M_e = N_{EP}\, N_{SP}\, .$$

In a situation of perfect load balance each processor simulates the occurrence of N_{EP}/P events per superstep. The cost of processing each event is C_e^p with $C_e^p \leq C_e^s$ (i.e., no synchronisation protocol overheads are considered). However, the value of w is the average maximum amount of work performed by any processor in a superstep. Thus we introduce an imbalance factor P_B with $\frac{1}{P} \leq P_B \leq 1$, so that

$$w = N_{EP}\, P_B\, C_e^p\, .$$

The remaining $N_{EP}\,(1 - P_B)$ events are assumed to be uniformly distributed among the remaining $P - 1$ processors. In addition, we assume that a fraction P_M of all the events simulated by each processor in a superstep causes message transmissions. Thus

$$h = z\, N_{EP}\, P_B\, P_M\, ,$$

where z is the size of each message. Let us define $P_S = N_{SP}/M_e = 1/N_{EP}$ so that the speedup $S = C_S/C_P$ is given by

$$S = \frac{C_e^s}{P_B\, C_e^p \;+\; z\, P_B\, P_M\, g \;+\; P_S\, l}\,.$$

However, a harder requirement for the parallel algorithm is to assume $C_e^s = C_e^p = C_e = O(1)$ rather than $C_e^p < C_e^s$. In addition, let us include in C_P the computational overheads associated with the processing of messages during a superstep (sending and reception of messages). We assume an overhead of $C_e\, P_M/r$ units of running time for each processed event where $r \geq 1$ is the granularity of events. We also define $g_e = g/(r\, C_e)$ and $l_e = l/(r\, C_e)$, but restrict C_e to be the lowest (feasible) cost of processing an event in a particular machine. Thus we obtain,

$$S = \frac{1}{P_B\,(1 + P_M/r) \;+\; z\, P_B\, P_M\, g_e \;+\; P_S\, l_e}$$

with $\frac{1}{P} \leq P_B \leq 1$, $0 \leq P_M \leq 1$, $0 \leq P_S \leq 1$, $r \geq 1$, and $z \geq 1$. The parameter P_S is a measure of slackness since $1/P_S = N_{EP}$. The parameter P_M accounts for locality, and P_B for load balance. In this way simulation models can be represented by an instance of the tuple (P_B, P_S, P_M, r, z).

The final expression for the speedup S is useful in the sense that it describes a methodology for performance prediction on different parallel computers represented by their enhanced g_e and l_e values. For example, for a particular computer it would be interesting to know the range of values $(P_B, P_S, P_M, r = 1, z)$ for which it is possible to achieve at least $S = 1$. This would define the systems that are feasible for efficient simulation on that parallel computer. To this end the user would need to determine a lower bound for the cost C_e of processing an event and an estimation of the number z of words of typical messages. Then the g and l values of the parallel computer can be used to evaluate S using different range of values for (P_B, P_S, P_M).

An empirical lower bound for C_e can be determined by simulating the work-load of a system with very low cost per event. Let us consider the work-load of the sequential *hold model* [5]. This model, like any sequential simulation, is organised around a *pending event set* whose implementation is called the *event-list*. The sequential program works as follows:

> After inserting N events in the event-list, the hold model performs a sequence of $c\,N$ hold operations with $c \gg 1$. Each hold operation (**i**) retrieves from the event-list the event e with the least simulation time $e.t$, (**ii**) increases this time by $e.t + X$ units where X is a random variable, and (**iii**) then re-insert this event e into the event-list.

An accurate value of the cost of processing each event e can be obtained by measuring the total running time spent in processing the whole sequence of hold operations and dividing it by $c\,N$. Cost $C_e = O(1)$ can be achieved if we implement the event-list with the *calendar queue* [1] and use the exponential distribution for the values of the random variable X. Note that in the general case $C_e = O(\log N)$ (by means of heap priority queues) since the calendar queue can easily degenerate to $O(N)$ cost per processed event in certain systems.

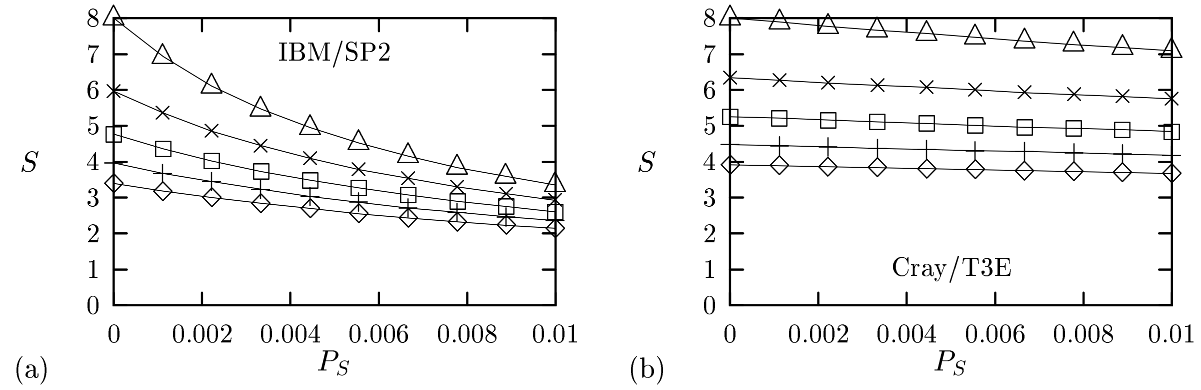

Figure 1: Performance prediction for IBM/SP2 and Cray T3E. $P = 8$, $P_B = 1/8$, $r = 1$, $z = 10$. Each curve is data for P_M=1.0 (◇), P_M=0.75 (+), P_M=0.5 (□), P_M=0.25 (×), and P_M=0.0 (△).

3 Example

An example of practical use of the speedup S is as follows. We implemented the above sequential $O(1)$ hold model and executed it on an 8-processors IBM/SP2 which has an average performance of 26 Mflops and whose BSP parameters are $l = 208.2$ μs and $g = 0.43$ μs/32-bit-word. We obtained $C_e = 12$ μs with error below 5% as we varied N from 10^2 to 10^4 (we used C-language and compiler optimisation -O3). We also extrapolated C_e to a Cray T3E with 46.7 Mflops, and BSP parameters for $P = 8$ of $l = 10.8$ μs and $g = 0.03$ μs/32-bits-word. We estimate $C_e = 12\,(26/46.7)$ μs for the Cray T3E.

Using the above values, the expression for S was evaluated for $P = 8$, $P_B = 1/P$, $r = 1$, $P_M = 1.0, ..., 0.0$, $P_S = 0.0, ..., 0.01$ (i.e., $N_{EP} = \infty ... 100$), and $z = 10$ words of 32 bits (we assume that every single event-message is composed of two doubles for send and receive timestamps, 6 integers for event and LPs identification plus data). The results are shown in figure 1. Some guidelines are the following.

Clearly, for $P = 8$ only large systems where $N_{EP} \gg 1$ can be simulated efficiently on both machines, whereas smaller systems are best suited for the Cray T3E as this machine has lower l and g values. In particular, the cost of barrier synchronisations has a significant impact in the efficiency of small systems.

For fixed P_S (specially the smaller ones), we also observe in figure 1 that computational overheads (represented by the term P_M/r in S) and the communication cost are responsible for an important part of the performance degradation as P_M approaches to one. Obviously as the event granularity r approaches to infinity, S indicates optimal speedup for any combination of parameters P_M and P_S. For example, we observed that S improves noticeably with $r = 2$.

The representative values of g and typical single message sizes ($z \approx 10$) we used in S, suggest

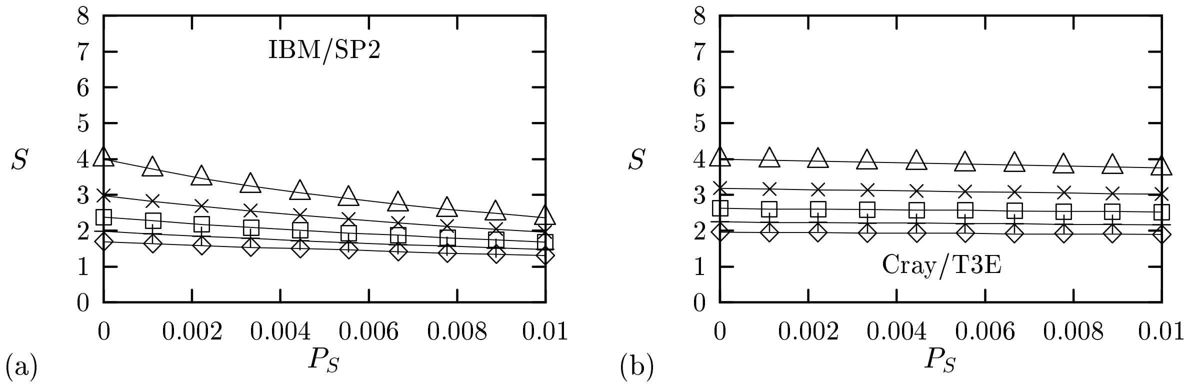

Figure 2: Performance prediction for IBM/SP2 and Cray T3E. $P = 8$, $P_B = 1/4$, $r = 1$, $z = 10$. Each curve is data for P_M=1.0 (◇), P_M=0.75 (+), P_M=0.5 (□), P_M=0.25 (×), and P_M=0.0 (△).

that in high performance parallel computers the effect of message transmissions is less important than the cost of computational overheads associated with the underlying synchronisation protocol (e.g., even for IBM/SP2 we have that $z\,g_e < 0.4$). That is, the design and implementation of efficient *sequential* algorithms for the computational part of a superstep plays a crucial role in the speedup achieved by the parallel simulation.

The effect of large imbalance $P_B = 2/P$ is shown in figure 2. These data show that S is highly sensitive to load balance and software overheads. Even though the parallel simulation can still achieve $S > 1$ for large imbalance, similar to other distributed memory architectures, the proper load balance of the simulation has a crucial effect in performance.

3.1 Validation

A BSP program was implemented to validate the performance predictions showed in figures 1 and 2. Its design is as follows. On each processor the above $O(1)$ hold model is executed. This hold model is parametrized so that the number of hold operations performed during a superstep is an input parameter of the program. This parameter enables us to control slackness. Our goal is to perform a total of 10^7 hold operations in the whole system. The measures of the sequential running time required to achieve this goal were made with the sequential program used to estimate C_e.

At the beginning of each superstep (**i**) the $N_{EP}\,P_B\,P_M$ ($N_{EP}\,\frac{1-P_B}{P-1}\,P_M$ for the other processors respectively) new messages are inserted in the calendar queue (for each message we delete an arbitrary item from the queue and insert the new event); (**ii**) then $N_{EP}\,P_B$ ($N_{EP}\,\frac{1-P_B}{P-1}$ respectively) hold operations are performed in the processor; and (**iii**) at the end of the superstep $N_{EP}\,P_B\,P_M$ ($N_{EP}\,\frac{1-P_B}{P-1}\,P_M$ respectively) messages are sent to other processors (an all-to-all communication is performed here). The size N of the whole system was set to 10^4 which means that in each processor the calendar queues maintains a total of 2500 events each.

On the IBM/SP2 we observed similar results to those in figures 1 and 2. The differences were below 20% in all the cases considered.

4 Simulating synchronisation sequentially

We now give directions to extend a sequential simulator with computations devised to predict the computation, communication and synchronisation requirements of a parallel simulator on a BSP machine. Note that both the event granularity r and the size of messages z can be determined from the sequential simulator.

The known synchronisation protocols [3, 4, 6, 13, 14, 15, 17, 18, 19, 20, 21] base their operation on one of two strategies of simulation time advance. Synchronous time advance (SYNC) protocols define a global time window to determine the events allowed to take place in each iteration (superstep). A SYNC protocol advances its time window forward in each iteration to let more events be processed. On the other hand, asynchronous time advance (ASYNC) protocols implicitly define windows which are local to the LPs (details in [11, 8, 7]).

Figure 3 describes a sequential simulation which predicts the supersteps executed by each protocol. For SYNC we describe event-horizon time advance as their associated protocols have been shown to be efficient [2, 18]. The system being simulated is composed of a set of LPs that send messages each other resembling a fully connected communication topology. More details are given in the figure caption where D_p is the number of LPs per processor. Some theoretical facts about these algorithms have been discussed in [10]. In particular, the ASYNC protocol is always able to complete a given simulation using the minimum number of supersteps.

5 Final comments

Basically the suggested methodology consists on estimating the tuple (P_B, P_S, P_M) for a given simulation model, and then using the particular g and l parameters of the available BSP machine to compute the speedup S. As the tuple (P_B, P_S, P_M) depends on the particular synchronisation protocol, we suggest using an augmented sequential simulator to determine its value if no other estimation (e.g., theoretical) is possible or accurate enough.

We emphasize that the algorithms in figure 3 only approximate the operation of actual synchronisation protocols. More research remains to be done in order to establish some measures of the degree in which these algorithms approximate the actual protocols. Empirical observations, for example, tell us that a protocol such as BSP Time Warp [11] achieves about twice the number of supersteps of ASYNC, but its load balance (which define computation and communication) is much better [9]. In general, we believe that the measures provided by the algorithms in figure 3 are useful enough to estimate the range of tuples (P_B, P_S, P_M) in which the system under study is in, and thereby they provide a first estimation of the suitability of PDES for the particular system. To the best of our knowledge, no other method which includes the effects of the synchronisation and architecture has been suggested in the literature.

Acknowledgement

This work has been partially funded by Fondecyt project 1990627.

References

[1] R. Brown. “Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem”. *Comm. ACM*, 31(10):1220–1227, Oct. 1988.

[2] P.M. Dickens, D.M. Nicol, P.F. Reynolds, and J.M. Duva. “Analysis of Bounded Time Warp and comparison with YAWNS”. *ACM Trans. on Modeling and Computer Simulation*, 6(4):297–320, Oct. 1996.

[3] R.M. Fujimoto. “Parallel discrete event simulation”. *Comm. ACM*, 33(10):30–53, Oct. 1990.

[4] D.R. Jefferson. “Virtual Time”. *ACM Trans. Prog. Lang. and Syst.*, 7(3):404–425, July 1985.

[5] D.W. Jones. “An empirical comparison of priority–queue and event–set implementations”. *Comm. ACM*, 29(4):300–311, 1986.

[6] B.D. Lubachevsky. “Efficient distributed event-driven simulations of multiple-loop networks”. *Comm. ACM*, 32(1):111–123, Jan. 1989.

[7] M. Marín. “Discrete-event simulation on the bulk-synchronous parallel model”. PhD Thesis, Programming Research Group, Computing Laboratory, Oxford University, 1998.

[8] M. Marín. “Direct BSP algorithms for parallel discrete-event simulation”. Technical Report PRG-TR-8-97, Computing Laboratory, Oxford University, 1997.

[9] M. Marín. “An Empirical Assessment of Optimistic PDES on BSP”. In *10th SCS European Simulation Symposium*, Oct. 1998.

[10] M. Marín. “Asynchronous (time-warp) versus synchronous (event-horizon) simulation time advance in BSP”. In *Workshop on Theory and Algorithms for Parallel Computation (Euro-Par’98)*, pages 897–905, Sept. 1998. LNCS 1470.

[11] M. Marín. “Time Warp On BSP Computers”. In *12th SCS European Simulation Multiconference*, June 1998.

```
Generate $N$ initial pending events;
[ $e$ is an event with time $e.t$ ]
$T_Z := \infty$; [ event horizon time ]
$S_Z \leftarrow \Phi$; [ buffer ]
loop
    if TimeNextEvent() $> T_Z$ then
        SStep := SStep + 1;
        Schedule($S_Z$);
        $T_Z := \infty$;
        $S_Z \leftarrow \Phi$;
    endif
    $e$ := NextEvent();
    $e.t := e.t$ + TimeIncrement();
    $p := e.p$; [ $e$ occurs in processor $p$ ]
    $e.p$ := SelectProcessor();
    if $e.p \neq p$ then
        $S_Z \leftarrow S_Z \cup \{e\}$;
        $T_Z$ := MinTime($S_Z$);
    else
        Schedule($e$);
    endif
endloop
```

(a) SYNC

```
Generate $N$ initial pending events;
[$e.s$ indicates the minimal superstep at
  which the event $e$ may take place in
  processor $e.p$.]
loop
    $e$ := NextEvent();
    $p := e.p$; [ $e$ occurs in processor $p$ ]
    if $e.s >$ SStep[$p$] then
        SStep[$p$] := $e.s$;
    endif
    $e.t := e.t$ + TimeIncrement();
    $e.p$ := SelectProcessor();
    if $p = e.p$ then
        $e.s$ := SStep[$p$];
    else
        $e.s$ := SStep[$p$] + 1;
    endif
    Schedule($e$);
endloop
```

The total number of supersteps is the maximum of the P values in array *SStep*.

(b) ASYNC

Figure 3:
Sequential programs describing the rate of superstep advance in the two approaches to simulation time advance. *Schedule*() stores events in the set of pending events. *NextEvent*() retrieves from this set the event with the least time. *TimeIncrement*() returns random time values. *SelectProcessor*() returns a number between 0 and $P-1$ selected uniformly at random. The variable/array *SStep* maintains the current number of supersteps of the *simulated* BSP machine. We assume $D_p = 1$ in these programs. For the case $D_p > 1$ and ASYNC algorithm, each LP maintains its own superstep counter *SStep*, and the counter values *e.s* carried by events are one unit larger than the respective *SStep* counter when the event is sent to a LP located in another processor. For the SYNC case, $D_p > 1$ makes no difference in the algorithm other than the associated increase of N since the event horizon time is calculated over all the events sent to other processors.

[12] W.F. McColl. “General purpose parallel computing”. In A.M. Gibbons and P. Spirakis, editors, *Lectures on Parallel Computation*, pages 337–391. Cambridge University Press, 1993.

[13] J. Misra. “Distributed discrete-event simulation”. *Computing Surveys*, 18(1):39–65, March 1986.

[14] D.M. Nicol. “The cost of conservative synchronization in parallel discrete event simulations”. *Journal of the ACM*, 40(2):304–333, April 1993.

[15] D.M. Nicol and R. Fujimoto. “Parallel simulation today”. *Annals of Operations Research*, 53:249–285, 1994.

[16] D.B. Skillicorn, J.M.D. Hill, and W.F. McColl. “Questions and answers about BSP”. Technical Report PRG-TR-15-96, Computing Laboratory, Oxford University, 1996. Also in *Journal of Scientific Programming*, V.6 N.3, 1997.

[17] L.M. Sokol, D.P. Briscoe, and F.P. Wieland. “MTW: A strategy for scheduling discrete simulation events for concurrent execution”. In *SCS Multiconference on Distributed Simulation 19 3*, pages 34–42, July 1988.

[18] J.S. Steinman. “SPEEDES: A multiple-synchronization environment for parallel discrete event simulation”. *International Journal in Computer Simulation*, 2(3):251–286, 1992.

[19] J.S. Steinman. “Breathing Time Warp”. In *7th Workshop on Parallel and Distributed Simulation (PADS’93)*, pages 109–118, May 1993.

[20] J.S. Steinman. “Discrete-event simulation and the event-horizon”. In *8th Workshop on Parallel and Distributed Simulation (PADS’94)*, pages 39–49, 1994.

[21] S.J. Turner and M.Q. Xu. Performance evaluation of the bounded time warp algorithm. In *6th Workshop on Parallel and Distributed Simulation (PADS’92)*, pages 117–126, 1992.

[22] L.G. Valiant. “A bridging model for parallel computation”. *Comm. ACM*, 33:103–111, Aug. 1990.