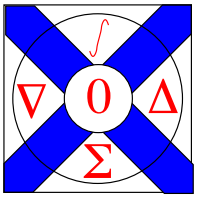

Computational Science Technical Note **CSTN-009**

Parallel Synchronization Issues in Simulating Artifical Life

H. A. James and C. J. Scogings and K. A. Hawick

2004

Simulating artificial life is a computationally demanding task as quite large systems need to be modelled for long timescales but over many different starting conditions to ascertain useful reductionist measurements from a model. It is therefore valuable to be able to incorporate parallelism to speed up the calculations. However incorporating concurrency into a model implies a knowledge of the mening of time and synchronisation properties of a model. In this paper we discuss the implications of local and global synchronisation in models for artificial life. These issues affect the possible ways that paralleism can be incorporated into our simulation models.

Keywords: artificial life, simulation, parallel computing; synchronisation

BiBTeX reference:

```
@INPROCEEDINGS{CSTN-009,
        author = {H. A. James and C. J. Scogings and K. A. Hawick},
        title = {Parallel Synchronization Issues in Simulating Artifical Life},
        booktitle = {Proc. 16th IASTED Int. Conf. on Parallel and Distributed Computing
                and Systems (PDCS)},
        year = {2004},
        editor = {Teofilo Gonzalez},
        pages = {815-820},
        address = {Cambridge, MA, USA},
        month = {9-11 November},
        publisher = {IASTED},
        note = {ISSN 1925-7937; ISBN 0-88986-421-7},
        keywords = {artificial life, simulation, parallel computing; synchronisation}
}
```


Prof Ken Hawick, Computer Science, Massey University, Albany, North Shore 102-904, Auckland, New Zealand.

PARALLEL SYNCHRONIZATION ISSUES IN SIMULATING ARTIFICIAL LIFE

H.A. James
Inst of Info & Math Sci
Massey University – Albany
North Shore 102-904
Auckland, New Zealand
email: h.a.james@massey.ac.nz

C.J. Scogings
Inst of Info & Math Sci
Massey University – Albany
North Shore 102-904
Auckland, New Zealand
email: c.scogings@massey.ac.nz

K.A. Hawick
Inst of Info & Math Sci
Massey University – Albany
North Shore 102-904
Auckland, New Zealand
email: k.a.hawick@massey.ac.nz

ABSTRACT
Update methods are an important aspect of the burgeoning Artificial Life research area. Artificial Life models, like the Predator-Prey model, are able to operate quite efficiently when implemented in a sequential manner only while population numbers are low to moderate. We find that for large populations sequential implementations are too slow to extract meaningful measurement statistics. In this paper we discuss the parallelisation of sequential update methods for use in Artificial Life systems. We also discuss the ramifications that parallel update algorithms introduce to data dependencies and also the meaning of correctness in parallel models.

KEY WORDS
parallel update; concurrency; artificial life; simulation.

1 Introduction

The developing field of Artificial Life (ALife) [1] has seen a renewed interest in simulating real life with simplified models [2–5] that researchers are able to control with more precision. However, these simplified models are showing a surprising amount of detail and complexity.

Typical implementations of ALife models are serial: a piece of sequential code is used to create a system and to serially update all the elements (agents) in the system until a final state has been reached. Unfortunately serial implementations suffer from the problem of becoming overwhelmed if there is a population explosion: the serial update process becomes slower and slower as the number of elements to update increases.

This paper describes a simple Predator-Prey model that we have developed in our on-going process of developing algorithms to support realistic ALife models. Described in the next section, our ALife model uses realistic probabilistic rule matching in a real coordinate space to implement the Predator-Prey system.

The paper's main contribution is a discussion on the effect that parallelising the implementation has on the model itself, and the performance that can be achieved through parallelisation. We discuss the need to state explicitly any assumptions that are made during the implementation of a parallel model. We also consider the ramifications of incorrectly implemented parallel update algorithms on the correctness of the resulting code using examples of the Eden/Epidemic model [6, 7]. This model is easier to analyse, and we demonstrate algorithm "sweep" effects in it as well as in our own model.

[1] In Proc. 16th Int. Conf. on Parallel and Distributed Computing and Systems (PDCS) 2004. Also Technical Note CSTN-009.

2 A Real-Space Predator-Prey Model

A simple prey-predator model has been constructed and is under investigation [8]. This type of model contains two groups of "animals" - the predators and the prey. Animals of both types move, breed and die after living a maximum life span. Predators eat prey if they can catch it. Predators that have not eaten prey within a certain time period die. Prey do not need to eat as the current simplistic model assumes sufficient natural resources to sustain life.

The model is executed as a sequence of **cycles**. During one cycle, the state of every animal is updated and thus one complete model state is constructed. In a specific model, the cycle might correspond to a year or a month but in this theoretical model it is simply referred to as "a time step".

The **state** of an animal is updated by applying **rules** to the current state. The current state of an animal is contained in a number of variables that are used to record information such as: location, age, hunger, number of prey neighbours, number of predator neighbours.

There is a different set of rules for each animal type (in this case, predator and prey). Typical rules include: predator will move towards prey if hungry; prey will move away from predator if adjacent; and animal will breed if adjacent to another animal of the same type.

Each rule has a priority and they are always applied in **priority order**, e.g. prey rate "moving away from predators" higher "than breeding". Most rules have a condition, i.e. the rule will only be applied under certain circumstances. Thus, for each animal, in each time step the list of rules is checked in priority order and as soon as a rule can be applied, it is and no further rules are checked.

There are two ways of updating the state of such models – simultaneous update or sequential update. **Simultaneous update** lends itself to a **parallel implementation** but is difficult and time consuming to implement in a sequential program as it requires two states to be maintained for the model at all times – the "current state" and the "future state" where the future state is constructed by applying rules to the current state. At the end of each time step, future state becomes the current state and the process is repeated.

Our initial model used a **sequential update** because it is faster (on a sinple processor system) and requires only a current state which is constantly changing as the state of each animal is changed. One possible drawback of this system is that certain animals are updated prior to other animals. In order to ensure that this does not consistently advantage one particular group, the animals are updated in a **random order** which is changed at the end of each time step.

A successful model is one in which a stable environment can be created enabling many cycles to pass before the collapse of one, or both, of the predator and prey populations. This success is dependent on 5 key parameters: predator maximum age; predator hunger threshold; predator birth rate; prey maximum age; and, prey birth rate.

Although each of these parameters is important in its own right, it is the combination of them that creates a successful, or otherwise, model environment. For example, a high prey birth rate will ensure a rapid increase in the number of prey animals, but a low predator birth rate will also increase the number of prey animals – since less predators means less prey are eaten.

In an attempt to test each of the combinations of the above parameters for convergence – to test whether the model enters a stable (or semi-stable) state, the cross product of the model's parameters are being executed across many different random starting configurations. Early tests [8] showed that the majority of our simulations in which the predator birth rate was independent of the prey birth rate lead to unstable models in which the prey died out (followed by the predators). Subsequently we have linked the predator birth rate to the perceived population of prey, producing many stable models. We have used this adapted model as a basis for the studies reported in this paper.

3 Concurrent Updates

In each time step of the model, the state of every agent is updated. There are two basic approaches to updating the state of such models: **sequential update** or **two-phase update**. Both approaches also require agents to be ordered and this could be a random order or a fixed order. Thus four update techniques are derived: sequential ordered update (also called an *in-situ* sweep update); sequential random update; two-phase ordered update; and two-phase random update.

During a sequential update, every agent is updated *in-situ*. This means that if an agent is updated after its neighbour then it is aware of any changes that have occurred in the states of other agents within the same time step. The order in which the agents are updated can be random or fixed (sweep). Iterating through the population in a fixed order is computationally efficient but introduces some definite **sweeping behaviours** that are not necessarily physical nor "correct" in the sense of being what was intented by the modeller. When the model uses sequential updates, each agent requires only a current state which is updated at the appropriate time and is also viewed by other agents (during their updates) before and after it is updated. Thus the storage requirements for a model with a huge population is minimised.

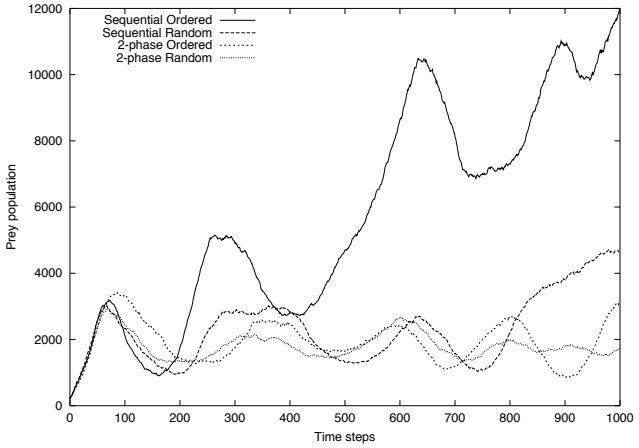

Figure 1. Effects on population of different update algorithms when applied to our simple Predator-Prey model over 1000 time steps.

However, when tested over *1000* time steps, the sequential update methods proved to be the most **unstable**, in that the number of prey agents rapidly increased with little sign of the population settling into a long term (stable) pattern. This is shown in figure 1, which shows that the two-phase update models are far more stable than the sequential models and that the two-phase random model is the most stable followed by the two-phase ordered model. The sequential ordered update method was less stable than the sequential random update.

Figure 2 shows the effects when the sequential sweep algorithm is applied to a single central infected cell with infection (or growth) probability $p = 1.0$. The **sweep** is a row-major raster. Due to the *in-situ* updating and the **sweeping** effects, the infection is **propagated** very quickly to all neighbouring cells that are updated after an infected cell. This is analogous to information travelling across the model at the speed of light. When the probability of infection is substantially less than 1, $p = 0.25$, the **skewed** results of the model are more subtle. Figure 3 shows the 'correct' result on the left and the skewed results on the

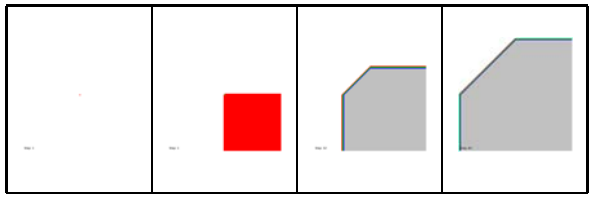

Figure 2. The effects of a sweeping *in-situ* update algorithm for the Eden Epidemic Model 128×128 across successive time steps (cells: white empty; dark live; grey dead, Infection probability 1.0). The simulation starts with a single infected cell at the centre and progresses to the right.

right. The correct results have been produced using a two-phase update algorithm.

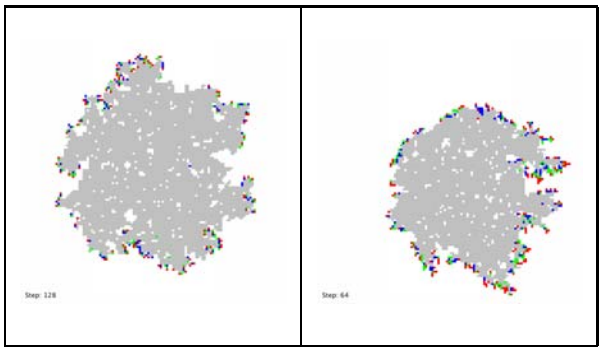

Figure 3. Comparing the Generational two-phase update algorithm (left) with a sweeping *in-situ* update algorithm (right) for the Eden Epidemic Model 128×128 across successive time steps (cells: white empty; dark live; grey dead. Infection probability 0.25)

If constrained to a sequential update method a better way of updating the system is to randomise the choice of sites to update. This can either be done by **randomly shuffling** the list of sites to update (perhaps using a pair-wise shuffle). An even more random approach can be taken by performing Monte-Carlo hits on the sites: on average all sites will be updated once every n time steps (where n is the number of sites in the system), but as the update sites are being chosen randomly, there is a possibility some sites will be updated more frequently than others over a short time period. This has the effect of slightly blurring the concept of 'time' in the simulation (see figure 4).

Figure 4a) illustrates the cellular growth behaviour of a variation of the Eden Epidemic model [7] when a single infected cell at the centre of the pattern infects nearest neighbouring cells with probability $p = 0.3$ at each time step. In the model shown, infected cells die after two time steps after being infected. Figure 4b) shows how a random algorithm can recover spatial symmetry in the growth model.

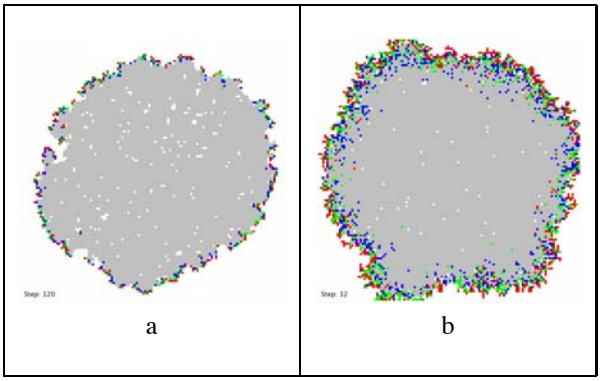

Figure 4. A variation of the Eden Epidemic model is used to show growth time scales and symmetries on a square lattice. Sites are infected from any live nearest neighbour with a probability $p = 0.3$ (left) or $p = 1.0$ (right), and once infected, die after two time steps. The cluster is grown from a single central infected cell. The left hand cluster illustrates the two phase update algorithm (a) and the right hand (b) uses a random algorithm whereby cells are essentially updated with mean probability of once per time step. Some cells are hit more often and although spatial symmetry is largely recovered, the time scale is accelerated.

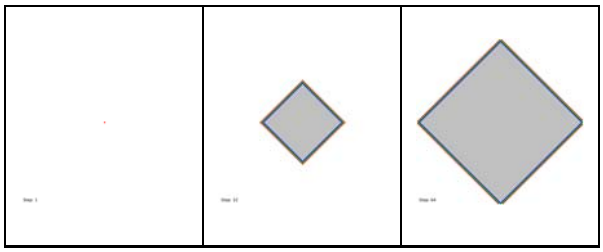

Figure 5. The Generational two-phase update algorithm for the Eden Epidemic Model 128×128 cells: white empty; dark live; grey dead. (Infection probability 1.0). The simulation starts with a single infected cell at the centre and progresses to the right.

4 Parallel Issues

A **two-phase update** simulates updating all agents in parallel. Each agent requires two states: a current state and a future state. In phase 1, each agent uses its own (and other) current states in order to construct the future state. Phase 2 occurs at the end of each time step when the newly constructed future state becomes the current state for each agent. When $p = 1.0$ the ordered two-phase algorithm is expected to produce updates as shown in figure 5. Because the new system state is generated from a copy of the current state, skewing effects are not exhibited.

However the two-phase update can lead to even more subtle anomalies. For example, if two predators are adjacent to one prey agent, the correct outcome should be that one predator eats the prey and the other predator goes hungry. However, in a two-phase update model, both predators can eat the same prey because each predator recognises that it is adjacent to prey at the start of phase 1 of the update. Note that this problem does not arise in a sequential update model as the predator which is updated first would eat the prey and the other predator would not have adjacent prey at the time of its update.

The current solution to these problems with the two-phase update is to force part of the update to occur sequentially when required. For example, in order to solve the problem listed above, as soon as a predator eats prey, the current state of the prey is immediately updated *in-situ* (i.e. that prey is removed from the list of viable agents). This means that within a two-phase update model, part of the update is occurring sequentially and thus an order is required for that part of the update. Hence it becomes possible to describe the **two-phase random update** and the two-phase ordered update. We stress the need to explicitly state any assumptions that are made by the architect of such an ALife model system (and hence the update algorithms that will be applied) especially when rules are introduced to remove conflicts.

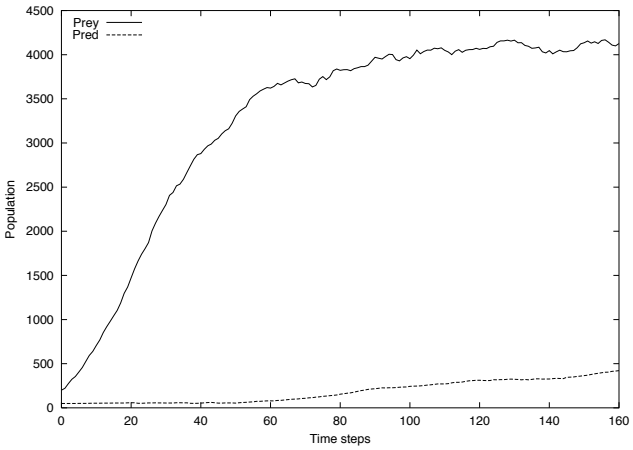

Figure 6. Predator and prey populations in our simple model run for the first 160 time steps.

Figure 6 shows predator and prey populations in our model when run using a two-phase ordered update. As expected, the numbers of of predators and prey rise and fall cyclically. This is due to prey being predated and also groups of prey breaking away from the predators and having the opportunity to reproduce over time. As the model executes more and more agents are added to the system. These place an increasing burden on the processor, leading to an increased time to process each time step. Figure 7 shows the average time to execute 10 time steps using different numbers of parallel processors from a single processor system to a system using 60 worker processors (the master process spends most of its time organising updates and performing communications).

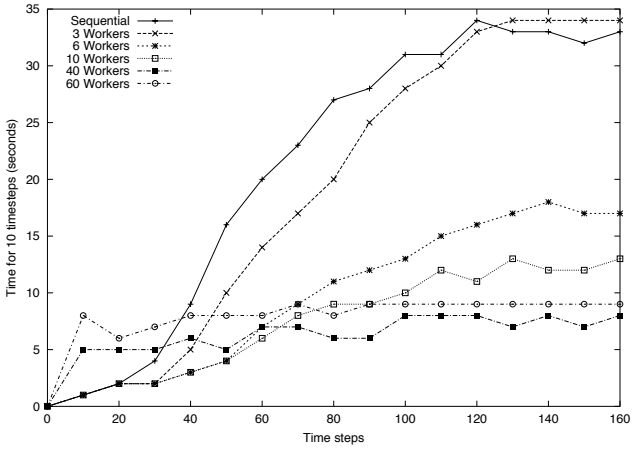

Figure 7. Average time to execute 10 time steps using the parallel two-phase update algorithm across varying numbers of parallel processors.

By taking the average number of time steps completed per second across varying numbers of processors (figure 8) we can generate a graph that shows where the ratio of computation to communications becomes smaller. This figure shows that it is most beneficial to use between 40 and 42 worker processors for our model and that adding processors after this point does not add any further value.

5 Phase Lag Effects

It is difficult to do any sensible trend analysis on the populations of predators and prey without first removing some of the jitter in the data caused by the model's randomness. As previously mentioned, raw population figures are shown in figure 6. It can be seen that the populations behave in a vaguely cyclic manner but not much more information can usefully be gained from looking at the graph.

We use a low-pass Fourier filter to remove some of the transient signals from the population results. A Fourier transform converts the time-based data into frequency-based data and the low-pass filter removes the higher-frequency signals. Converting the attenuated frequency-

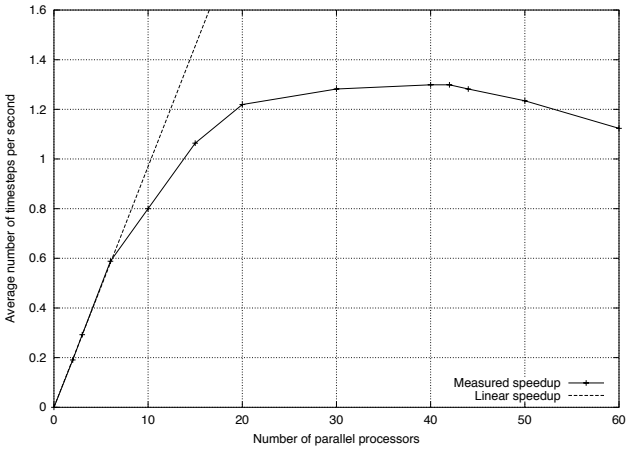

Figure 8. Average number of time steps per second for populations in a stable state across varying numbers of parallel processors (solid line). The dotted line indicates linear speedup, which is exhibited by our model up to seven processors (six workers).

based data back to time-based data provides us with only the low-frequency (underlying) data. The graph is shown in figure 9. It can be seen that a matching set of peaks and troughs exist in both the predator (top) and prey (bottom) curves. The two matching curves are slightly off-set in phase, showing the effects of basing the predators' birth rate on the current population of prey.

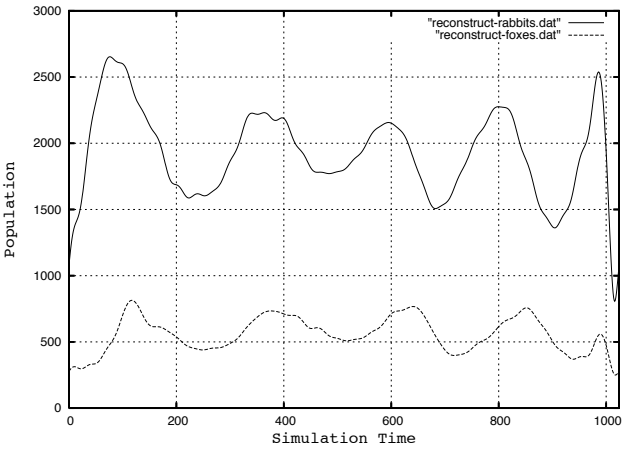

Figure 9. Predator and prey populations after the data has been passed through a low-pass Fourier filter.

6 Summary and Conclusions

We have discussed the relative ease of implementing a simple ALife model using a serial execution environment. We have also discussed the problems with applying a naive sweeping update algorithm: inconsistencies arise between the previous state and current state of agents. We find the models produce far more consistent results when a two-phase update is used or at the very least the sweep update is randomised.

Two-phase updates are preferable because they lend themselves to a greater degree of parallelism than a simple randomised sweep update. This is because the two-phase update algorithm maintains a complete copy of the present state of the system while generating the new state; parallel processors can compute the new state with complete independence from other processors, whereas in the sweep update processors must broadcast their agents' new state to all other processors asynchronously. Parallelism is vitally important for our goal of scientifically studying the characteristics of large populations with varying model parameters.

The view maintained by the serial model is based on a list of agents, which means that agents who are neighbours in the list could be far away from each other in terms of the model's geography. In parallelising the model we have considered different ways in which the model can be split between processors: geographically and also on a per-agent basis.

We believe we have identified the major properties of a suitable parallel algorithm of our ALife model updates. Having successfully parallelised the simple Predator-Prey model and shown that the parallel version does converge to the same results as the serial, we will be measuring the characteristics of the populations thus generated. We are also looking to incorporate evolutionary aspects into this model so that we can allow different agents to evolve different behaviours such as specialisation. We will also be experimenting with a variation of this model removing the assumption that prey always have food available.

Exploring these models in a scientifically systematic manner involves making measurements of many different runs of large models. Parallelism is therefore crucial to this programme.

Acknowledgements

The authors gratefully acknowledge the contribution of Helix supercomputer time by Massey University and the Allan Wilson Centre in the production of the frameworks and data reported in this paper.

References

[1] Levy, S. "Artificial Life" Penguin Books, 1992. ISBN 0-14-023105-6

[2] Adami, C. Avida (Digital Life Laboratory) Available at http://dllab.caltech.edu/avida

[3] Holland, J. ECHO Available from http://-www.santafe.edu/projects/echo/echo.html

[4] Ray, T. Tierra Available at http://www.isd.adr.co.jp/-~ray/tierra

[5] Wilson, S. The Animat Path to AI in *From Animals to Animats 1: Proceedings of The First International Conference on Simulation of Adaptive Behavior*, (pp. 15-21); Meyer, J-A. & Wilson, S. (eds), Cambridge, MA: The MIT Press/Bradford Books (1991).

[6] Miramontes, O. and Luque, B. "Dynamical small-world behaviour in an epidemical model of mobile individuals", in Physica D, 168-169 (2002), pp 379-385.

[7] Eden, M. "A two-dimensional growth process." In Proceedings of Fourth Berkeley Symposium on Mathematics, Statistics, and Probability, volume 4, str. 223-239. University of California Press, Berkeley, 1960.

[8] James, H.A., Scogings C.J. and Hawick, K.A. "A framework and simulation engine for studying artificial life" in Research Letters in the Information and Mathematical Sciences, Vol 6, May 2004, pp143–155, ISSN 1175-2777. Available from http://-iims.massey.ac.nz/research/letters/volume6/