Asynchronous (time-warp) versus synchronous (event-horizon) simulation time advance in BSP

Mauricio Marín

Programming Research Group, Computing Laboratory, University of Oxford
E-mail: mmarin@comlab.ox.ac.uk

Abstract. This paper compares the very fundamental concepts behind two approaches to optimistic parallel discrete-event simulation on BSP computers [10]. We refer to (**i**) *asynchronous* simulation time advance as it is realised in the BSP implementation of Time Warp [3], and (**ii**) *synchronous* time advance as it is realised in the BSP implementation of Breathing Time Buckets [8]. Our results suggest that *asynchronous* time advance can potentially lead to more efficient and scalable simulations in BSP.

1 Introduction

As BSP is a (bulk) synchronous model of parallel computing [10], the obvious method of parallel discrete-event simulation [2, 7] on BSP computers seems to be synchronisation protocols based on *synchronous* time advance, such as Breathing Time Buckets [8]. In this paper we show that using synchronous time advance on a BSP computer might not be a good idea for two reasons. Firstly, synchronous simulation methods force the execution of periodical parallel min-reductions during which no interesting advance in simulation time can be made. These operations take some number of BSP *supersteps* to complete [10], and the simulation is stopped during their execution. The cumulative cost of these operations is relevant since the total number of min-reductions is equal to the total number of supersteps used to process the simulation events. Secondly and more importantly, synchronous methods impose barriers in simulation time which tend to reduce the rate of time advance per superstep of the simulation. This increases the total number of supersteps dedicated to parallel event-processing. Equivalently, time barriers tend to decrease the number of events that are processed in each superstep.

In BSP, supersteps are delimited by the barrier synchronisation of all the processors. In turn this operation takes some amount of real time to complete and it can become comparatively expensive in simulation models where the amount of computation involved in the processing of events is small (e.g., queuing networks). As many real life simulations can easily demand the execution of hundreds of thousands of event-processing supersteps, a significant reduction of the total number of supersteps can also cause a significant reduction of the total running time of the simulation. In addition, the cost of the remaining barrier synchronisation of processors can be amortised by the processing of a comparatively larger number of events per superstep.

On the other hand, approaches based on *asynchronous* time advance do not stop the simulation to perform periodical min-reductions, and simulation time barriers are not required (cf., [6]). For symmetric work-loads and excluding min-reduction supersteps, we have observed that as the size of the simulation model scales up, the rate of simulation time advance per superstep decreases comparatively faster using synchronous time advance. In particular, synchronous time advance requires on the average $O(\sqrt{P}/\ln\ln P)$ times more event-processing supersteps than asynchronous time advance, with P being the total number of logical processes (LPs). Moreover, for any simulation model, we show that the number of supersteps for the asynchronous approach is at most the supersteps required by the synchronous approach, and in the end the total number of supersteps required by the asynchronous approach is optimal for any model. In addition, for the same symmetric work-loads, we have observed that load balance tends to be better under asynchronous time advance for moderate level of aggregation of LPs onto processors (here load balance refers to balance in computation and communication as it is understood in BSP), whereas load balance is optimal in both approaches when the aggregation of LPs is large enough (domain of current practical simulations).

Obviously the nature of simulation work-loads is completely irregular and it is hard (perhaps impossible) to draw general conclusions about the overall running time achieved by protocols based on the two approaches analysed in this paper. We contribute with the analysis of an important performance indicator, namely total number of supersteps. In our view, the results presented in this paper suggest that the domain of simulation models and current BSP machines in which protocols based on synchronous time advance can be efficient should be expected to be limited. As to the development of general purpose BSP simulation environments, we conjecture that protocols based on asynchronous time advance offer better opportunities to achieve scalable performance.

2 Two approaches to time advance

Approaches based on the event-horizon concept [9] like Breathing Time Buckets [8], advance in simulation time in a *synchronous* manner by consuming supersteps as in figure 1.a. Note that here we do not consider the additional supersteps required for min-reductions which compute a new global event-horizon on each cycle. That is, each cycle of the pseudo-code shown in this figure should be followed by a min-reduction. Approaches like Time Warp [3], on the other hand, advance in simulation time in an *asynchronous* manner [6] by consuming supersteps as in figure 1.b. Note that only silent min-reductions are needed in these approaches since values such as *global virtual time* (GVT) [3] are calculated without stopping the simulation. That is, these min-reductions are carried out using the same supersteps used to simulate events. We call these two styles of superstep advance as SYNC and ASYNC respectively.

Lemma 1. *The total number of supersteps required by ASYNC is optimal.*

Proof: Dependencies among events form trees. If the occurrence of event e generates event e_1, then e_1 is a child of e. If e_1 is scheduled to occur in a different processor and e takes place at superstep s, then e_1 must be processed at least in the superstep $s+1$. But the simulation of e_1 may be delayed up to some superstep $s' > s+1$ if earlier events take place in e_1's processor at superstep s'. In this case, each descendant of e_1 may only take place at superstep $\geq s'$ and if an e_1's decendant takes place in another processor, it may occur at least in superstep $s'+1$. In this sense we say that every event has an initial "energy" which indicates the minimum superstep at which it can take place.

Figure 1.b helps us to realise that given any set of events to be processed during the whole simulation, the superstep counters are only updated with the *initial energy* of chronological events that cannot be processed in earlier supersteps. This rule is inductively applied in each processor from the first to the last superstep. In this way the simulation, as mapped onto the processors, is completed in the minimum number of supersteps. □

Lemma 2. *The total number of supersteps required by ASYNC is at most the supersteps of SYNC.*

Proof: A subset E_H of all simulation events are the events that mark the event-horizon times. These events are not necessarily causally related and they may be generated in different processors. Also note that these events are the least timestamped messages buffered during the respective SYNC supersteps. The size $|E_H|$ of this subset is the total number of supersteps required by SYNC since by construction these events cannot occur in the same superstep.

Consider the simulation with ASYNC. The first chronological event in E_H is necessarily the first event processed by one of the processors in its second superstep. However, the remaining processors may advance farther in time during their first superstep since they are not barrier synchronised by event-horizon times. This implies that more than one horizon event may be processed in the second superstep of ASYNC. Similar argument applies to the following supersteps so that, in general, SYNC is not optimal in supersteps. Both approaches require identical number of supersteps when, for example, each event in E_H is causally related so that they must be simulated sequentially. □

Lemma 3. *Under assumption of unlimited memory, Time Warp as realised in BSP can approximate the supersteps of ASYNC within* $\log P$ *supersteps.*

Proof: The lemma follows trivially by letting Time Warp process every available event (with time within the simulation period), correct erroneous computations accordingly, and start a new GVT calculation in each superstep. □

Note that processing every available event per superstep may lead to large roll-back overheads. However, near-optimal supersteps can be achieved at low overheads by limiting the number of events processed in each superstep [6].

```
Generate N initial pending events;
T_Z := ∞; [ event horizon time ]
S_Z ← Φ; [ buffer ]
loop
    if TimeNextEvent() > T_Z then
        SStep := SStep + 1;
        Schedule(S_Z);
        T_Z := ∞;
        S_Z ← Φ;
    endif
    e := NextEvent();
    e.t := e.t + TimeIncrement();
    p := e.p ; [ e occurs in processor p ]
    e.p := SelectProcessor();
    if e.p ≠ p then
        S_Z ← S_Z ∪ {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

Fig. 1. Sequential programs describing the rate of superstep advance in two approaches to parallel simulation. This program, called the hold-model [11], simulates work-loads of systems such as queuing networks. *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.

3 Average case analysis

It is not difficult to see that the number of supersteps per unit simulation time, denoted by S_p, required by SYNC and ASYNC is the optimal, $S_p = 1$, for fully connected communication topology when the *TimeIncrement* function of figures 1 returns 1. However, when this function returns random values, say exponentially distributed, the S_p values increase noticeably as shown in the following analysis (details in [5]). We assume one LP per processor.

Suppose that the initial event-list has N pending events e with timestamps $e.t$. Let X be a continuous random variable with p.d.f. $f(x)$ and c.d.f. $F(x)$. A hold operation consists of (**i**) retrieving the event e^* with the least timestamp from the event-list, (**ii**) creating an event e with timestamp $e.t = e^*.t + X$, and (**iii**) storing e in the event-list. e^* is discarded so that N remains constant throughout the whole sequence of hold operations. It is known [11] that after executing a long sequence of hold operations the probability distribution of the times $e.t$ approaches an steady state distribution with p.d.f. $g(y) = (1 - F(y))/\mu$ where $\mu = \mathrm{E}[X]$. We represent the $e.t$ values with the random variable Y. The values of Y are measured *relative* to the timestamp of the last event e^* retrieved by the last hold operation. Let $G(y)$ be the c.d.f. of Y.

SYNC: We use the above defined $f(x)$ and $g(x)$ probability density functions to calculate SYNC's S_p, say S_p^s, as follows. Let A be the minimum of the N random variables Y. Since $\mathrm{Prob}[A > x] = \overline{G}(x)^N$ the c.d.f. of A is $M_A(x) = 1 - \overline{G}(x)^N$. Let B be the minimum of the N random variables resulting from the sum of X and Y. The variable B represents the time of the event-horizon. The c.d.f. of $X + Y$, say $F_2(x)$, can be calculated using $F_2(x) = \int_0^x F(x-t)g(t)dt$ or $F_2(x) = \int_0^x G(x-t)f(t)dt$. The c.d.f. of B is then $M_B(x) = 1 - \overline{F_2}(x)^N$. Note that $E[B] - E[A]$ is the average time advance per superstep. Thus if the simulation ends at time $T \gg 1$, then it will require an average of $T/(E[B] - E[A])$ supersteps to complete. Therefore $S_p^s = 1/(E[B] - E[A])$.

Let us consider the negative exponential distribution with mean $\mu = 1$ for time increments X. In this case $f(x) = g(x) = e^{-x}$, therefore $E[A]$ is given by

$$E[A] = \int_0^\infty \overline{M}_A(t)\,dt = \int_0^\infty e^{-N\,t}\,dt = \frac{1}{N}\,,$$

and $E[B]$ is

$$E[B] = \int_0^\infty (e^{-t} + t\,e^{-t})^N\,dt = \sum_{i=0}^{N} \binom{N}{i} \int_0^\infty t^i\,e^{-N\,t}\,dt = \frac{1}{N} \sum_{i=0}^{N} \binom{N}{i} i! \left(\frac{1}{N}\right)^i .$$

Using the approximation given in [4] (pp. 112-117) we obtain

$$E[B] \approx \frac{5}{4}\,\frac{1}{\sqrt{N}} + \frac{3}{4}\,\frac{1}{N}\,,$$

so that

$$S_p^s \approx \frac{4}{5}\sqrt{N}\,.$$

Let us now consider SYNC's event-efficiency E_f, say E_f^s, which is defined as the ratio of the optimal number of events to the average *maximum* number of events processed in each processor per superstep. This is a measure of load balance in computation and communication for the studied symmetric workload. On average, a total of m events take place in each superstep. Given a sensible distribution of LPs onto the P processors, by Valiant's theorem [10] we learn that the average maximum number of events per superstep should tend to the optimal m/P as m scales up. Thus for m large enough the efficiency is close to 1. For exponential distribution, $m \approx \frac{5}{4}\sqrt{N}$ [5, 9]. Define $D = N/P$. Regression analysis on simulation data from the program in figure 1.a (validated with numerical evaluation of the expected maximum of P binomial random variables) produces (asymptotically) for exponential distribution [5],

$$E_f^s = \frac{D^{1/4}}{P^{1/4}\ \ln\ P + D^{1/4}}\ .$$

ASYNC: In this case there is no global synchronisation in simulation time. In a given superstep each LP p_i simulates events with time less than the minimum event time of all new events (messages) arriving to p_i from other LPs by the next superstep. For large number of LPs (which justifies parallel simulation itself), it is reasonable to assume that a significant amount of LPs will work with its own statistically independent local event-horizon. The average instance of this local event-horizon being the average minimum of D independent random variables $Z = X + Y$. Thus, from the previous results for SYNC, these comments lead to a first view of ASYNC's S_p, say $S_p^a \approx O(\sqrt{D})$ for fixed number of LPs P. That is, we conjecture that for $P \gg 1$ and $D \gg 1$, S_p^a asymptotically tends to the S_p value of a SYNC simulation with just D events. In the following we provide evidence supporting this claim.

The LPs advance the simulation in a generally different amount of time in each superstep. Note that the *average* time advance per superstep is by definition $1/S_p^a$. Let us consider a lower bound T_p for this time advance. Consider all the order statistics associated with the set of N random variables $Z = X + Y$. Thus $T_p =$ Average among the first P values $\mathrm{E}[Z_i]$, where the lower bound T_p comes from the assumption that the first P order statistics of Z are all located in a different LP. However, the actual average cannot be much larger than T_p since any difference $Z_i - Z_k$ increases very slowly with N. For example, for exponential distribution, the maximum Z_N increases only logarithmically with N [1]. In this case we conjecture that the true average time advance per superstep is $\approx 1.25/\sqrt{D}$. Let us then compare T_p with this quantity.

Unfortunately calculating T_p is mathematically intractable. Thus we performed the following experiment. For large N, say $N = 10^4$, we generated $I = 10^3$ different instances of a set of N random values $X + Y$, with X and Y being exponentially distributed. For each instance i we calculated the partial sums $\mathrm{Sum}[i, P] = \frac{1}{P}\sum_{k=1}^{P} Z_k$ with $1 \leq P \leq N$, so that T_p for a particular D is given by $T_p = \frac{1}{I}\sum_{i=1}^{I} \mathrm{Sum}[i, \frac{N}{D}]$. The results show that in the range $D \geq 10$, T_p behaves like $O(1/\sqrt{D})$. Assuming $T_p = \beta\,(P/N)^{\alpha}$ least squares regression on the points $(\log T_p, \log P)$ produced $\alpha = 0.47738$ and $\beta = 0.01194$ which should be compared with the conjectured $\alpha = 0.5$ and $\beta = 0.0125$ resulting from the curve $1.25/\sqrt{D}$.

We now consider the effect of P in S_p^a when D is fixed. Let χ_k be the sum of $k \geq 1$ random variables with p.d.f. $f(x)$ so that they represent the time increments of events forming a thread of causally related events. Then, for any positive difference between LP time advances, say $\Delta = T_j - T_i$, there is some non-zero probability that $T_j < T_i + \epsilon_i + \chi_k < T_j + \epsilon_j$ for some $\epsilon_i, \epsilon_j \geq 0$, such that $T_i + \epsilon_i + \chi_k$ is the time of the next event e_j in LP p_j after superstep s. This event e_j is processed in some superstep s' with $s + 1 < s' \leq s + k + 1$. An increase in

P implies an increase in the diversity of T_i values and thereby an increase in the probability of events of type "e_j". Thus, for fixed D, S_p^a may increase in about k supersteps with P. For exponential distribution the k values are Poisson. The average k for the maximum time interval $\Delta = Z_N - Z_1$ is bounded from above by $Z_N = O(\ln N)$. So a conservative upper bound for S_p^a is $O(\ln P)$. However, similar experiment to the above described tells us $Z_P < 1 \ln \ln P$ in the range $D \geq 10$. This leads to

$$S_p^a \approx \ln \ln P \sqrt{D} .$$

This expression was validated with simulation results obtained with the program in figure 1.b. Regression analysis on data from the same program produces (asymptotically) for exponential distribution [5],

$$E_f^a = \frac{D^{1/4}}{(\ln \ln P)^{1/2} \ln P + D^{1/4}} \left(\frac{1}{\ln P \ln D} \right) .$$

Comments: Note that the ratio S_p^s/S_p^a behaves in practice as $O(\sqrt{P})$ for large systems. For fixed P and large D values, SYNC's efficiency is better than ASYNC's efficiency. For large P's, $P^{1/4}$ goes to ∞ faster than $(\ln \ln P)^{1/2} \ln P$. Thus for large P and small D (practical simulation models) the efficiency of ASYNC is better than SYNC efficiency. However, the expression for E_f^a was obtained considering just one LP per processor. As more LPs are put on the processors, E_f^a improves noticeably. This is so because the variance of the cumulative sum of co-resident LP time advances is reduced as the number of LPs per processor increases. Consequently, the variance of the number of events simulated in each processor decreases [5]. Conversely, E_f^s is insensitive to the relation $D_{lp}\, D = m/P$ with D_{lp} being the number of LPs per processor. Another important point here is that we can always move ASYNC towards SYNC by imposing upper limits to the number of events processed in each processor and superstep (this filters peaks). If these limits are sufficiently small, ASYNC degenerates to SYNC [5].

To compare the two approaches under more practical grounds we performed experiments with programs similar to those shown in figure 1. First note that the work-load generated by these programs is equivalent to a fully connected queuing network where each node (LP) contains infinite servers. The service times are given by the time increments of events. In our experiments we considered a more realistic queuing network: one non-preemptive server per node, exponential service times, fully connected topology, and unlimited queue capacities with moderate number of jobs flowing throughout the network (closed system). Figure 2 shows the results. The comparison is made in terms of $R_S = S_p^s/S_p^a$ and $R_E = E_f^s/E_f^a$. The plotted data clearly show that ASYNC outperforms SYNC in a wide range of parameters (the minimal value of R_S in figure 2.a is 1.9). The data indicate that R_S increases as $\sqrt{P}$ and R_E decreases when N and P scale up simultaneously. In other words, the results show that ASYNC has better scalability than SYNC. In addition, the level of aggregation of LPs onto processors contribute to further increase R_S and decrease R_E. In particular, for fixed P, the ratio R_S increases as $\sqrt{D_{lp}}$ in figure 2.a.

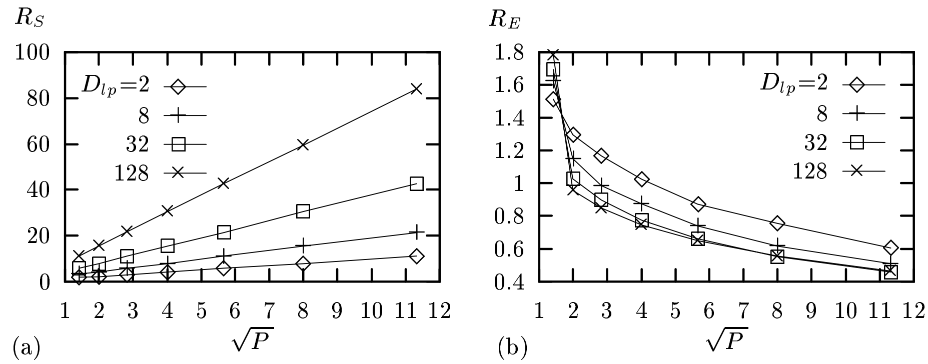

Fig. 2. Fully-connected non-preemptive single-server queuing network with exponential service times. Figure a: R_S= supersteps-SYNC/supersteps-ASYNC. Figure b: R_E= AvgMaxEv-SYNC/AvgMaxEv-ASYNC where AvgMaxEv is the average maximum number of events per superstep. D_{lp}= number of LPs per processor, and P= number of processors. Initially $D = 8$ "jobs" are scheduled in each LP (server).

References

1. R. Felderman and L. Kleinrock. "An upper bound on the improvement of asynchronous versus synchronous distributed processing". In *SCS Multiconference on Distributed Simulation V.22*, pages 131–136, Jan. 1990.
2. R.M. Fujimoto. "Parallel discrete event simulation". *Comm. ACM*, 33(10):30–53, Oct. 1990.
3. D.R. Jefferson. "Virtual Time". *ACM Trans. Prog. Lang. and Syst.*, 7(3):404–425, July 1985.
4. D. E. Knuth. *"The Art of Computer Programming, Vol. 1,* Fundamental algorithms*"*. Addison-Wesley, Reading, Mass., 1973.
5. M. Marín. "Simulation time advance in BSP". Technical Report Oxford University, May 1998. http://www.comlab.ox.ac.uk/oucl/groups/bsp/.
6. M. Marín. "Time Warp On BSP Computers". Technical Report Oxford University, Feb. 1998. To appear in 12th European Simulation Multiconference.
7. D.M. Nicol and R. Fujimoto. "Parallel simulation today". *Annals of Operations Research*, 53:249–285, 1994.
8. J.S. Steinman. "SPEEDES: A multiple-synchronization environment for parallel discrete event simulation". *International Journal in Computer Simulation*, 2(3):251–286, 1992.
9. J.S. Steinman. "Discrete-event simulation and the event-horizon". In *8th Workshop on Parallel and Distributed Simulation (PADS'94)*, pages 39–49, 1994.
10. L.G. Valiant. "A bridging model for parallel computation". *Comm. ACM*, 33:103–111, Aug. 1990.
11. J.G. Vaucher. "On the distribution of event times for the notices in a simulation event list". *INFOR*, 15(2):171–182, May 1977.

This article was processed using the LaTeX macro package with LLNCS style