Open Systems From a Dynamical Viewpoint: Some Examples

Albert Boulanger
BBN Laboratories Inc.
10 Moulton St.
Cambridge MA 02238

Extended Abstract

1. Introduction

Carl Hewitt and his group at MIT [Hewitt 84] have coined the term "open system" for large, distributed systems of computation which are open-ended and nonstationary. This paper will redefine the term "open system" to include the notion of open system in thermodynamics, i.e., coupling with an external environment. In this revised definition, as it will be shown below, the computational outcome of an open system is tied to the environment in which it is embedded.

The outcome of an open computation depends on the environment because the system as a whole is a dynamical system. The asynchronous nature of the distributed computation of the individual components means that the time evolution of each component of the computation is relative to the other components with which it couples. In one of the open system computations presented below, it will be shown that, because of the ever-present coupling at the level of hardware, interaction goes beyond the immediate software level of the application to all levels of software and hardware and potentially to an external environment. In a tightly synchronized parallel or serial computation the dynamics of such coupling is normally irrelevant to computation because temporal precedence is known.

Normally, the nonlinear nature of a computation where logic and decision making is present ***can be ignored.*** Thus, we can forget that the implementation of logic in a transistor is its nonlinear saturation region and make use of the macroscopic nonlinear switching behavior of transistors to build a computer engine to manipulate numbers and symbols. The nonlinear aspect of computation rears it head in an open system because dynamics at all levels of hardware and software can affect computation. The dynamics of computation in a distributed system is ***inherently*** non-linear. In addition to the nonlinear aspect of computer logic, the time delays introduced in relaxing temporal precedence contribute to the dynamics.

Two examples of work that I have done with open system computation will be presented. The first example illustrates some of the phenomenology of the asynchronous time evolution of each component of

a parallel computation. A formalism based on time-delay equations from the theory of dynamical systems will be suggested for understanding the behavior of this computation. The second example illustrates open system coupling. This paper will conclude with an outline of future investigations. Two appendices will be included. One is a selected survey of the dynamical behavior of open system computation that has appeared in the literature. The second appendix is a short tutorial on dynamical system theory.

2. Example 1: The Orbiting Electron

There is much folklore about observer-observed interactions in concurrent systems. This observer-observed interaction has been termed the *probe effect* by Gait. In [Gait 86], an example is presented where an erroneously synchronized concurrent program to compute the maximum of a list of numbers actually performs correctly for certain values of delay introduced by a monitor utility between statements of the concurrent program. Furthermore, there are wild swings in the percentage of correct answers with the length of delay until a certain delay is reached where it is always correct.

It is easy to construct simple open systems where the act of doing an observation effects the outcome of the computation. Consider the equations[1] for an orbiting electron in Figure 1. Suppose we place the (discrete time version) momentum equation on one processor and the position equation on another. Suppose further that the two processors run without synchronizing to each other. It is clear that the dynamics (the behavior of the two processors in time) will contribute to the trajectory of the simulated electron. Suppose further that obtaining information about momentum slows down the momentum processor, and getting information about position slows down the position processor. It is now clear that this process of observation affects the outcome of the computation. The varying of the trajectory of the electron due to internal fluctuations as well as delays due to observation can be seen in Figure 2[2].

One surprising discovery was that there where orbits were time delay does not influence the trajectory. This occurs for all circular orbits. In this case[3] the orbital radius is a constant and the angular velocity is a

---

[1]These equations come from a *variational* formalization of mechanics known as the *Hamiltonian formulation* that is typically introduced in an advanced mechanics course. The equations of motion are derived from a principle of *least action* -- nature evolves in a way that minimizes action. First the action or total energy of the system is formulated. In the present example, this is called the *Hamiltonian*. Then the momentum and position are derived by the relations: $\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i}$ and $\frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i}$ where p_i and q_i are the momentum and position of the ith degree of freedom of the system. This formalization of mechanics is general enough to be maintained as a tool in both relativity and quantum mechanics. See [Lanczos 66] for an excellent treatment of the subject

[2]These were simulations implemented with multitasking on the lisp machine.

[3]When $p_\phi^2 = me^2r$.

From Messiah Vol I[Messiah 58], Page 34, EQN I.15

For an electron of mass m in 2d with a potential of:

$$\frac{e^2}{r}$$

The equations for the conjugate variables, position and momentum, in polar coordinates:

Momentum equations:

$$\frac{dp_\phi}{dt} = 0 \quad , \quad \frac{dp_r}{dt} = \frac{p_\phi^2}{mr^3} - \frac{e^2}{r^2}$$

Position Equations:

$$\frac{d\phi}{dt} = \frac{p_\phi}{mr^2} \quad , \quad \frac{dr}{dt} = \frac{p_r}{m}$$

Classical Hamiltonian:

$$H = \frac{1}{2m}(p_r^2 + \frac{p_\phi^2}{r^2}) - \frac{e^2}{r}$$

Discrete version:
Momentum:

```
ptheta = constant
pr     = pr + ptheta^2/mr^3 - e^2/r^2
```

Position:

```
theta  = theta + ptheta/mr^2
r      = r     + pr/m
```

Figure 1: The equations of motion for an orbiting electron.

constant. In such a circumstance, no delay in either the momentum processor or the position processor can affect a change in the circular orbit. Preliminary simulations using Scheme on the Butterfly computer indicate that actual fluctuations in processor timing cause many trajectories to settle down into this stable orbit. This may be a limit cycle attractor[4] of the asynchronous simulation.

A research topic in dynamical systems that may characterize the behavior of the above example concerns time-delay systems. To see this, note that the discrete equations for the orbiting electron could be modeled as two coupled recurrence relations with fluctuating time delays:

[4]The notion of an attractor comes from the theory of dynamical systems. An attractor is what the dynamical behavior of a system settles down to, or is attracted to. For an attractor there exists a *basin* of attraction which is the region of state space where all trajectories with initial conditions in that region go to the attractor. A common attractor is the *point* attractor. As its name suggests, this attractor describe a system that settles down to a point in phase space in its trajectories. This is a common attractor and occurs in both linear and non-linear systems. Another common attractor is the *limit-cycle* attractor and occurs with sustained oscillation, as in a organ pipe or an electronic oscillator. See the appendix for a short tutorial on dynamical system theory.

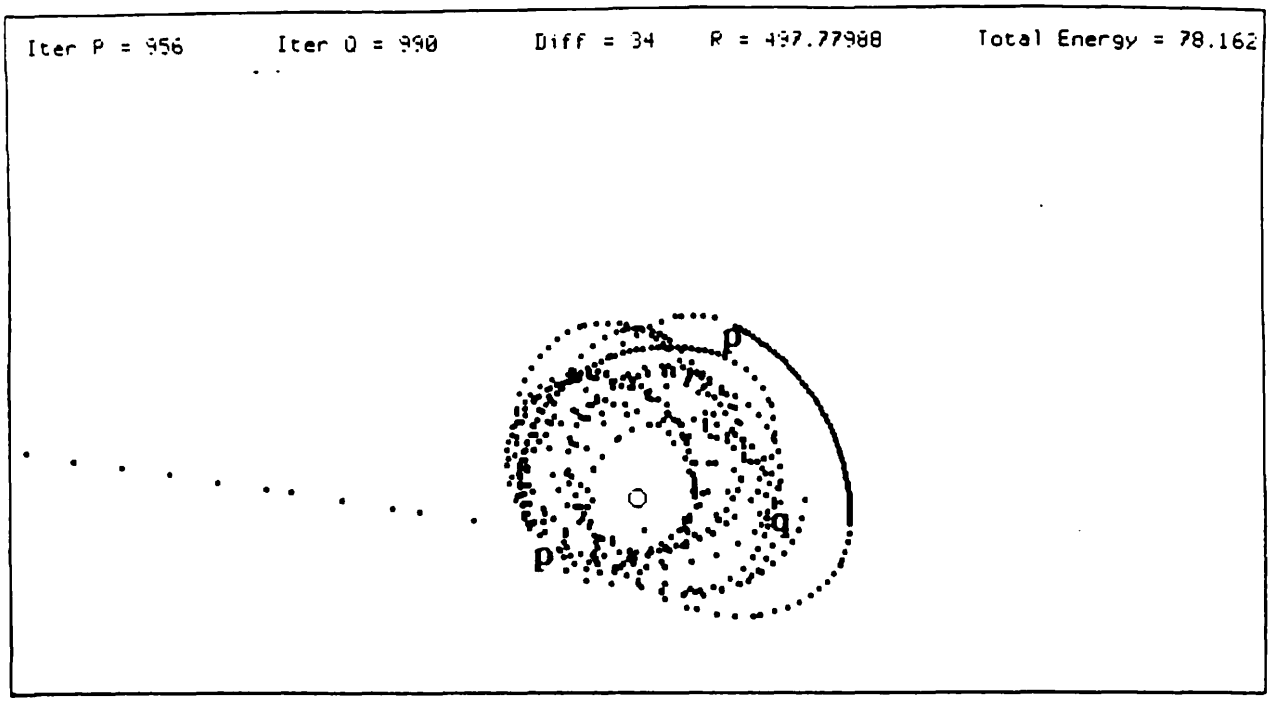

Figure 2: This is a typical result from the asynchronous simulation of an orbiting electron. Note the jumbled trajectory due to timing differences of the two processes. Also note the affect of delay due to observation of position at the point labeled "q" and momentum at the points labeled "p".

$$P_{t+1} = P_t + f_p(P_t, Q_{t-\varepsilon_p}) \ .$$

$$Q_{t+1} = Q_t + f_q(Q_t, P_{t-\varepsilon_q})$$

where ε_p and ε_q are random deviates that could be characterized as distributions which are functions of the task size and machine architecture[5].

Researchers have investigated delay differential and delay difference equations. Delay difference equations are appropriate models for the dynamics of open systems that have time delays between the interacting agents.[6] So far, studies only investigated fixed delay. I know of no work where delay is allowed to fluctuate. Jack Hale at Brown has done theoretical work on delay equations. Doyne Farmer [Farmer 82] has done an empirical study of delay differential equations. For the delay differential equation that he investigated[7], small delays exhibit a fixed point attractor. With larger delay, a limit cycle attractor appears which then becomes a period-doubling-bifurcation sequence. Finally, chaos is reached. Within the chaotic regime, there are intermittent returns to limit cycles. He found that the dimension of

---

[5]One issue is the systematic difference in computation times a distributed systems will normally have. Then one of the delayed variables, say P, would be $P_{t-r_q t-\varepsilon_q}$, where r_q is an average slow-down ratio.

[6]See [Huberman 88] for other work treating open systems as delay equations.

[7]He studied a model of blood supply in the body due to Mackey and Glass

the chaotic attractor to increase linearly with delay. It is speculated that similar behavior occurs in open system calculations like the asynchronous orbiting electron example.

3. Example 2: A Parallel Random Sequence Generator

In discussions about the Butterfly computer with its designers, I developed a gedanken algorithm to help characterize its asynchronous nature. Several interesting facts came out of these discussions:

- The Butterfly computer is asynchronous at the task level but *synchronous* at the clock level.
- The Butterfly switch uses a random backoff delay to resolve switch contention. The random delay is generated by a hardware-implemented feedback shift-register.
- One chief designer's prediction of how the Butterfly processor behaves with the simple asynchronous gedanken algorithm is completely different from actual behavior.

The above gedanken algorithm forms the basis of a random sequence generator involving **n** processors described below:

Process i | i= 1 .. n-1

```
while still-more-to-do do
  Toggle state_i between {0,1}
enddo
```

Process n

```
For j = 1 to vectorlength do
begin
  Toggle state_n between {0,1}
  For k = 1 to n-1 do
    outputvector[j] = outputvector[j] ⊕ state_k
  enddo
end
enddo
still-more-to-do = false
```

Note that there is no synchronization between the processors. It was observed that this algorithm produces a sequence that is highly non-random with a few random "hiccups" scattered about using a few processors, but quickly gets more random with more processors. Initially, it was suspected that the random backoff was contributing to the random behavior of the composite algorithm, but when contention problems were dealt with one-by-one, the random behavior persisted. The program execution in time was profiled with the Gist tool[8] available for the Butterfly computer, but that yielded little insight. Finally, one of the designers mentioned that since the dynamics of the processes is critical to the behavior

[8]This tool allows one to view graphically the temporal layout of a program by a user specified event recording mechanism.

of the computation, the dynamics of the lower level operations of the processor board become important: the system was ***dynamically coupling*** to the lower levels of hardware and software as well as to its external environment[9]. This coupling happens because the outcome of the computation is sensitive to the dynamics of the composite of algorithm/support-software/hardware; these dynamics from the substrate for the asynchronous computation.

Several tests were made of the random sequences generated by this asynchronous algorithm. To get a lower bound on the recurrence length of the algorithm using two processors, a random sequence of length 60000 was generated and an autocorrelation test run on it. It showed no significant correlations and no recurrences. Thus at least 15 bits of state *beyond* that of the algorithm's state (2 bits) are being coupled into the system. Another test consisted of sliding an n-bit window through the sequence and measuring the equi-partitioning of the patterns. For example, each partition (out of four) of a two bit window should have around 25% of the patterns. This was characterized using chi-square. Successively larger windows were used until sample counts became too small. The chi-square values became acceptable at around seven processors. See Figure 3.

Finally, note that scaling is helps to increase the randomness of this open system algorithm. For every bit of state the system couples with, there is a potential doubling in the number of states the composite algorithm has access to. This exponential trend towards the ideal system of true deterministic randomness (a chaotic system with access to numbers of infinite precision) is the essence of the concurrent-systems sense of nondeterminism.

4. Further Research

The above two examples illustrate some of the phenomenology arising from open distributed computation. These systems are too complicated to analyze in a formal way using the tools from dynamical systems. One of the big road blocks is the characterization of time-delay equations with variable delay. In order to simplify the system enough to begin to characterize what fluctuations in time delay do to asynchronous computation, I plan to look at a class of algorithms that are known as

---

[9]The external environment influence is probably not that great for the Butterfly. It would be much greater in a distributed implementation. In such a situation, thermal drifts of the processor clocks can influence the computation and we literally have an open thermodynamic system.

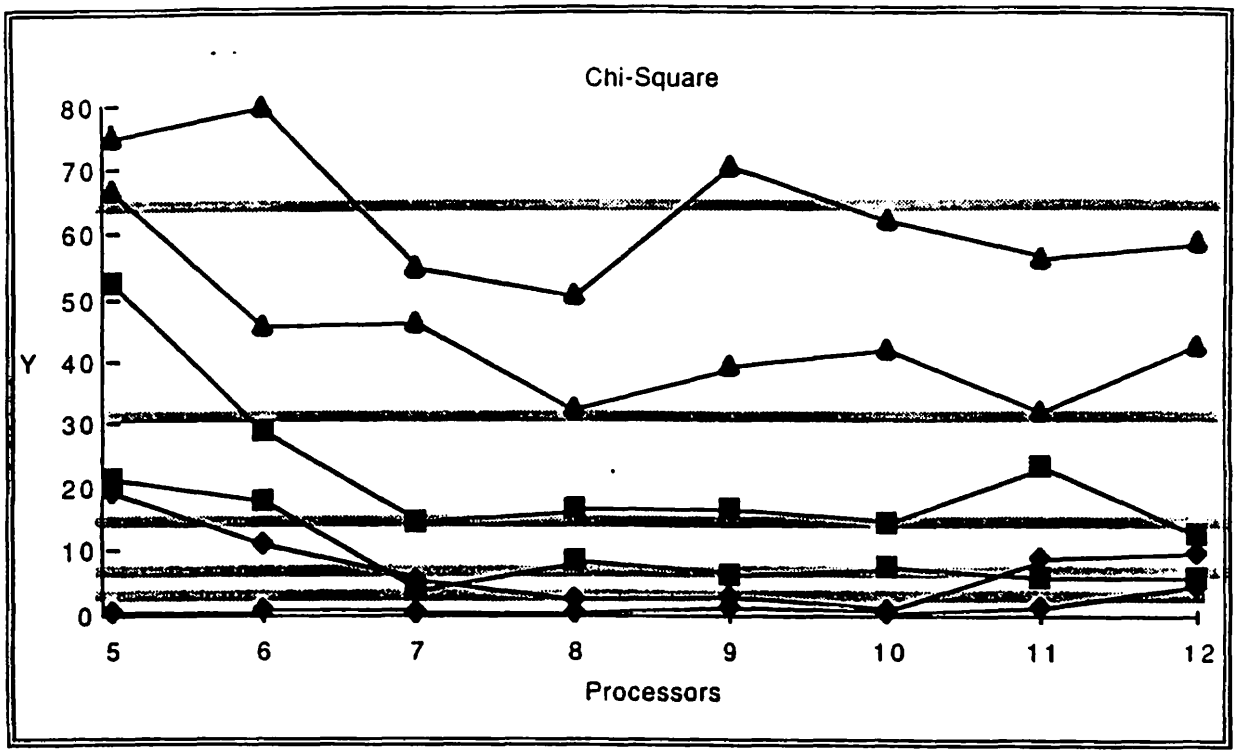

Figure 3: Chi-square values for the sliding-window test of the random sequence vs. number of processors. Each line represents a different window size (from 1-6). The grey bars are the 50% values for chi-square with 3,7,15,31, and 63 degrees of freedom.

asynchronous iterative methods.[10]

For example, consider the iterative Newton-Raphson method for root finding:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

One can formulate an asynchronous iterative approach to this by placing the $f'(x_k)$ derivative evaluation on one processor and the rest of the calculation on another. This is a simple dynamical system where the fluctuations in processor timings will be studied and the basins of attraction will be developed to characterize the convergence of the related time-delay system. The relationship between the basins of attraction of the asynchronous method and basins of the synchronous one should be very interesting. Since the trajectories by which the basins will be developed vary from run to run, a probabilistic approach

[10]Several asynchronous formalisms of parallel iterative methods have been devised for optimization, shortest path, dynamic programming, root finding, the solution of systems of linear and non-linear equations, and the Jacobi or Gauss-Seidel iterative procedures for PDEs - in general where contraction and monotone mappings arise [Li 87], [Tsitsikilis 86], [Bertsekas 83], [Spiteri 86], [Lubachevsky 86], [Baudet 78], [Chazan 69], [Reed 85], [Wienke 86], and [Mitra 87]. See [Kronsjo 85] for an excellent discussion of these techniques. While some bounds for the convergence of such algorithms have been derived, their actual behavior is still poorly understood [Voigt 85]. See the appendix for other simple open systems, such neural networks, where researchers are beginning to look at time delays in their dynamics.

to their characterization will be taken.[11]

Finally, given the discussion of the asynchronous random sequence generator one could speculate that open systems with asynchronous clocks may have access to high-algorithmic-complexity[12] numbers that arise in the timing relationships of the distributed computational agents. Note that the representation for these numbers would not be *explicit* in a computer word, but implicit in the timing. However, this may only be academic, given the exponential scaling law discussed in Section 3.

5. Conclusion

It has been shown by argument and examples that open systems are dynamical systems and that the tools of dynamical system theory can be useful in understanding their behavior. A possible formalism of coupled time-delay recurrence relations where time delays are characterized as a distributions was proposed. Unfortunately, little is known about the dynamical behavior of time-delay equations with variable delay. A research plan is presented that looks at simpler systems to analyze time-delay equations with variable delay.

It was shown in the second example that fluctuations in timings of processes of an asynchronous computation can be use as a source of noise. It is challenging to find computational paradigms where asynchronization can help (either by speeding up computation or by obtaining better answers) the outcome of a distributed computation.[13]

---

[11]It should also be known that the basins of attraction of the synchronized algorithm exhibits a complicated fractal structure on the complex plane for polynomials greater than two. See [Saari 84] and [Peitgen 82] for more details.

[12]This is a measure of the randomness of an isolated number. It corresponds to the shortest algorithm that can generate the number. The theory of such random numbers is algorithmic information theory [Chaitin 87]. There is a deep relationship between algorithmic information theory and chaos. See [Ford 83] for more details.

[13]Current research in neural networks offers several examples and are included in the appendix.

References

[Baudet 78] Baudet, G.M. Asynchronous Iterative Methods for Multiprocessors. *Journal of the ACM* 25(2):226-244, April, 1978.

[Bertsekas 83] Bertsekas, Dimitri. Distributed Asynchronous Computation of Fixed Point. *Mathematical Programming* 27:107-120, 1983.

[Chaitin 87] Chaitin, G.J. Incompleteness Theorms for Random Reals. *Advances in Applied Mathematics* 8:119-146, 1987.

[Chazan 69] Chazan, D. and Miranker, W. Chaotic Relaxation. *Lin. Alg. and Its Applications* 2:199-222, 1969.

[Farmer 82] Farmer, J. Doyne. Chaotic Attractors of an Infinite-Dimensional Dynamical System. *Physica* 4D:366-393, 1982.

[Ford 83] Ford, Joseph. How Random is a Coin Toss? *Physics Today* :40-47, April, 1983. Very good introduction to the notions of chaos and deterministic randomness.

[Gait 86] Gait, Jason. A Probe Effect in Concurrent Programs. *Software-Practice & Experience* 16(3):225-233, March, 1986.

[Hewitt 84] Hewitt, Carl & Peter de Jong. Open Systems. In Michael Brodie, John Mylopouos, & Joachim Schmidt (editors), *On Conceptual Modeling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*, chapter 6, pages 147-164. Springer-Verlag, 1984.

[Huberman 88] Huberman, Bernado, & Tad Hogg. The Behavior of Computational Ecologies. In Bernado Huberman (editor), *The Ecology of Computation*. North-Holland, 1988. To appear.

[Kronsjo 85] Kronsjo, Lydia. The Root-finding Algorithm for a Non-linear Function, & Iterative Parallel Algorithms. *Computational Complexity of Sequential and Parallel Algorithms*. John Wiley & Sons, 1985, pages 123-144, Chapter 10,11. These two chapters go hand-in-hand.

[Lanczos 66] Lanczos, Cornelius. *The Variational Principles of Mechanics. 3rd Edition*. University of Toronto Press, Toronto Canada, 1966.

[Li 87] Li, Shu & Tamer Basar. Asymptotic Agreement and Convergence of Asynchronous Stochastic Algorithms. *IEEE Trans. on Automatic Control* AC-32(7):612-618, July, 1987.

[Lubachevsky 86] Lubachevsky, Boris and Debasis Mitra. A Chaotic Asynchronous Algorithm for Computing the Fixed Point of a Nonnegative Matrix of Unit Spectral Radius. *Journal of the ACM* 33(1):130-150, January, 1986.

[Messiah 58] Messiah Albert. *Quantum Mechanics*. John Wiley & Sons, 1958.

[Mitra 87] Mitra, Debasis. Asynchronous Relaxations for the Numerical Solution of Differential Equations by Parallel Processors. *SIAM J. Stat* 8(1):43-58, January, 1987.

[Peitgen 82] Peitgen H.O., D. Saupe, & F.v. Haeseler. Cayley's Problem and Julia Sets. *Mathematical Intelligencer* 6(2):11-20, 1982. This paper is referred to in section 2.1.3.

[Reed 85] Reed, Daniel & Merrell L. Patrick. Parallel, Iterative Solution of Sparse Linear Systems: Models and Architectures. *Parallel Computing* 2:45-67, 1985. An asynchronous iterative paradigm for solving sparse linear systems is presented and an ideal architecture to support it is given.

[Saari 84] Saari, Donald G., & John B. Urenko. Newton's Method, Circle Maps, and Chaotic Motion. *Am. Math. Month.* :3-17, January, 1984.

[Spiteri 86] Spiteri, Pierre. Parallel Asynchronous Algorithms for Solving Boundry Value Problems. In M. Cosnard et al. (editors), *Parallel Algorithms & Architectures*, pages 73-84. Elsevier, 1986.

[Tsitsikilis 86] Tsitsikilis, John, Dimitri Bertsekas, & Michael Athans. Distributed Asynchronous Deterministic and Stochastic Gradient Optimization Algorithms. *IEEE Trans. on Automatic Control* AC-31(9):803-812, September, 1986.

[Voigt 85] Voigt, Robert G. *Where Are the Parallel Algorithms?*. ICASE Report 85-2, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, Virginia 23665, January, 1985.

[Wienke 86] Wienke B. R. & R. E. Hiromoto. Chaotic Iteration and Parallel Divergence. In Feilmeier, M., G. Joubert, and U. Schendel (editors), *Parallel Computing 85*, pages 205-210. Elsevier, 1986.