APPLICATION OF TIME WARP TO PARALLEL SIMULATIONS WITH ASYNCHRONOUS CELLULAR AUTOMATA

B. J. Overeinder and P. M. A. Sloot
Parallel Scientific Computing and Simulation Group
Department of Computer Systems, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands

ABSTRACT

A new asynchronous cellular automata model is described. The model provides an experimental and theoretical framework to investigate quantitatively realistic simulations. The parallel execution of such simulations introduces the need of synchronisation of the asynchronous parallel processes. Time Warp is proposed as a synchronisation method for the parallel asynchronous cellular automata. We introduce two new extensions to the Time Warp method to make the method more effective for the application to parallel simulation of asynchronous cellular automata.

1 INTRODUCTION

The progress in information technology and the increasing need for more and more detailed simulations of Dynamic Complex Systems (DCS) has resulted in the exploitation of parallelism. The modelling and simulation of dynamic complex systems incorporate the abstraction and detailed representation of basic mechanisms of real world systems. In this respect, simulation allows for the understanding, prediction and optimisation of real world systems. It is to be expected, with the improving hardware and software technology, that the degree of accuracy and realism of the simulation of dynamic complex systems will grow vastly beyond present day capabilities.

A distinguished computational solving method for a large class of dynamic complex systems are Cellular Automata or Synchronous Cellular Automata. The method is in itself a set of dynamic systems where space, time, and variables are discrete. Cellular Automata exhibit remarkable self-organisation that can be used in models for real world systems. For instance, the CA technique has proven to be useful for direct simulation of fluid flow experiments in both two and three dimensions. Another application of CA can be found in lattice spin models such as the Ising model. In biology, CA are used in, for example, immune deficiency in cancer tissue simulations.

A less known computational solving method for complex dynamic systems are Asynchronous Cellular Automata (Ingerson and Buvel 1984; Lubachevsky 1987; Overeinder et al. 1992). Here, the commonly made preassumption that all cell values are updated synchronously is relaxed. In this paper we report on extension of the CA model to incorporate asynchronous cell updates, and consequences of the functional and implementational aspects of the reformulation. This extended model allows for a more general approach to CA and can result in efficient parallel implementations for certain classes of long standing realistic problems.

2 THE ASYNCHRONOUS CELLULAR AUTOMATA MODEL

2.1 Asynchronous State Changes

The cellular automata theory describes a universe consisting of a homogeneous array of cells. Each cell is provided with a finite number of states and evolves in time according to well defined uniform local transition rules. The Asynchronous Cellular Automata (ACA), like the Synchronous Cellular Automata (SCA), is a set of dynamic systems where space and variables are discrete. But unlike the SCA, the ACA evolves not in discrete, but in continuous time. In the ACA model the cellular automata theory is extended to take the asynchronous state changes into account. State changes are instantaneous and can occur asynchronously and at unpredictable random times. State changes with these characteristics are also called events.

A model that evolves in continuous time and has a discrete state can be classified as a discrete event model (Zeigler 1976). In a discrete event model, even though time flows continuously, state changes can occur only at countable points in time, i.e., time jumps form one event to the next, and these events can occur arbitrarily separated in time.

Homogeneous dynamic systems with asynchronous updates, such as the proposed ACA model, can be forced to behave in a highly inhomogeneous fashion. For

instance in a random iteration model it is assumed that each cell has a certain probability of obtaining a new state and that cells iterate independently. As an example one can think of the continuous time probabilistic dynamic model for an Ising system (Lubachevsky 1988). With independent clocks, the most obvious model to use would be a system where each cell takes a certain amount of time to iterate but each has a slightly different environment with its own specific iteration time. With asynchronous cellular automata we can solve more complicated problems, closer to reality.

2.2 The Model

We define a deterministic asynchronous cellular automata by $Z = (I^d, N, V, v_0, f, F, \mathrm{T})$, where:

1. I^d is the set of d-tuple integers, called an array of the cellular space. An element of I^d represents the coordinate of a cell or a cell at that coordinate. The positive integer d is called the dimension of the cellular space.

2. N is an n-tuple of different elements of I^d, called the neighbourhood index. With $N = (\mathsf{n}_1, \cdots, \mathsf{n}_n)$ and given a cell $\boldsymbol{a}$, an element of the set $\{(\boldsymbol{a} + \boldsymbol{n}_1), \cdots, (\boldsymbol{a} + \boldsymbol{n}_n)\}$ is called a neighbourhood cell of $\boldsymbol{a}$, and $\boldsymbol{a}$ is the centre cell of those neighbourhood cells. The set consisting of elements of N is simply called the neighbourhood.

3. The cellular space is homogeneous. Thus for all cells, V is a nonempty finite set, called a state set.

4. The state set V has an element v_0 in which the cell is at rest, called the quiescent state.

5 The local function f is a mapping from V^n to V and satisfies the specific property $f(v_0, \cdots, v_0) = v_0$.

6. The next state of $\boldsymbol{a}$ is given by $s_t(\boldsymbol{a}) = F(\boldsymbol{a}, N, f, t)$, where f is instantaneous applied to the neighbourhood N of $\boldsymbol{a}$ at time t.

7. The time of the next state change evaluation of $\boldsymbol{a}$ is described by $t' = \mathrm{T}(\boldsymbol{a}, N, t)$, where $t' > t$.

A nondeterministic asynchronous cellular automata can be obtained by introducing a random experiment and defining a random variable on the sample space of the experiment. The nondeterministic local function f is augmented with ξ, which is a realisation of the random variable. The local function can be written as f_ξ, where one can think of f_ξ as a tabulated function depending on ξ, or if the random variable is discrete, ξ can be considered as an argument to the function.

In the same way the time increment function can be written as T_ξ, where T_ξ can be a tabulated function depending on ξ, or, for both continuous or discrete random variables, ξ can be considered as an argument to function T_ξ. The use of ξ for continuous random variables is valid since time is continuous in asynchronous cellular automata.

3 THE SIMULATION ENVIRONMENT

To provide a platform for experimentation with dynamic complex systems that can be modelled with ACA, a simulation environment has been developed on a parallel architecture.

A SCA evolves in discrete time where the simulated time advances in fixed increments, called ticks. During each clock tick all cells undergo simultaneous state changes: state of a cell at $t + \Delta t$ is calculated from the state of the cell and its neighbours at time t. All cells must finish their state transition before any can start simulating the next tick.

The parallelisation of the discrete time simulation is achieved by imitating the synchronous behaviour of the simulation. The simulation is arranged into a sequence of rounds with one round corresponding to one clock tick. Between each round a global synchronisation of all cells indicates that the cells have finished their state change at time step t and the new time step $t + \Delta t$ can be started.

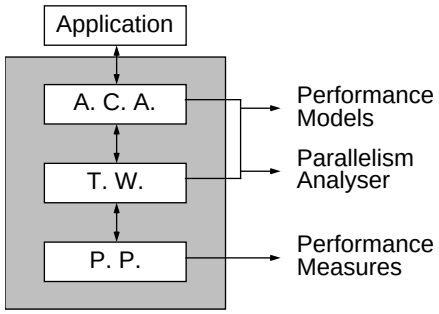

Figure 1: The simulation environment (T.W. is the Time Warp synchronisation method, and P.P. is a parallel platform).

The ACA described in section 2 is formulated as an asynchronous discrete event model. An efficient execution mechanism for this class of models is discrete event simulation, where periods of inactivity are skipped by the simulation process. The asynchronous execution mechanism complicates the parallelisation of the simulation through the need of explicit synchronisation of the distributed cells (see section 4.1 for more detailed explanation). In our implementation on a parallel machine we incorporate the so-called Time Warp method (Jefferson 1985; Overeinder

et al. 1991; Overeinder et al. 1992) to account for this explicit synchronisation.

In addition to the modelling and parallel simulation of ACA, we are also interested in the performance prediction and evaluation of the simulation environment. Therefore an analytical performance model has to be constructed, and within the simulation environment a parallelism analyser and a performance measurement tool must be incorporated.

The analytical performance model of the ACA and the Time Warp method is constructed for the prediction of the parallel performance of the ACA with the Time Warp method. We describe the stochastic characteristics of the components of the system with use of Markov modelling and measure their response to well defined stimuli.

The use of the parallelism analyser is twofold. First, it can be used within the validation/verification of the stand-alone ACA performance model. The ACA performance model gives a prediction about the parallel performance, and the parallelism analyser actually measures the potential parallelism, i.e., the parallelism inherent to the application. Second, this measured inherent parallelism is used in the evaluation of the efficiency and effectiveness of the parallel simulation method. The performance metrics measured after execution of the ACA with Time Warp can be compared with the results of the parallelism analyser. In this way it is possible to see how much of the potential parallelism in the application is actually realised. This allows for fine tuning of the simulation environment to the parallel platform.

The analytical model of the ACA and the Time Warp method can than be verified by performance measurements extracted from the execution of the simulation system. This will result in a better understanding whether an application can be effectively solved by an ACA in combination with the Time Warp mechanism.

In this paper we concentrate on the ACA model and the Time Warp method (the grey area in Fig. 1).

4 THE SIMULATION ENGINE: TIME WARP WITH EXTENSIONS

4.1 Time Warp

The parallel discrete event simulation method described here is based on a process-oriented view of simulation. The system being modelled is viewed as being composed of some number of physical processes that interact at various points in simulated time. The simulation is constructed as a set of logical processes, LP_0, LP_1,..., one per physical process. All interactions between physical processes are modelled by time stamped messages sent between the corresponding logical processes. Each logical process contains a portion of the state corresponding to the physical process it models, as well as a local clock that denotes the progress of the process.

In parallel discrete event simulation, the fundamental problem is the local causality constraint. A simulation is locally causal if and only if each process in the simulation executes events in non-decreasing time stamp order.

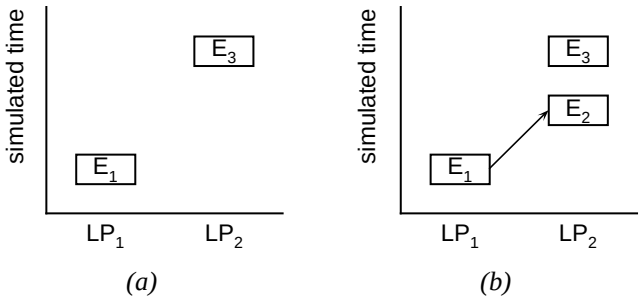

Figure 2: Causality error.

In Fig. 2 the possible appearance of a causality error is depicted. In Fig. 2(a) two events are scheduled, E_1 at logical process LP_1 with time stamp 10, and E_2 at LP_2 with time stamp 20. If the execution of E_1 schedules a new event E_3 for LP_2 containing a time stamp less than 20 (Fig. 2(b)), then E_3 could affect E_2, necessitating sequential execution of all three events. As a consequence a synchronisation method is needed to deal with the possible appearance of causality errors in the parallel discrete event simulation.

In Time Warp the basic synchronisation mechanism is process roll back. Basically, the parallel execution of a simulation with use of Time Warp can be characterised by three phases.

- Process executes until causality error occurs.
- Process rolls back in simulated time.
- Recovery by undoing the effects of all events executed prematurely.

Recovery from the premature execution of an event results in two things that have to be rolled back: the state of the logical process and the event messages sent to other processes. Rolling back the state is accomplished by periodically saving the process state and restoring a state vector valid for the current (rolled back) simulation time. “Unsending” a previously sent message (event) is accomplished by sending an anti-message that annihilates the original when it reaches its destination. These anti-messages are negative copies of the messages sent during normal simulation and are saved in a data structure. The messages that are sent while the process propagates in simulated time are called positive messages.

If a process receives an anti-message that corresponds to an unprocessed positive message, then the two will

annihilate each other and the process will proceed. If an anti-message arrives that corresponds to a positive message that is already processed, then the process has made a error and must also roll back. A consequence of the roll back mechanism is that more anti-messages may be sent to other processes recursively.

Although the Time Warp method is a solid general simulation methodology in itself, several extensions need to be included to make the application of the method to ACA efficient and effective. The first problem is the size of the state vector. Many asynchronous cellular automaton are aggregated in one process to have control over the grain size of the process and match this to the parallel machine. As a consequence the state vector can be quite large, while the changes to the state vector are minimal. We developed a new method that, instead of saving the whole state vector, only saves the changes to the state vector. This method is called *incremental* state saving. Another problem encountered is the need to cancel future events. In the simulation of the ACA, the execution of an event can make another event (scheduled or already executed) obsolete. We have introduced primitives that enable the simulation to undo the generation or execution of an event.

4.2 Incremental State Saving

A serious problem in Time Warp is the need to periodically save the state of each process. Normally, state saving is based on a method called check pointing, which periodically copies the whole state of a process into a state queue. The related overhead limits the effectiveness of the Time Warp mechanism, especially when the state consists of large dynamic data structures, which is the case with ACA.

But not only time considerations are important. Memory size limits the number of copies of the state vector that can be saved. Due to large state vectors, the limited number of saved state vectors can implicate that upon a causality error the process must roll back further in time than strictly necessary for correct simulation. This also degrades the effectiveness of the Time Warp mechanism.

We developed and implemented a new incremental state saving strategy, by exploiting the Markovian behaviour of the state evolution. This is accomplished by saving the processed event together with the side effects it is responsible for. Since one Time Warp process consists of a large number of cellular automata, the state is actually a vector of states of all the cellular automata it contains. The state of the process can be written as $s = \left(s(\boldsymbol{a}_{\boldsymbol{i}_1}),\ldots,s(\boldsymbol{a}_{\boldsymbol{i}_n})\right)$, where $\boldsymbol{i}_1,\ldots,\boldsymbol{i}_n \in I^d$.

The execution of event $e^t_{\boldsymbol{i}_j}$, $\boldsymbol{i}_j \in I^d$, only changes the state of a cellular automaton $\boldsymbol{a}_{\boldsymbol{i}_j}$ at simulation time t. Thus the new state is $s' = \left(s(\boldsymbol{a}_{\boldsymbol{i}_1}),\ldots,s'(\boldsymbol{a}_{\boldsymbol{i}_j}),\ldots,s(\boldsymbol{a}_{\boldsymbol{i}_n})\right)$. Instead of saving the complete state vector, the old state $s(\boldsymbol{a}_{\boldsymbol{i}_j})$ is sufficient to be saved. In incremental state saving is $s(\boldsymbol{a}_{\boldsymbol{i}_j})$ now associated with event $e^t_{\boldsymbol{i}_j}$.

Upon roll back of event $e^t_{\boldsymbol{i}_j}$ the state vector can be reconstructed by processing the event-state collection in reverse order, i.e. restoring state $s(\boldsymbol{a}_{\boldsymbol{i}_j})$, until the last event with a time stamp before the event that caused the roll back. As a consequence, the correct state has been reconstructed and simulation can proceed. Incremental state saving requires less state saving time and memory, at the cost of state reconstruction. The efficiency of the incremental state saving method compared to the check pointing method depends on the frequency at which roll backs occurs. If during simulation many roll backs arise, the overhead of state reconstruction in incremental state saving can dominate over the state copying in check pointing. On the other hand if roll backs are infrequent, incremental state saving will be more time and space efficient than check pointing.

4.3 Direct Event Cancellation

Depending on the model being simulated with an Asynchronous Cellular Automata, it is possible that an event will be scheduled that, by the execution of subsequent events, will be incorrect. Consider for example a physical model where different particles can annihilate or be combined into new particles. If at two neighbouring cells one particle and one anti-particle resides, it is possible that the execution of an event for the anti-particle results in the annihilation of the anti-particle with the particle. The next scheduled event for the particle at the neighbouring cell is now obsolete and should be retracted.

For these situations it is of great importance to have a means of retracting scheduled events. The proposed cancel primitive retracts previous scheduled events by cancelling the respective send operation (events are modelled by time stamped messages). The annihilation mechanism in the Time Warp method is transparent to the application program and is used to recover from synchronisation errors. It can therefore not be efficiently used for our purpose. The new cancel primitive on the other hand is invoked by the application itself for the retraction of previously scheduled events.

To cancel a previous send operation it is necessary to identify the message that has to be cancelled. For this reason, the send primitive returns a message descriptor for later use: *mess_id = tw_send(LP, ts, M)*. The primitive *tw_cancel(mess_id)* cancels a previously sent message and retracts the scheduled event. The simulation time at which the cancel primitive is invoked must be less then the schedule time of the event cancelled.

The cancel operation is inverse to the send operation in that it sends an anti-message and saves a copy of the corresponding positive message in a data structure. The anti-message annihilates with the original positive message according to the annihilation mechanism described in section 4.1. The cancel operation can also be rolled back in the same way as the send operation can be rolled back. If a cancel operation is rolled back, the original positive message is re-sent as if the event message was never cancelled.

The cancel primitive introduced here has similarities to the one described in (Lomow et al. 1991), in that it is a natural extension of the annihilation mechanism available in the Time Warp method. Another approach to direct event cancellation is described in (Agre and Tinker 1991). Agre and Tinker consider the cancel operation as a non-retractable event, i.e., an event that can not be rolled back. This means that it is not executed until the cancel event is the unprocessed event with the smallest time stamp of the whole simulation system. This avoids having to deal with roll back of a cancellation, but significantly reduces the parallelism in the simulation system.

5 THE IMPLEMENTATION OF AN ACA: WA-TOR

To validate our ACA model and the application of the Time Warp synchronisation method to the parallel ACA, we have implemented a well-defined test problem Wa-Tor. The original problem description of Wa-Tor is changed to obtain a typical irregular distributed simulation problem, suited to test the ACA and the introduced extensions to Time Warp.

5.1 The Problem: Wa-Tor

The Wa-Tor algorithm consists of a set of simple rules that describe the behaviour of sharks and fish in a torus shaped ocean. In the original presentation of Wa-Tor (Dewdney 1984), time passes in discrete steps (discrete time simulation), during which a fish or shark may move north, east, south, or west to an adjacent point. As time is discrete and space is discrete and homogeneous, Wa-Tor can be considered as a two dimensional CA. By introducing independent continuous time clocks for each shark and fish, and a "time to activate" parameter, Wa-Tor can be forced to behave in an asynchronous way. The resulting asynchronous Wa-Tor problem is well suited for implementation as an ACA and incorporation of the extensions to Time Warp.

The dynamics of the fish and sharks are given by a set of rules working on a rectangular ocean grid with periodic boundary conditions of a torus.

The behaviour of fish is described by the following rules. Each fish selects an unoccupied neighbouring location and moves to there. If no empty location is found, the fish does not move. In either case, the "time to activate" parameter takes a new value, and the fish is put to sleep. After the fixed period of time the fish is activated and the age is accordingly incremented. If the age has reached a value at which fish breed and the fish has actually moved, a new fish is placed at the old location.

The behaviour of sharks is similar to fish, except hunting for fish takes priority over mere movement. Sharks search for fish at their neighbouring locations, select one at random, and eat the fish. If no fish are in the neighbourhood, the sharks move just as fish do. The shark is put to sleep for some period of time, after which it is re-activated and its age increments accordingly. As with the fish, the sharks breed if their age have reached some value. If a shark swims for a certain amount of time without eating, it dies.

5.2 The Implementation

The Wa-Tor problem is directly mapped to an ACA by considering each ocean grid point as an asynchronous cellular automaton. Each ocean grid point (asynchronous cellular automaton) behaves according to the dynamic rules described in section 5.1 and depends on the state of the automaton and its neighbours.

The decomposition of the ocean into subdomains preserves the local characteristics of the Wa-Tor problem in the parallel simulation. Each subdomain send messages to other subdomains, embodying the movements of creatures from one subdomain to another. The synchronisation between the subdomains is provided by the Time Warp method.

The cancel operation finds its use when a shark eats a fish. At the moment a shark has selected a neighbouring location containing a fish, the shark moves to the new location and eats the fish. The scheduled activation event for the non-existent fish must be cancelled now, since the execution of the event is incorrect.

The implementation of the Time Warp synchronisation method includes the incremental state saving method. The state change of each ocean grid point is registered independently of the other ocean grid points in the subdomain. Every fish or shark move, birth, and death is saved per ocean grid point and all the partial state changes describe the state evolution of the complete subdomain.

5.3 Results

The first test implementation of Wa-Tor (as an ACA with Time Warp synchronisation method) was performed on a 64-node Meiko Computing Surface, written in C with use of CSTools CSN communication primitives. Recently, the Wa-Tor implementation has been migrated to a Parsytec GCel-3/512, a 512-node distributed T805 transputer machine. This implementation is written in C with use of the Parix parallel programming environment. The implementation on the Parsytec GCel implies the availability of a new generation parallel architecture and parallel programming environment, but more important, it enables a seamless transmigration to the Parsytec GC, a T9000 transputer parallel architecture. With the T9000, virtual processors and virtual communication channels will be available that facilitate the efficient implementation of dynamic loadbalancing and irregular communication patterns, which will be important to achieve significant performance.

The parallelism analyser is completed and is available as a stand-alone tool. It analyses the event trace of the parallel simulation execution generated by the Time Warp run-time system. The output of the parallelism analyser is among other things the average parallelism, the minimum and maximum parallelism, and a parallelism profile. These are characteristics of the application itself, i.e. the Wa-Tor problem, and not of the typical execution on a specific parallel machine with use of Time Warp.

The Wa-Tor (ACA with Time Warp) implementation is completed on the Parsytec GCel. Performance measurements will be made and compared with the results of the parallelism analyser to assert the efficiency and effectiveness of the Time Warp method. Also the trade-off between incremental state saving and the check point method will be studied.

6 CONCLUSIONS AND FUTURE RESEARCH

The described ACA model is a generalisation of the SCA model in that it relaxes the synchronous cell update and evolves in continuous time. By this generalisation problems solved with use of ACA can be modelled closer to reality. The parallel implementation of ACA incorporates the Time Warp method with some extensions to synchronise the simulation time of the asynchronous processes. These extensions are *incremental state saving* and *direct event cancellation*. The proposed ACA model and the Time Warp method with the introduced extensions are validated and verified by an implementation of the adapted Wa-Tor problem.

Future research is directed to the formal symbolic description of ACA and Time Warp. The stochastic characteristics of the ACA and Time Warp will be described by Markov modelling. With the resulting model a characterisation is obtained of the applications that can be successfully solved with ACA and Time Warp.

REFERENCES

Agre, J.R. and P.A. Tinker (1991). Useful extensions to a Time Warp simulation system. In *Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, Anaheim, CA, pp. 78–85.

Dewdney, A.K. (1984, December). Computer recreations. *Scientific American 251* (12), pp. 14–22.

Ingerson, T.E. and R.L. Buvel (1984, January). Structure in asynchronous cellular automata. *Physica 10D* (1&2), 59–68.

Jefferson, D.R. (1985, July). Virtual Time. *ACM Transactions on Programming Languages and Systems 7* (3), 404–425.

Lomow, G., S.R. Das, and R.M. Fujimoto (1991). User cancellation of events in Time Warp. In *Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, Anaheim, CA, pp. 55–62.

Lubachevsky, B.D. (1987, December). Efficient parallel simulations of asynchronous cellular arrays. *Complex Systems 1* (6), 1099–1123.

Lubachevsky, B.D. (1988, March). Efficient parallel simulations of dynamic Ising spin systems. *Journal of Computational Physics 75* (1), 103–122.

Overeinder, B.J., L.O. Hertzberger, and P.M.A. Sloot (1991, May). Parallel discrete event simulation. In W.J.□Withagen (Ed.), *Proceedings of the Third Workshop Computer Systems*, Eindhoven, The Netherlands, pp. 19–30.

Overeinder, B.J., P.M.A. Sloot, and L.O. Hertzberger (1992, September). Time Warp on a transputer platform: Pilot study with asynchronous cellular automata. In M.□Valero et al. (Eds.), *Parallel Computing and Transputer Applications*, Barcelona, Spain, pp. 1303–1312.

Zeigler, B.P. (1976). *Theory of Modelling and Simulation*, John Wiley & Sons, New York.