

ASYNCHRONOUS DISCRETE EVENT SYSTEMS AND EMERGENCE OF COMPUTATIONAL CHAOS

Sarit Barhen* Jacob Barhen♦ Vladimir Protopopescu♦

Center for Engineering Science Advanced Research

Oak Ridge National Laboratory

Oak Ridge, TN 37831-6355

barhenj@ornl.gov

Abstract. Asynchronous computing environments provide an ideal framework for conceptual modeling and simulation of large scale, distributed discrete event systems. Such environments may, however, exhibit an aperiodic oscillatory behavior referred to as “computational chaos”, which impedes the correct processing of quantities of interest. In this paper, we illustrate the emergence of computational chaos from fixed point and limit cycle attractors for a simple network model. In particular, the complete Lyapunov spectrum associated with the network dynamics is computed, and conditions that prevent its emergence are briefly discussed.

Keywords: asynchronous computing, computational chaos, discrete event systems, Lyapunov spectrum.

1. INTRODUCTION

Over the past twenty years, distributed computation has emerged and developed into a very exciting area. Two driving forces are behind this development. One is the desire within the scientific community to solve, in ever greater details, highly complex problems such as controlling the behavior of materials at the molecular level [1], modeling the climate, or weather prediction. The other arises from ever more rapid progress in electronic circuit integration at the nanoscale, production of faster and larger memories, and the availability of very high bandwidth communication networks. This has resulted in the development of massively parallel systems that enable the solution of such problems, and, in turn, is opening the possibility of addressing new problems, hitherto considered intractable.

A distributed computing system is defined as a set of cooperating processes that evolve on multiprocessor architectures *without* a common memory [2]. Each processor and its local memory form a unit known as a *node*. Nodes are connected by physical communication

channels (networks), which allow any two processors to exchange information (directly or indirectly) via message passing.

The overarching paradigm of process cooperation to achieve a common goal has often been interpreted as requirement for processes to synchronize with each other. Since the inception of distributed computing (see [3] for historical references), considerable resources have been devoted to the development of efficient synchronization tools. Two approaches were followed. One consisted of maintaining a single process, the controller, which through message exchanges would coordinate the activities of the individual processes in the system. Such a “*centralized control*” solution clearly failed because (1) the precedence constraints implied by such message exchanges with the controller slowed down each process, and (2) if the node containing the controller failed, the entire system had to halt. The alternative approach, *distributed control*, installed a local process controller at each node. But, even here, local algorithms had to wait at predetermined points for predetermined messages to become available [4]. This gave rise to load imbalance across the system and often resulted in severe processor underutilization.

The concept of *asynchronous computing* emerged from an attempt to overcome these constraints, by creating a distributed environment that enables uncoordinated, system-wide activity, while ultimately producing a *correct* solution, i.e., a solution that would have been obtained had synchronization been enforced. One area that may directly benefit from such a paradigm is the modeling and simulation of *discrete event systems* (DES).

The DES paradigm deals with processes whose output is a dynamic function of time [5]. In a traditional discrete event specification (DEVS), one approximates the input, output, and state trajectories through piecewise constant segments defined over discrete time intervals of varying length. The accurate modeling of many realistic processes, however, had long been recognized to pose significant challenges to this conventional approach. The G-DEVS formalism [5] models the trajectories in terms of a piecewise polynomial representation, yielding higher accuracies in simulating continuous processes as

* School of Engineering and Applied Science, Washington University in Saint Louis, MO; ♦ also with the University of Tennessee (JB: Computer Science, VP: Mathematics), Knoxville, TN.

discrete event abstractions. Moreover, it enables the development of a uniform simulation environment for *hybrid* (i.e., both continuous and discrete) systems.

Recently, a *Discrete Event Calculus Model* (DECM) was proposed [6], where it is argued that the concept of *event* is more natural for real-time systems than the concept of *state*. Centering a formulation on the former, allows one to express *asynchronous behavior* without having to rely on the classical paradigm of state transition, which is typically a challenge for systems where the number of states is very large. In addition, DECM offers an explicit representation of time that allows the use of timed simulations for the validation of formal specifications.

In this paper, we focus on the concept of *concurrent asynchronicity* as implied by an uncoordinated, system-wide activity. In view of existing application challenges, there is a strong motivation to develop algorithms that can fully exploit such a behavior. One of the main reasons progress in this direction has been slow is that concurrent asynchronous relaxation algorithms usually give rise to an aperiodic oscillatory behavior. This long known phenomenon was originally referred to as *chaotic relaxation* [7] or *computational chaos*.

In the sequel, we first present some of the basic concepts underlying concurrently asynchronous computing. We illustrate our discussion in terms of the simple, but well established neurodynamics model attributed to Grossberg and Hopfield [8]. Then, we specify the simulation framework. We characterize the chaotic behavior of our discrete event system by estimating the complete Lyapunov spectrum associated with its dynamics, and demonstrate the emergence of complex behaviors. Finally, we briefly address the issue of preventing the emergence of computational chaos.

2. BASIC CONCEPTS

We begin by defining more precisely what we mean by concurrent asynchronous computation. Let N denote the total number of nodes in a network. We assume that a quantity of interest, $x_n(\nu)$, is being estimated at each node n , where ν indexes a discrete temporal sequence. Let φ be the nonlinear operator from \mathbb{R}^N to \mathbb{R}^N representing the model of interest, with network components expressed as $\varphi_n(x_1, x_2, \dots, x_N)$. Also, let $\tau_n(\nu)$ index the availability of the most recently updated state of node n . The successive temporal configurations of the network are in the set $\psi = \{\tau_1(\nu), \dots, \tau_N(\nu) \mid \nu = 1, \dots\}$. Two potential paradigms can be envisioned.

P₁: *random node delays:* at each temporal grid point, only a subset of nodes (e.g., randomly determined) is allowed to update their state. Note that the expression time-grid refers to the numerical solution of the model equations (e.g., coupled ODEs, etc...).

More precisely, a concurrently asynchronous node-delayed system iteration, denoted by the tuple $\{\varphi, \mathbf{x}(0)$,

$\xi, \psi\}$, is a sequence of state iterates $\mathbf{x}(\nu)$ of vectors in \mathbb{R}^N , obtained by the following recursion starting from a given vector of initial node states $\mathbf{x}(0)$:

$$x_n(\nu) = \begin{cases} x_n(\nu-1) & \text{if } n \notin S_\nu \\ \varphi_n(x_1(\tau_1(\nu)), \dots, x_N(\tau_N(\nu))) & \text{if } n \in S_\nu \end{cases} \quad (1)$$

Here S_ν denotes the set of nodes that carry out an update at the ν -th time grid point. The set $\xi = \{S_\nu \mid \nu = 1, 2, \dots\}$ is the sequence of nonempty subsets of nodes that performed an update at each ν .

P₂: *random communication delays:* such delays are assumed to occur between nodes of the network. Each node updates its state at each ν , using the latest available information stored in its local buffer. The updated state is then broadcast to all nodes.

$$x_n(\nu) = \varphi_n(x_1(\tau_1(\nu)), \dots, x_n(\nu-1), \dots, x_N(\tau_N(\nu))) \quad (2)$$

For both paradigms, three operational assumptions are made. Specifically, we require that:

- Each consecutive update uses only state information previously available at the node under consideration, i.e., $\tau_n(\nu) \leq \nu - 1$.
- Conservation of temporal logic: evermore recent state information must be used in evolving each node.
- Node n is not starved in ξ , i.e., there exists a finite natural number $s \in \mathbb{N}$, such that each node updates its estimate at least once in every s successive time grid points.

These two paradigms provide, potentially, a conceptual framework for modeling asynchronous, distributed discrete event systems. The dynamics underlying such systems would be capable of updating the nodes in an uncoordinated manner, where programs at each node are seen as a collection of functionally cooperating processes, with no explicit dependencies to enforce waiting at synchronization points for the purpose of swapping partially computed results.

To date, only paradigm **P₁** has been considered in the literature [9, 10]. The primary contribution of this study is the development of a computational framework corresponding to paradigm **P₂**. Next, we briefly summarize the model we will use to illustrate this study.

3. THE MODEL

We will illustrate our discussion in terms of the temporal evolution of a fully (logically) interconnected Grossberg – Hopfield (GH) network [8], implemented on a spatially distributed computing system. Such a model is represented by the following system of coupled, nonlinear differential equations:

$$\frac{dx_n}{dt} + a_n x_n = \sum_l T_{nl} g(\gamma_l x_l) + I_n. \quad (3)$$

Here x_n represents the internal state of the n th neuron. The strength of the synaptic coupling from neuron l to neuron n is denoted by T_{nl} , and the external bias is denoted by I_n . The sigmoid function g modulates the neural response, γ_n denotes the gain of the transfer function of the n th neuron, and a_n represents the inverse of a characteristic time constant or a decay scaling term. To create a discrete-event model, we replace the time derivative of x_n with a first order finite difference representation. Then the n -th component of the GH discrete-event operator obtained from Eq. (3) is

$$\varphi_n(\mathbf{x}) = x_n + \Delta(-a_n x_n + \sum_{l=1}^{l=N} T_{nl} g(\gamma_l x_l) + I_n). \quad (4)$$

where Δ refers to the time discretization.

4. ASYNCHRONOUS SCHEMES

We begin by briefly summarizing an implementation of paradigm \mathbf{P}_1 . Then, we introduce the new computational framework proposed for paradigm \mathbf{P}_2 .

4.1 Random Node Delays

Let K_n^a denote the maximum number of time-grid points over which updating of node n can be delayed. The actual delay this node experiences is given by

$$K_n^a = r K_n^b + 1 \quad (5)$$

where r is a random number ($0 \leq r < 1$). We see that the following inequality holds:

$$1 \leq K_n^A \leq K_n^B \quad (6)$$

A ‘‘counter’’ denoted K_n^c is associated to each node. It is initialized to zero. Because each K_n^a is defined by using a different random number r , each node n will experience a different delay K_n^a . In other words, at each point of the integration time-grid, a different subset of neurons will update, thus satisfying the paradigm’s assumptions.

The implementation algorithm then proceeds as follows. Each time the dynamics evolves by one time step, the counter array \mathbf{K}^c is incremented by one (recall that initially $K_n^c = 0, \forall n$). Then,

- If $K_n^c < K_n^a$: no update is allowed; $dx_n / dt = 0$, which implies $x_n^{\nu+1} = x_n^\nu$, where ν denotes the time-step (time-grid point).
- If $K_n^c = K_n^a$: node n is allowed to update, i.e., we evaluate the RHS of Eq (1) using Eq (4) and, for all nodes $l \neq n$, their latest available value (i.e., their value at time-step ν when $\nu+1$ is being calculated). Also, when $K_n^c = K_n^a$, after updating x_n , a new actual delay for node n (i.e., K_n^a) is obtained via Eq (5), and K_n^c is reset to zero.

4.2 Random Communication Delays

In the previous paradigm, a node can stay idle up to K_n^b steps. This may result in substantial processor under-utilization in a distributed, multi-processing environment. Moreover, many actual delays may occur not at the nodes, but during information transfer over the network. It is therefore important to account for this phenomenon.

We begin by rewriting Eq (4) in a way that will naturally highlight the network delays. Ignoring the external bias, we have

$$x_n^{\nu+1} = (1 - \Delta a_n) x_n^\nu + \Delta T_{nn} g(\gamma_n x_n^\nu) + \Delta \sum_{l \neq n}^{l=N} T_{nl} g(\gamma_l x_l^{\eta(n,l,\nu,\theta)}) \quad (7)$$

Observe that in the first two terms on the RHS of Eq (7), we use as state variable x_n^ν . We assume that process n evolves on node n , and that its previous values are therefore immediately accessible. In other words, there is no need to transfer these values over a network and no delay is incurred.

On the other hand, for all nodes $l \neq n$, the state variables x_l^η used for updating x_n^ν to $x_n^{\nu+1}$ are evaluated at a previous time-step η , where $\eta = \eta(n, l, \nu, \theta)$. Here θ denotes a random delay experienced by the data packet x_l^η sent at time η on a path from l to n and used by n at ν . Its actual expression will be specified shortly.

To enable the simulation of such a network communication process, we will assume that each node n keeps a virtual buffer matrix for incoming data. Each row of that matrix corresponds to a specific logical channel. For instance, $\langle nl \rangle$ will denote the channel that transfers data from node l to node n . At each time step ν , buffer channel $\langle nl \rangle$ stores (at node n) updates x_l^ν from node l . The buffer, however, is *virtual* because, even though these updates were generated at the same time step ν , they will only be made available to node n at future time steps μ determined from random propagation delays between l and n .

Now, we need to interpret this concept. We see that when Eq (7) is executed (that is, when node n updates its state at time step ν from x_n^ν to $x_n^{\nu+1}$) it fetches a value for x_l from the buffer. This value, denoted x_l^η was actually calculated and sent from node l at a previous time step $\eta = \eta(n, l, \nu, \theta)$. Note that the actual values η and μ may be inferred at the nodes from the time stamps associated with each data packet. When a data packet is sent through a logical channel (which may correspond to multiple hops on a physical network), a random delay occurs. This delay is used, in conjunction with the time stamp, to

position the data in the virtual buffer of the destination node.

Let $\mathbf{B}^{(n)}$ denote the buffer matrix of node n . It is dimensioned as $N \times K$. A row of $\mathbf{B}^{(n)}$ stores data arriving at node n from a particular node l (over channel $\langle nl \rangle$). Thus, the length of such a buffer vector is K , which means that the maximum delay that is allowed to occur on the network is K time steps. Data stored in the buffer will be referred to as *back-values* for a specific channel.

A back-value is therefore simply the value of a state-variable under consideration at some previous time step. For a *synchronous* system, $K = 1$, and a *single-cell buffer* is used for each channel. The K back-values of x_n^ν that will be stored successively in such a buffer at any node l will be: $x_n^\nu, x_n^{\nu+1}, x_n^{\nu+2}, \dots, x_n^{\nu+K-1}$.

In an *asynchronous system*, the situation is quite more complex, since data packets sent from node to node may arrive in time-locally random order. Thus, a more sophisticated data structure is needed for storage. Here, we use the concept of “*circular list*” for storing back-values. In particular, the value x_l^ν sent from node l toward node n at time step ν over channel $\langle nl \rangle$ that exhibits a time-local random delay θ_{nl}^ν will be stored in the buffer matrix of node n at location $B_{lk}^{(n)}$, where k is given by:

$$k = (\mu - 1) \mathbf{I} K + 1 \quad \text{and} \quad \mu = \nu + \theta_{nm}^\nu \quad (8)$$

In Eq (8) the vertical bar denotes integer modular division. That is $(\mu - 1) \mathbf{I} K$ represents the remainder of the integer division of $(\mu - 1)$ by K . Note that the time stamp ν of data packet x_l^ν is stored in a buffer matrix $\Omega_{lk}^{(n)}$. Recall also that $\theta_{nn}^\nu = 0$, i.e., there is no delay in same node buffering.

The process of retrieving data from the virtual buffer is more delicate. In particular, it is essential to insure that *temporal logic* is enforced. This means that data from node l being processed at node n should be retrieved in monotonously increasing temporal order. But, in a communications network subject to random delays, a situation could occur, whereby a data packet $x_l^{\nu-1}$ arrives at node n for iteration $\nu + 1$, after x_l^ν was used at time step ν . This can not be allowed.

Specifically, data fetching from the virtual buffer proceeds as follows. Assume, for illustrative purpose, that we are at node n at time step ν , in the process of generating $x_n^{\nu+1}$. We wish to retrieve the latest available information, x_l^ν , from node l as stored on buffer channel $\langle nl \rangle$. Let ξ denote the time stamp associated with the information from l used in the previous update (at time step $\nu - 1$). These time stamps are retrieved from the buffer $\Omega^{(n)}$ at locations (l, k_η) and (l, k_ξ) , where

$$k_\eta = (\nu - 1) \mathbf{I} K + 1 \quad \text{and} \quad k_\xi = (\nu - 2) \mathbf{I} K + 1 \quad (9)$$

If $\eta > \xi$, we use x_l^η . Else, we must use x_l^ξ and have to reset the buffer

$$B_{lk}^n = x_l^\xi \quad \text{and} \quad \Omega_{lk}^n = \xi, \quad \text{where} \quad k = k_\mu \quad (10)$$

We illustrate this concept below, in Table 1, by showing the time evolution of the content of the back-values stored in channel $\langle nl \rangle$ with $n = 1$ and $l = 2$. We assume that $K = 4$, and that the random delays successively encountered by data packets from l are 3, 1, 2, 2, 3 (in units of Δ). The values to be retrieved at node n at each successive time step ($\nu = 1, 2, \dots$) from buffer locations specified by in Eq (9) would be: $x_2^0, x_2^0, x_2^2, x_2^1, x_2^3, \dots$. As can be observed, this would result in fetching x_2^1 for time step $\nu = 4$, whereas x_2^2 had already been used at $\nu = 3$. Accordingly, we need to invoke k_ξ in Eq.(9), which results in the utilization of x_2^2 for the update at $\nu = 4$. As per Eq (10), the buffer at $\nu = 4$ is reset to $x_2^3, x_2^4, x_2^2, x_2^2$.

$\nu = 0$	x_2^0	x_2^0	x_2^0	x_2^0	initial conditions	
$\nu = 1$	x_2^0	x_2^0	x_2^0	x_2^1	$\mu = 4$	$k = (4-1)4+1=4$
$\nu = 2$	x_2^0	x_2^0	x_2^2	x_2^1	$\mu = 3$	$k = (3-1)4+1=3$
$\nu = 3$	x_2^3	x_2^0	x_2^2	x_2^1	$\mu = 5$	$k = (5-1)4+1=1$
$\nu = 4$	x_2^3	x_2^4	x_2^2	x_2^1	$\mu = 6$	$k = (6-1)4+1=2$
$\nu = 5$	x_2^3	x_2^4	x_2^2	x_2^5	$\mu = 8$	$k = (8-1)4+1=4$

Table 1. Time evolution of the content of the back-values stored in channel $\langle 1, 2 \rangle$

5. LYAPUNOV SPECTRUM

In order to quantitatively characterize the behavior of the network dynamics, consider an equilibrium point \mathbf{x}_e of the autonomous system described by the vector field \mathbf{f} corresponding to the RHS of Eq (3). The local behavior of the flow near \mathbf{x}_e is determined by linearizing the vector field at \mathbf{x}_e , i.e.,

$$\delta \dot{\mathbf{x}} = \mathcal{D}\mathbf{f}(\mathbf{x}_e) \delta \mathbf{x} \quad \delta \mathbf{x}(0) = \delta \mathbf{x}_0 \quad (11)$$

The linear vector field governs the time evolution of a perturbation $\delta \mathbf{x}_0$ in the neighborhood of the equilibrium point. Let the eigenvalues and eigenvectors of $\mathcal{D}\mathbf{f}$ at \mathbf{x}_e be $\lambda_n \in \mathbb{C}$, and $\xi_n \in \mathbb{C}^N$, for $n = 1, 2, \dots, N$. We know from linear systems theory that (assuming that the eigenvalues are distinct) the trajectory with initial conditions $\mathbf{x}_e + \delta \mathbf{x}_0$ evolves as

$$\begin{aligned} \mathbf{x}(t, \mathbf{x}_e + \delta \mathbf{x}_0) &= \mathbf{x}_e + e^{\mathcal{D}\mathbf{f}(\mathbf{x}_e)t} \delta \mathbf{x}_0 \\ &= \mathbf{x}_e + c_1 e^{\lambda_1 t} \xi_1 + \dots + c_N e^{\lambda_N t} \xi_N \end{aligned} \quad (12)$$

where $c_n \in \mathbb{C}$ are constants determined from the initial conditions. If λ_n is real, then ξ_n and c_n are also real. It is clear that λ_n corresponds to the rate of contraction ($\lambda_n < 0$) or expansion ($\lambda_n > 0$) near \mathbf{x}_e in the direction ξ_n .

Since the matrix $\mathfrak{D}\mathbf{f}$ is real, if the eigenvalues λ_n are complex, they occur in complex conjugate pairs, and the real part of λ_n gives the rate of expansion or contraction.

The Lyapunov exponents generalize the concept of eigenvalues at an equilibrium point. Their intended use is to characterize the behavior of a dynamical system which may include equilibrium points, periodic solutions, as well as quasi-periodic and chaotic regimes. To find all N Lyapunov exponents, a set of N linearly independent perturbation vectors $\delta\mathbf{x}^{(m)}$ is repeatedly integrated and orthonormalized [11]. Here, a modified Gram-Schmidt (GS) procedure [12] is used for improved numerical stability. After each integration stage ρ of duration T , the GS generates two sets of vectors, $\mathbf{v}^{(m)}(\rho)$ and $\mathbf{r}^{(m)}(\rho)$, such that $\mathbf{v}^{(1)}(\rho) = \delta\mathbf{x}^{(1)}$, and

$$\begin{aligned} \mathbf{r}^{(m)}(\rho) &= \delta\mathbf{x}^{(m)} - \sum_{i=1}^{m-1} \langle \delta\mathbf{x}^{(m)}, \mathbf{v}^{(i)}(\rho) \rangle \mathbf{v}^{(i)}(\rho) \\ \mathbf{v}^{(m)}(\rho) &= \mathbf{r}^{(m)}(\rho) / \|\mathbf{r}^{(m)}(\rho)\|, \quad m = 2, \dots, N. \end{aligned} \quad (13)$$

Note that the set of vectors $\{\mathbf{v}^{(m)}\}$ spans the same subspace as $\{\delta\mathbf{x}^{(m)}\}$ for $m = 1 \dots N$. Then, at the A -th stage (for A sufficiently large), the n -th Lyapunov exponent is computed as

$$\lambda_n \approx \sum_{\rho=1}^{\rho=A} \text{Log}_e(\|\mathbf{r}^{(n)}(\rho)\|) / AT. \quad (14)$$

For an asynchronous DES with *random network delays*, the variational equation corresponding to Eq (3) takes the form

$$\begin{aligned} \frac{d}{dt} U_{nm} &= -a_n U_{nm} + T_{nn} (1 - g^2(\gamma_n x_n)) \gamma_n U_{nm} \\ &+ \sum_{\substack{l=N \\ l \neq n}}^{l=N} T_{nl} (1 - g^2(\gamma_l x_l^{\eta(n,m,v,\theta)})) \gamma_l U_{lm}^{\eta(n,m,v,\theta)} \end{aligned} \quad (15)$$

where

$$U_{nm}(t) = \delta x_n(t) / \delta x_m(0), \quad \mathbf{U}(0) = \mathbf{I}. \quad (16)$$

Note that the vector $\delta\mathbf{x}^{(m)}$ in (13) refers to the m -th column of \mathbf{U} . In deriving Eq (15), a distribution of one process (here neuron) per node was assumed, for exposition simplicity.

We now apply these concepts to illustrate the emergence of computational chaos in a distributed discrete event system.

6. EMERGENCE OF COMPUTATIONAL CHAOS

We now consider a low-dimensional model made up of 4 spatially distributed, but logically fully interconnected neurons. We show that, even in such a small network, asynchronous dynamics gives rise to a variety of complex behaviors.

In a previous study [13], we examined emergent behaviors under random *node* delays. Here, the focus is on random *network* delays. All simulations were performed with the recently developed *A♦NET* code

[14]. This code is entirely written in *Intel* Visual FORTRAN 95. The figures correspond to a direct screen dump at the conclusion of a simulation. The *A♦NET* visualization software creates, in real time, these graphics displays.

Each figure comprises three regions. The upper region displays the evolution of the complete Lyapunov Spectrum. The color-coded magnitude of each exponent is plotted versus consecutive time intervals.

The lower left region displays the signal output of each neuron versus integration time. This time-series plot is a basic observational tool for dynamical systems. As integration time progresses, one is able to monitor the output of each neuron, to ascertain whether it converges to a single (fixed) point, follows a cyclic path, or wanders chaotically.

The lower right region displays a Poincaré plot. This is a phase-space diagram where the signal output of one neuron is plotted versus the signal output of another neuron over time. From a dynamical system perspective, the Poincaré plot depicts the trajectories (orbits) of two particular components (neurons) of the network. Trajectories that enter the domain of a *point attractor* will approach and remain at that point. Then a stable equilibrium solution or fixed point has been reached. Trajectories that enter the domain of a *limit cycle attractor* will approach and generate a periodic solution. Finally, trajectories that enter the domain of a *strange attractor*, will exhibit divergence from one another, and are usually in a state of chaos.

Our first case addresses a situation, whereby in the synchronous regime the dynamics converges to a fixed point attractor. In the asynchronous regime, as delays are introduced, we observe first a transition to a quasi limit cycle. Then, as delays become larger, a chaotic regime emerges. The following parameters are used: $\forall n, a_n = \gamma_n = 1, I_n = 0$. The elements of the synaptic matrix are shown in Table 2.

0.850	-2.000	1.100	0.500
1.800	1.150	0.600	0.300
1.100	2.500	2.500	0.050
0.100	-0.400	-1.441	1.450

Table 2

Synchronous dynamics is an idealized situation in which no network delays are assumed to occur in information propagation between processes. This is a convenient simulation assumption, which is essentially equivalent to the requirement of synchronization blocking (and possible processor idling) in an actual distributed system. Of course, in a real-life system, synchronization requirements reduce the overall efficiency of the information processing throughput.

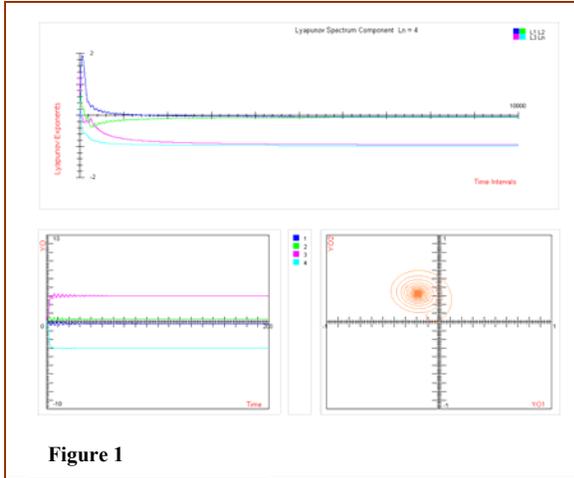


Figure 1

As can be observed in Fig. 1, the network dynamics converges to a *fixed point* attractor. All Lyapunov exponents are accordingly negative. The actual spectrum calculated by the *A♦NET* code is:

$$(\lambda_1 = -0.047, \lambda_2 = -0.059, \lambda_3 = -0.948, \lambda_4 = -0.975).$$

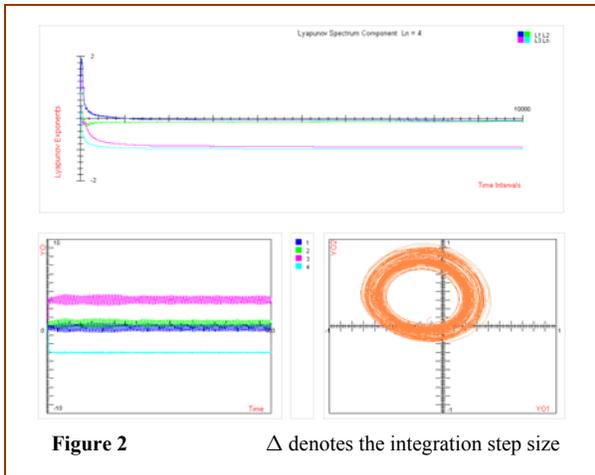


Figure 2

Δ denotes the integration step size

For random communication delays of up to 5Δ , the Poincaré plot in Fig. 2 appears to indicate a quasi limit cycle. This is supported by the output signal from each neuron, which exhibits almost periodic motion.

When the bound on random communication delays increases, aperiodic oscillations arise. For a maximum allowable communication delay of 200Δ , the emergence of computational chaos (Fig. 3) is confirmed by the existence of positive components in the Lyapunov spectrum. The exponents calculated by the *A♦NET* code are:

$$(\lambda_1 = +0.162, \lambda_2 = +0.026, \lambda_3 = -0.303, \lambda_4 = -0.608).$$

Our second case addresses a situation, whereby in the synchronous regime the dynamics converges to a *limit cycle* (see Fig. 4). In the asynchronous regime, as delays are introduced, we observe (Fig. 5) transition to chaos. The limit cycle was obtained by changing three synaptic parameters. The following data were changed:

$$T_{1,2} = -6.00; T_{1,3} = -0.55; T_{1,4} = +2.08.$$

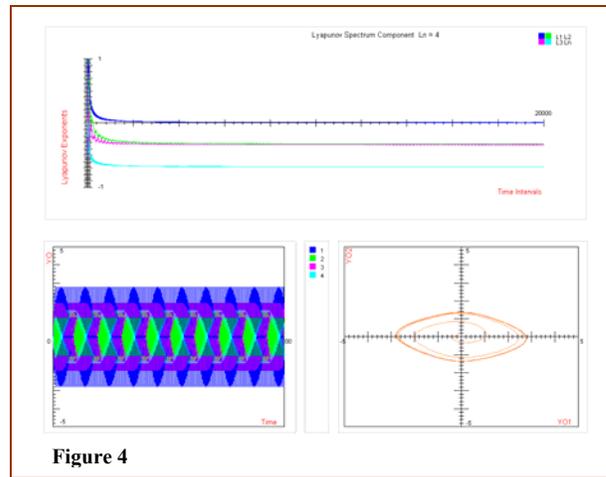


Figure 4

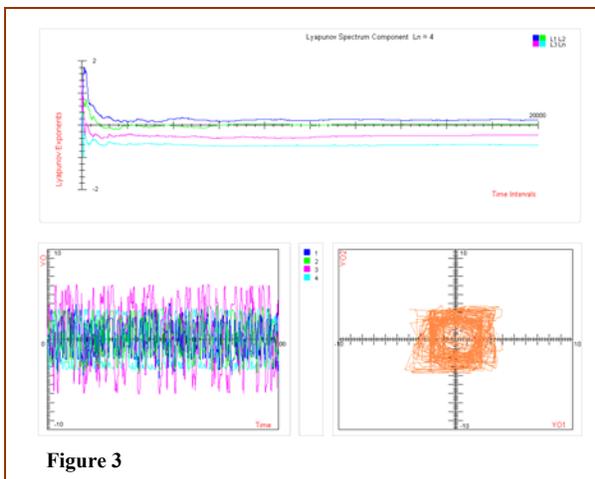


Figure 3

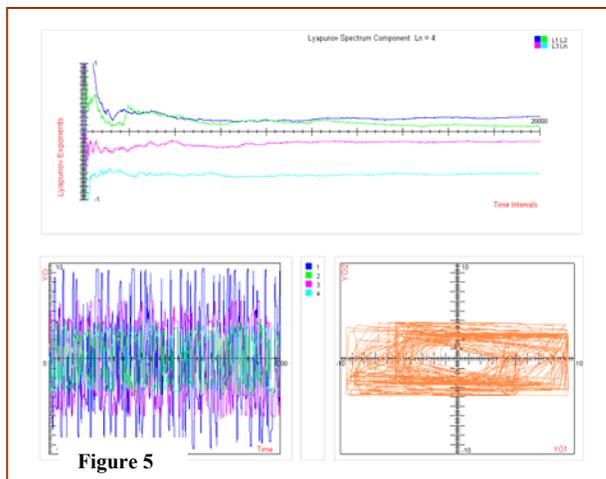


Figure 5

The lower left region in Fig 4 clearly indicates periodic behavior. The leading Lyapunov exponent is zero. In presence of network communication delays, Fig 5 illustrates the emergence of computational chaos. The Lyapunov spectrum exhibits a distribution with two positive exponents:

$$(\lambda_1 = +0.216, \lambda_2 = +0.008, \lambda_3 = -0.139, \lambda_4 = -0.607).$$

7. TAMING COMPUTATIONAL CHAOS

Since asynchronous discrete event systems may become chaotic, additional tools are needed to guarantee that correct results are ultimately obtained. The tools we are proposing are based on the concept of contraction [15]. Contraction plays a fundamental role in the iterative solution of nonlinear equations. It is most useful to express contraction in terms of vector norms, defined as $\|\mathbf{x}\| = (\|x_1\|, \dots, \|x_N\|)$ [15]. This norm induces a partial ordering on \mathbb{R}^N .

An operator $\varphi: \mathcal{D} \subset \mathbb{R}^N \rightarrow \mathbb{R}^N$ is called a Φ -contraction on a set $\mathcal{D}_0 \subset \mathcal{D}$, if there exists a linear operator $\Phi \in L(\mathbb{R}^N)$ with the following properties:

- $\|\varphi(\mathbf{x}) - \varphi(\mathbf{y})\| \leq \Phi \|\mathbf{x} - \mathbf{y}\| \quad \forall \mathbf{x}, \mathbf{y} \in \mathcal{D} \quad (17-a)$
- $\Phi \geq 0 \quad (17-b)$
- $\rho(\Phi) < 1. \quad (17-c)$

The first property implies Lipschitz continuity. Indeed, Φ is often referred to as the Lipschitz *matrix* of φ . The latter requirements, namely non-negativity and spectral radius of Φ , generalize the typical specification of the contractive constant used in conjunction with the usual norm on \mathbb{R}^N . We make use of the following result [9].

Baudet's Theorem. If $\varphi: \mathbb{R}^N \rightarrow \mathbb{R}^N$ is a Φ -contraction on the closed subset $\mathcal{D} \subset \mathbb{R}^N$, and if $\varphi(\mathcal{D}) \subset \mathcal{D}$, then any asynchronous iteration corresponding to φ and starting with a vector $\mathbf{x}(0) \in \mathcal{D}$, converges to a unique fixed point of φ on \mathcal{D} .

These concepts can be applied to study the convergence of concurrently asynchronous time-evolving processes in general, and discrete event systems in particular. In a recent effort [13], we derived specific conditions for taming computational chaos for distributed asynchronous systems under *node* delays. A preliminary analysis indicates that similar derivations can be applied to networks with *communication* delays, even though different variational equations are needed for estimating the Lyapunov spectrum. These results will be reported at an upcoming conference.

8. CONCLUSIONS

In this paper, we have shown that a discrete event model associated with a spatially distributed, concurrently asynchronous system may exhibit complex dynamical

behaviors. Controlling the dynamics of materials at the molecular level is a typical example of a hard problem requiring such a formalism. Here, we have used for illustrative purposes a much simpler, but phenomenologically comparable model, namely a discretized version of the well known Grossberg-Hopfield neural network. A computational framework based on network communication delays was proposed, including node buffer architecture details enabling the conceptual modeling of concurrently asynchronous processes. The emergence of computational chaos from fixed point and limit cycle attractors was observed and accurately characterized. To that effect, the complete Lyapunov spectrum associated with the network dynamics was computed. Future work directed toward the taming of computational chaos was also briefly addressed.

ACKNOWLEDGEMENTS. Funding for this effort was provided by the DOE Office of Basic Energy Sciences (JB and VP), and by the Missile Defense Agency (JB). ORNL is operated for DOE under contract DE-AC05-00OR22725 with UT - Battelle, LLC.

REFERENCES

1. Braiman, Y., J. Barhen, and V. Protopopescu, "Control of friction at the nanoscale", *Phys. Rev. Lett.*, **90**(9), 094301_1-4 (2003).
2. V. Garg, *Elements of Distributed Computing*, Wiley (2002)
3. Lamson, B., M. Paul, and H. Siegert, *Distributed Systems*, Springer (1981).
4. Rodrigue, G., *Parallel Computations*, Academic Press (1982).
5. Giambiasi, N., B. Escude, and S. Ghosh, "GDEVS: A generalized discrete event specification for accurate modeling of dynamic systems", *Trans Soc Comp Simul*, **17**(3), 120-124 (2000).
6. Paillet, J.L. and N. Giambiasi, "DECM, a user oriented formalism for high level discrete event specifications of real-time systems", *Jour Intel & Robotic Sys*, **34**(1), 27-81 (2002).
7. Chazan, D. and W. Miranker, (1969). "Chaotic relaxations", *Linear Algebra & Applic.*, **2**, 199-222.
8. Hopfield, J., (1984). "Neurons with graded response have collective computational properties like those of two-state neurons", *Proc. Nat. Acad. Sci.*, **91**, 3088-3092.
9. Baudet, G. M., (1978). "Asynchronous iterative methods for multiprocessors", *Jour. ACM*, **25**, 226-244.
10. Bertsekas, D. and J. Tsitsiklis, *Parallel and Distributed Computation*, Athena Scientific (1997).
11. Wolf, A., J. Swift, H. Swinney, and J. Vastano, (1985). "Determining Lyapunov exponents from a time series", *Physica* **16D**, 285-317.
12. Golub, G. and C. Van Loan, (1996). *Matrix Computations*, Johns Hopkins University Press.
13. Barhen, J. V. Protopopescu, S. Barhen, and J. Wells, "Asynchronous Computation and Emergence of Computational Chaos", *Proceedings of ADHS'03s*, pp. 123-128, IFAC Press (2003).
14. Barhen, S., "Asynchronous computing in artificial neural networks", ORNL TM report (in press, 2004).
15. J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press (1970).