Flexible parallel implementation of logic gates using chaotic elements

Sudeshna Sinha,[1] Toshinori Munakata,[2] and William L. Ditto[3]
[1]*The Institute of Mathematical Sciences, CIT Campus, Chennai 600 113, India*
[2]*Computer and Information Science Department, Cleveland State University, Cleveland, Ohio 44115*
[3]*Georgia Tech/Emory Biomedical Engineering Department, 315 Ferst Drive, Atlanta, Georgia 30332-0535*
(Received 21 June 2001; revised manuscript received 19 October 2001; published 21 February 2002)

We demonstrate the basic principles for the direct and flexible implementation of all basic logical operations utilizing low dimensional chaos. Then we generalize the concept to high dimensional chaotic systems, and show the parallelism inherent in such systems. As a case study we implement the proposed parallel computing architecture to obtain parallelized bit-by-bit addition with a two-dimensional chaotic neuronal and a three-dimensional chaotic laser model.

DOI: 10.1103/PhysRevE.65.036216 PACS number(s): 05.45.−a, 89.70.+c

I. INTRODUCTION

A recurring theme of research into chaotic systems over the last decade has been that chaos provides ''flexibility'' in the performance of natural systems and provides such systems with a rich variety of behaviors that can be utilized for more versatile performance. In particular, attempts to bridge dynamics and computations [1–7] present a new direction in harnessing chaos. For instance, the capability of dynamical systems to perform the fundamental NOR logic has been successfully demonstrated recently [7], and this indicates the scope of building general-purpose machines from chaotic processors.

The motivation for exploring chaos as a candidate for direct and controlled computing (as in endeavors such as DNA [8] and quantum computing [9]) is to find new ways to exploit physical phenomena that are well understood in the context of physics to do computations, i.e., to use new concepts of physics to build better computing devices. Our general strategy here will be to exploit the determinism of dynamics on one hand, and its richness on the other. The determinism will allow us to ''reverse engineer,'' so to speak, and the richness of dynamical patterns will allow flexibility and versatility in accomplishing all the fundamental operations.

The basic components of computer architecture today are the logical AND, OR, NOT, and XOR (exclusive OR) operations, from which we can directly obtain basic operations like bit-by-bit addition and memory [10]. These operations act on two inputs I_1 and I_2 (for AND, OR, and XOR) or one input I (in case of NOT) and outputs a signal O. The logical operations are defined by patterns of input-to-output mapping represented by the truth table in Table I. Now all the above-mentioned gates can be constructed by combining the NOR operation proposed in [7]. For example, AND can be realized by $\mathrm{AND}(X,Y)=\mathrm{NOR}(\mathrm{NOR}(X,Y),\ \mathrm{NOR}(X,Y))$ and $\mathrm{XOR}(X,Y)=\mathrm{NOR}(\mathrm{NOR}(\mathrm{NOR}(X,\mathrm{NOR}(X,Y))),\ \mathrm{NOR}(\mathrm{NOR}(\mathrm{NOR}(X,Y),Y)))$. Clearly though, this conversion process is inefficient in comparison with direct implementation, considering perhaps such fundamental operations may be performed a large number of times. So the direct and flexible implementations of gates is useful and could prove very cost effective. Our problem then is to design chaotic elements that yield the appropriate outputs for the different fundamental gates for all possible sets of inputs.

TABLE I. The truth table of the basic logic operations. Column 1 shows $\mathrm{AND}(I_1,I_2)$. column 2 shows $\mathrm{OR}(I_1,I_2)$, and column 3 shows $\mathrm{XOR}(I_1,I_2)$, where the two inputs are I_1 and I_2. Column 4 shows the NOT gate, where there is one input, I.

I_1	I_2	AND	OR	XOR	I	NOT
0	0	0	0	0	0	1
0	1	0	1	1	1	0
1	0	0	1	1		
1	1	1	1	0		

Towards this aim, in this work we first show the *direct and flexible implementation* of *all* these logical operations utilizing low dimensional chaos. Then we give details of the general concept and specific implementation of parallelized logic using high dimensional chaotic elements.

The organization of this paper is as follows: In Sec. II we demonstrate the basic principles of obtaining different logic operations and implement these on one-dimensional chaotic maps. Section III outlines the generalization to multidimensional chaotic systems and the parallelism inherent in such systems. Section IV gives a specific implementation using a two-dimensional chaotic neuronal map and Sec. V utilizes a three-dimensional chaotic laser model. Finally Sec. VI discusses and summarizes the results.

II. BASIC LOGIC OPERATIONS WITH A CHAOTIC MAP

Here we will endeavor to obtain clearly defined AND, OR, NOT, and XOR gate response patterns with a single chaotic element. These are the necessary and sufficient ingredients of the most basic components to build a computer. Our technique is simple and direct, and is targeted at achieving universal general purpose computing, rather than a narrowly specialized problem domain.

Consider a single chaotic element whose state is represented by a value x, as our *chaotic chip* or *chaotic processor*. The state of the element evolves according to some dynamical rule exhibiting chaos. For instance, the updates of the state of the element from time n to $n+1$ may be well de-

 ©2002 The American Physical Society

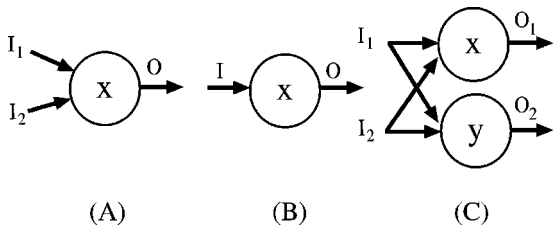

FIG. 1. Three types of input/output configurations: (a) logical AND, OR, and XOR; (b) logical NOT; (c) bit-by-bit arithmetic addition.

scribed by a map, i.e., $x_{n+1}=f(x_n)$, where f is a *nonlinear function* chosen to obtain chaotic dynamics.

Now this element receives two inputs (for AND, OR, and XOR) or one input (in case of NOT) and outputs a signal. In our scheme, the inputs to the chaotic elements are denoted by $x_{\rm in}^1$, $x_{\rm in}^2$,... and the action of the inputs is to stimulate (kick) the state of the system as follows:

$$x \rightarrow x + x_{\rm in}^1 + x_{\rm in}^2 + \cdots ,$$

where $x_{\rm in}^i = 0$ if $I_i = 0$ and $x_{\rm in}^i = \delta > 0$ if $I_i = 1$.

The *outputs* from the chaotic element are obtained by a simple *threshold mechanism*. After suitable time if the evolved state variable $f(x)$ is larger than a prescribed threshold x^*, i.e., $f(x) > x^*$, the state variable is reset to threshold value x^* emitting the excess amount $x_{\rm out} = \{f(x) - x^*\}$. If the value of the variable is under the critical value x^*, i.e., $f(x) \leqslant x^*$, there is no response from the element and $x_{\rm out} = 0$.

This emitted excess $x_{\rm out}$ encodes the output of the operation: when $x_{\rm out} = 0$ it encodes 0 and when $x_{\rm out} \sim \delta$ it encodes 1, where δ is a prescribed positive constant [11]. So if the variable x of the evolved state of the chaotic element is lower than threshold it yields 0, i.e., $O \equiv 0$. If the variable x has a value greater than threshold, with the excess amount being $\sim \delta$, the output has value 1, i.e., $O \equiv 1$ [12].

Setting the threshold value and initial state of the system so that it directly gives the desired response (i.e., outputs the desired excess $x_{\rm out}$) constitutes *programming* the gates.

In our implementation we demand that the *input and output have equivalent definitions* (i.e., one unit is the same quantity for input and output), as well as among various logical operations. This requires that constant δ assumes the same value throughout a network, and this will allow the output of one gate element to easily couple to another gate element as input, so that gates can be ''wired'' directly into gate arrays implementing compounded logic operations.

The number of inputs and outputs for each chaotic element depends on the specific operation. Figure 1 depicts the three types of input/output configurations for a chaotic element. The circles in the figures represent chaotic elements. For example, Fig. 1(a) shows a configuration with two inputs I_1 and I_2, and one output O. This configuration is used for the Boolean AND, OR, and XOR operations (see Table I). Figure 1(b) has one input I and one output O for the NOT operation. Figure 1(c) has two inputs I_1, I_2 and two outputs O_1, O_2, and it is the configuration necessary for bit-by-bit arithmetic addition (see Table V in Sec. IV for truth table).

TABLE II. The correspondence between actual input and output, designated as $x_{\rm in}$ and $x_{\rm out}$, respectively, in the text and their interpreted values. Here output $O=0$ represents false in a Boolean operation and 0 in bit-by-bit addition, and $O=1$ represents true in a Boolean operation, and 1 in bit-by-bit addition.

Interpreted I/O	Actual I/O
0	0
1	δ

All the fundamental logic gate operations involve the following steps,

(1) Initialization and external inputs,

$x \rightarrow x_0 + x_{\rm in}^1 + x_{\rm in}^2$ for the AND, OR, and XOR operations, and

$x \rightarrow x_0 + x_{\rm in}$ for the NOT operation,

where x_0 is the initial state of the system, and $x_{\rm in} = 0$ when $I = 0$ and $x_{\rm in} = \delta$ when $I = 1$.

(2) Chaotic update, i.e., $x \rightarrow f(x)$, where $f(x)$ is a chaotic function.

(3) Threshold mechanism to obtain output $x_{\rm out}$ is

$$x_{\rm out} = 0 \qquad \text{if } f(x) \leqslant x^*, \text{ and}$$

$$x_{\rm out} = \{f(x) - x^*\} \qquad \text{if } f(x) > x^*,$$

where x^* is the threshold. As shown in Table II this is interpreted as $O=0$ if $x_{\rm out} = 0$ and $O=1$ if $x_{\rm out} \sim \delta$.

In order to obtain the desired input-output response we need to satisfy the conditions enumerated in Table III for the different gates. Note that the symmetry of inputs reduces the four conditions in the truth table of Table I to three distinct conditions, with rows 2 and 3 of Table I leading to condition 2 in Table III.

For instance, for the AND gate implementation the three conditions arise as follows: Condition 1 comes from row 1 of Table I which has $I_1 = I_2 = 0$. This implies $x_{\rm in}^1 = x_{\rm in}^2 = 0$, and so the initial state after inputs remains at x_0. After chaotic evolution the state is $f(x_0)$. Since the output of an AND gate for inputs (0,0) is 0, $x_{\rm out}$ should be 0, i.e., $f(x_0) \leqslant x^*$.

Condition 2 comes from rows 2 and 3 of Table I, which has either I_1 or I_2 to be one. This implies either $x_{\rm in}^1 = \delta$ and $x_{\rm in}^2 = 0$ or $x_{\rm in}^1 = 0$ and $x_{\rm in}^2 = \delta$. So the initial state after inputs now is $x_0 + \delta$, and the state after chaotic evolution is $f(x_0 + \delta)$. Since the output of an AND gate for inputs (1,0) and (0,1) is 0, $x_{\rm out}$ should again be 0, i.e., $f(x_0) \leqslant x^*$.

Condition 3 comes from row 4 of Table I, which has $I_1 = I_2 = 1$. This implies $x_{\rm in}^1 = x_{\rm in}^2 = \delta$ and so, the initial state after inputs is $x_0 + 2\delta$. After chaotic evolution one then has state $f(x_0 + 2\delta)$. Since the output of an AND gate for inputs (1,1) is 1, $x_{\rm out}$ should now be δ, i.e., $[f(x_0 + 2\delta) - x^*] \sim \delta$.

All the three conditions have to be satisfied *simultaneously* to implement the AND gate, as the mapping from $(x_{\rm in}^1, x_{\rm in}^2)$ to $x_{\rm out}$ must hold for all combinations of $(x_{\rm in}^1, x_{\rm in}^2)$. Conversely, when all three conditions are satisfied, $x_{\rm out} \equiv {\rm AND}(I_1, I_2)$ holds. That is, these conditions are neces-

TABLE III. Necessary and sufficient conditions to be satisfied by a chaotic element in order to implement the logical operations AND, OR, XOR, and NOT.

Operation	AND	OR	XOR	NOT
Condition 1	$f(x_0)\leqslant x^*$	$f(x_0)\leqslant x^*$	$f(x_0)\leqslant x^*$	$f(x_0)-x^*\sim\delta$
Condition 2	$f(x_0+\delta)\leqslant x^*$	$f(x_0+\delta)-x^*\sim\delta$	$f(x_0+\delta)-x^*\sim\delta$	$f(x_0+\delta)\leqslant x^*$
Condition 3	$f(x_0+2\delta)-x^*\sim\delta$	$f(x_0+2\delta)-x^*\sim\delta$	$f(x_0+2\delta)\leqslant x^*$	

sary and sufficient for implementing AND. Similarly, one can obtain conditions for OR, XOR, and NOT.

So given a dynamics $f(x)$ corresponding to the physical device in actual implementation, one must find values of threshold and initial state satisfying the conditions derived from the truth table to be implemented.

Here is a numerical example of the basic procedure laid out above. As a representative chaotic function, we take $f(x)$ to be the prototypical logistic map: a map known to be of widespread relevance to physical and biological chaotic phenomena,

$$f(x)=4x(1-x),$$

where $x\in[0,1]$. Select the constant δ, common to both input and output and to all logical gates to be $\frac{1}{4}$. The following Table IV shows the initial x_0 and threshold x^*, which satisfy the conditions in Table III. For instance, for AND, selecting $x_0=0$ and $x^*=3/4$ satisfies the three conditions in Table III as follows:

$$f(x_0)=f(0)=0\leqslant x^* \quad (=3/4),$$

$$f(x_0+\delta)=f(1/4)=0\leqslant x^* \quad (=3/4),$$

$$f(x_0+2\delta)-x^*=f(1/2)-3/4=1-3/4=1/4=\delta.$$

Further, bit-by-bit arithmetic addition, the most fundamental form of arithmetic operation, is constructed from XOR and AND gates. Other types of arithmetic operations can then easily be performed using this basic addition or a similar operation. For example, addition of larger numbers (e.g., addition of two 32-bit numbers) can be carried out by extending the bit-by-bit operation to a higher number of bits. Subtraction can be done as addition of the complemented numbers. Multiplication can be achieved as repeated addition or its variations; similarly, division can be done as repeated subtraction. Further, using logical operations such as AND, computer memory based on integrated circuits can be constructed. Such memory architecture is based on flip flops, which in turn are built by combining logical gates [10].

TABLE IV. Numeric example of implementation of the logical operations AND, OR, XOR, and NOT, with $\delta=\frac{1}{4}$.

Operation	AND	OR	XOR	NOT
x_0	0	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{2}$
x^*	$\frac{3}{4}$	$\frac{11}{16}$	$\frac{3}{4}$	$\frac{3}{4}$

In summary, we have given a direct implementation of all the basic logic gates using a *single* chaotic element, by merely changing the threshold and the initial state [13]. This feature can be exploited to obtain programmable hardware, i.e., to obtain different gate arrays, on demand, from the same set of chaotic processors, exploiting the fact that the chaotic element can act as different gates by simply changing the threshold parameter and initial state, which is programmed information (or the ''software'').

Note that *nonlinearity* in the processing units is clearly necessary for various Boolean implementations. The complex relationship between input(s) and output(s) eliminates the possibility of any linear function mimicking all of the Boolean operations. Only sufficient richness of dynamical behavior can ensure the capacity to get *all* the different applications from the *same* processing units. Here we could readily ''control'' the chaotic map to yield the dynamical response necessary for the various desired applications.

Now we will demonstrate how high dimensional chaotic systems can be exploited for parallel computing, a key information processing technology for increasing the effective speed of computing. We show that a single chaotic dynamical element of dimension N can effectively serve as a parallel processor of N inputs. We exploit the dimensionality of these systems and the richness of temporal patterns inherent in their dynamics to obtain parallelized operations. The space costs, i.e., the number of processors necessary, need not then scale up with the number of operations in such systems, thus effecting reduction of computational effort.

III. GENERALIZATION TO MULTIDIMENSIONAL SYSTEMS

Most complex physical systems have many degrees of freedom and are characterized by many dimensions, with the system's physical size very often not scaling with this dimensionality of the dynamical system, i.e., typically, the *dimensionality* of such systems can be very *large* even if the *spatial extent of the device is very small*. The evolving state of a system at any point of time is characterized by many components, each described by a dynamical variable. For instance, certain neuronal cells can be realistically described by 120 variables [14,15]. Our approach now will exploit the variety of multidimensional dynamical states available in a single chaotic element to implement parallel computing (see Fig. 2).

The objective here is to obtain N clearly defined logic gate response patterns from the N components characterizing the state of an N-dimensional system. This enables us to implement N operations in parallel with a *single* N-dimensional

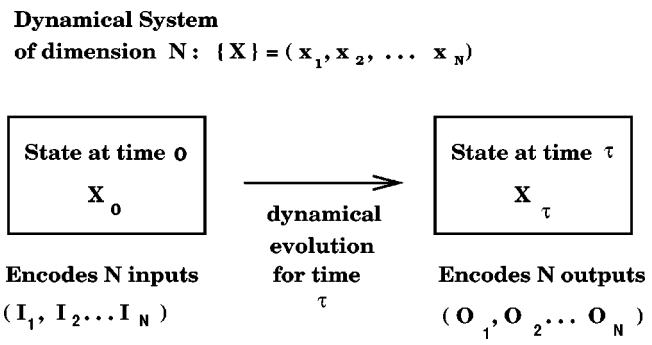

FIG. 2. Schematic figure of parallelism achievable through high dimensional chaotic systems. Note that high dimensionality reflects the *complexity* of the dynamics rather than actual physical size.

chaotic element. Thus one can gain processing power without having to add more elements [16]. Below we generalize the approach introduced in Sec. II to multidimensions.

Specifically our processor now is an N-dimensional chaotic element, whose state is characterized by $\mathbf{x} \equiv (x_1, x_2, \ldots, x_N)$. As before, the M *inputs* $I_1^i, I_2^i, \ldots, I_M^i$ to the ith variable of the chaotic element are encoded through values $x_{\rm in}^1(i), x_{\rm in}^2(i), \ldots, x_{\rm in}^M(i)$. The action of the inputs is to stimulate (kick) the state of the system as follows:

$$x_i \rightarrow x_i + x_{\rm in}^1(i) + x_{\rm in}^2(i) + \cdots + x_{\rm in}^M(i)$$

with $i = 1, \ldots, N$ for the N state variables. As before, $x_{\rm in}^k(i) = 0$ when $I_k^i = 0$ and $x_{\rm in}^k(i) = \delta$, when $I_k^i = 1$.

The *outputs* from the chaotic element are again obtained by the simple threshold mechanism. After time τ, if the evolved state variable $x_i(\tau)$ is larger than a prescribed threshold x_i^*, i.e., $x_i(\tau) > x_i^*$, the state variable is reset to threshold value x_i^* emitting the excess amount $x_{\rm out}^i = \{x_i(\tau) - x_i^*\}$. If the value of the variable is under the critical value x_i^*, i.e., $x_i(\tau) \leqslant x_i^*$, there is no response from the element and $x_{\rm out}^i = 0$. As before, $x_{\rm out}^i = 0$ encodes 0 and $x_{\rm out}^i \sim \delta_i$ encodes 1.

Setting the evolution time τ, threshold values, and initial state of the system so that it directly gives the desired response (i.e., outputs the desired excesses $x_{\rm out}^1, x_{\rm out}^2, \ldots, x_{\rm out}^N$) again constitutes programming the gates. We also demand that the input and output have equivalent definitions and interpretations i.e., one unit is the same quantity for input and output. This will allow the output of one gate element to easily feed into another gate element as input, so that elements can be coupled into gate arrays implementing compounded logic operations. We now describe two specific implementations of this parallel computation architecture below.

IV. IMPLEMENTATION OF PARALLEL COMPUTING BY A TWO-DIMENSIONAL CHAOTIC NEURON MAP

We obtain clearly defined logic gate response patterns, of two logic gates in parallel, with a single two-dimensional (2D) chaotic element. Specifically we demonstrate the representative example of *bit-by-bit arithmetic addition*. This involves two logic operations, AND and XOR, for which there are two inputs, i.e., $M = 2$ in the general scenario given above. We perform these operations in parallel with the two-dimensional chaotic neuron.

TABLE V. The truth table of bit-by-bit arithmetic addition.

I_1	I_2	O_1	O_2
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

The two inputs to the 2D chaotic processor are denoted by $(x_{\rm in}^1(1), x_{\rm in}^2(1))$ and $(x_{\rm in}^1(2), x_{\rm in}^2(2))$ for the two variables x_1 and x_2, respectively. Setting the threshold values and initial state of the system so that it directly gives the desired response (i.e., outputs the desired excesses $x_{\rm out}^1$ and $x_{\rm out}^2$) constitutes designing the gates.

Bit-by-bit addition operates on two-inputs I_1, I_2 and yields two outputs O_1, O_2, with $O_1 \equiv {\rm XOR}(I_1, I_2)$ being the first (rightmost) digit of the sum and $O_2 \equiv {\rm AND}(I_1, I_2)$ being the carry of the answer to the next digit (see Table V). So the inputs to the two variables $\{x_1, x_2\}$ are the same, namely, $I_1^1 = I_1^2 = I_1$ and $I_2^1 = I_2^2 = I_2$ here.

The actual input, designated as $x_{\rm in}^k(i)$ (with $k = 1,2$ and $i = 1,2$) and the actual output, designated as $x_{\rm out}^i$ ($i = 1,2$) will be interpreted as follows: $x_{\rm in}^k(i) = 0$ when $I_k^i = 0$ and $x_{\rm in}^k(i) = \delta_1$ when $I_k^i = 1$. Output O_i is 0 when $x_{\rm out}^i = 0$, and O_i is 1 when $x_{\rm out}^i \sim \delta_i$. In our representative example, we will look for solutions adhering to the stringent conditions $\delta_1 = \delta_2 = \delta$ (but this need not be the case, in general).

The two-dimensional model for biological neurons we focus on is given by [17]

$$x_1(n) = \{x_1(n-1)\}^2 \exp\{x_2(n-1) - x_1(n-1)\} + k,$$

$$x_2(n) = a x_2(n-1) - b x_1(n-1) + c. \qquad (1)$$

This map displays markedly neuronlike dynamics and was found to agree qualitatively with experiments [18]. Here n is the discrete time index, variable x_1 is related to an instantaneous membrane potential of the neuron and the variable x_2 is equivalent to a recovery current. The model has four parameters: a determines the time constant of reactivation, b the activation dependence of the recovery process, c the maximum amplitude of the recovery current, and parameter k can be viewed either as a constant bias or as a time-dependent external stimulation. The parameters here are chosen so as to keep the dynamics completely chaotic ($a = 0.89$, $b = 0.18$, $c = 0.28$, $k = 0.03$). Now we will employ three steps to implement our logical operations on the above system.

Step 1. Initialization of the state of the system to $\{x_{10}, x_{20}\}$ and addition of external inputs,

$$x_1 \rightarrow x_{10} + x_{\rm in}^1(1) + x_{\rm in}^2(1),$$

and likewise,

TABLE VI. Necessary and sufficient conditions to be satisfied by a 2D chaotic element in order to implement the logical operations AND and XOR in parallel. Here variable x_1 is implementing the XOR operation and variable x_2 is implementing the AND operation.

Initial state	XOR	AND
x_{10}, x_{20}	$x_1(n) \leqslant x_1^*$	$x_2(n) \leqslant x_2^*$
$x_{10}+\delta_1, x_{20}+\delta_2$	$x_1(n)-x_1^* \sim \delta_1$	$x_2(n) \leqslant x_2^*$
$x_{10}+2\delta_1, x_{20}+2\delta_2$	$x_1(n) \leqslant x_1^*$	$x_2(n)-x_2^* \sim \delta_2$

$$x_2 \rightarrow x_{20} + x_{\text{in}}^1(2) + x_{\text{in}}^2(2).$$

Step 2. Chaotic evolution for n time steps, from the initial state given above, via Eq. (1).

Step 3. Threshold mechanism on the two variables to obtain the two outputs, namely, the outputs x_{out}^1, x_{out}^2 (encoding O_1 and O_2 in Table V) are given by $x_{\text{out}}^i = 0$ if $x_i(n) \leqslant x_i^*$; $x_{\text{out}}^i = \{x_i(n) - x_i^*\}$ if $x_i(n) > x_i^*$. If $x_{\text{out}}^i = 0$ it encodes 0 and if $x_{\text{out}}^i \sim \delta_i$ it encodes 1.

Having laid out our three-step procedure, the next step is to design our system in such a way that it yields the desired input-to-output mapping defined by Table V, i.e., given a multidimensional chaotic evolution function, we want to have the free parameters to be consistent with the above procedure and also achieve the required mapping.

So for the bit-by-bit addition example, x_{out}^1 should yield XOR(I_1, I_2) and x_{out}^2 should yield AND(I_1, I_2). There are four rows in Table V corresponding to the four possible combinations of I_1, I_2. As the I/O relations are symmetric with respect to I_1 and I_2 we can combine rows 2 and 3 in Table V into a single case. So we need to consider the following.

Case 1. Both I_1, I_2 are 0 (row 1 in Table V) i.e., the initial states of the variables x_1 and x_2 are $x_{10}+0+0=x_{10}$ and $x_{20}+0+0=x_{20}$, respectively.

Case 2. One of I_1, I_2 is 0, the other 1 (row 2 or 3 in Table V) i.e., the initial states of the variables x_1 and x_2 are $x_{10}+\delta_1+0=x_{10}+0+\delta_1=x_{10}+\delta_1$ and $x_{20}+\delta_2+0=x_{20}+0+\delta_2=x_{20}+\delta_2$, respectively.

Case 3. Both I_1 and I_2 are 1 (row 4 in Table V), i.e., the initial states of the variables x_1 and x_2 are $x_{10}+\delta_1+\delta_1=x_{10}+2\delta_1$ and $x_{20}+\delta_2+\delta_2=x_{20}+2\delta_2$, respectively.

Say variable x_1 is implementing the XOR operation and variable x_2 is implementing the AND operation. Then for the XOR operation, x_{out}^i in step 3 should be 0 for cases 1 and 3.

Consequently we demand that $x_1(n) \leqslant x_1^*$ when the initial state is x_{10} (i.e., case 1) and $x_{10}+2\delta$ (i.e., case 3). For case 2, when the initial state is $x_{10}+\delta_1$, $x_1(n)$ should be $x_1^*+\delta_1$, so that after the thresholding action x_{out} will be δ_1, yielding an output of 1.

For the AND operation, implemented via the x_2 variable, x_{out}^2 in step 3 should be 0 for both cases 1 and 2. So we demand that $x_2(n) \leqslant x_2^*$ when the initial state is x_{20} (i.e., case 1) and $x_{20}+\delta$ (i.e., case 2). For case 3, when the initial state is $x_{20}+2\delta_2$, $x_2(n)$ should be $x_2^*+\delta_2$, so that after thresholding $x_{\text{out}} \equiv \delta_2$, encoding an output of 1.

TABLE VII. Inital values x_{10}, x_{20} and thresholds x_1^* and x_2^* yielding the parallel logic operation of XOR and AND, with $\delta \sim 0.7$. Here the chaotic evolution takes place over ten steps, i.e., $n=10$ in Table VI. Note that $n=1, \ldots, 4$ does not yield any suitable solution.

x_{10}	x_{20}	x_1^*	x_2^*
1.8	1.7	0.44	1.11
1.4	1.65	0.38	1.07

It is necessary to have all the six conditions described above to be satisfied *simultaneously* to implement $x_{\text{out}}^i \equiv$ XOR(I_1, I_2) and $x_{\text{out}}^2 \equiv$ AND(I_1, I_2) in parallel, since the mapping from (I_1, I_2) to $(x_{\text{out}}^1, x_{\text{out}}^2)$ must hold for all combinations of (I_1, I_2). Conversely, when these conditions are satisfied, $x_{\text{out}}^1 \equiv$ XOR(I_1, I_2) and $x_{\text{out}}^2 \equiv$ AND(I_1, I_2) hold. That is, these conditions are *necessary and sufficient for the parallel logic operations* implementing bit-by-bit arithmetic operation, namely, XOR and AND. Similar considerations for the other parallel logical operations can be straightforwardly formulated.

A summary of the necessary and sufficient conditions to be satisfied for the parallelized bit-by-bit addition operation is given in Table VI. In the representative example discussed here, we demand $\delta_1 = \delta_2 = \delta$ to be a common positive constant, so that an output from one adder can directly be fed into another adder as input, as noted earlier.

In this neuronal model, suitable *stable* solutions satisfying the necessary and sufficient conditions tabulated in Table VI can be found in a *large region of state space*. Out of the many possible implementations, one should choose those that give large responses (i.e., yield large δ) and those that lie inside a wide basin in state space, in order to have enhanced stability with respect to noise. Further, the evolution times should be kept short. This enhances both the efficiency of the operation and keeps errors from blowing up significantly. Table VII lists a few specific numerical examples. For instance, when the thresholds are set at $x_1^* \sim 0.4$ and $x_2^* \sim 1.0$, with $\delta \sim 0.7$, after an evolution over ten iterates (i.e., $n=10$ in Table VI) all the relevant gate responses are obtained parallel. Table VIII shows the variation of δ under noisy evolution and imprecise initial condition setting. Clearly the operation is robust with respect to small fluctuations.

More generally, we can have different sets of inputs for different operations. Let (I_1^1, I_2^1) be the inputs to the first variable x_1 and (I_1^2, I_2^2) be the inputs to the second variable x_2. Consider the representative example of x_1 and x_2 implementing the AND operation for two different sets of inputs simultaneously. If we want the relevant AND logic relations to hold for all sets of inputs we need the truth table given in Table IX to hold.

Again we employ the following steps to implement our logical operations.

Step 1. Initialization of the state of the system to x_{10}, x_{20} and addition of external inputs,

$$x \rightarrow x_{i0} + x_{\text{in}}^1(i) + x_{\text{in}}^2(i),$$

where $x_{\text{in}}^k(i)$ encodes I_k^i [$x_{\text{in}}^k(i) = 0$ if $I_k^i = 0$ and $x_{\text{in}}^k(i) = \delta_i$ if $I_k^i = 1$], with $k = 1, 2$.

TABLE VIII. Output values obtained for the parallel operations XOR and AND, under noise, for the representative example of the 2D neuronal system with inital states and thresholds given by the first row of Table VII. The strength of noise (both in the initial states and in the evolution) is given by η. In the case of output 0 (i.e., for input (1,1) in XOR, input (0,1) [$\equiv$(1,0)] in AND, and input (0,0) in XOR and AND) the $x(n)$ values are well below the threshold. So under small noise these states still remain below threshold, and one always obtains output 0, as desired. The value of $x_{\rm out}\sim\delta$, encoding 1, is somewhat more sensitive to noise. Shown in the table are the values $\delta_{\rm logic}^{(I_1,I_2)}$ for a particular (generic) noise realization, for the cases yielding an output of 1 (namely, for inputs (0,1) [$\equiv$(1,0)] in XOR and input (1,1) in AND). Clearly they are all within an accuracy of 0.02 [12].

η	$\delta_{\rm XOR}^{(0,1)}$	$\delta_{\rm AND}^{(1,1)}$
0	0.7037	0.7035
0.001	0.7065	0.7054
0.01	0.7245	0.7219

Step 2. Chaotic evolution for n time steps, from the initial state given above via Eq. (1).

Step 3. Threshold mechanism on the two variables to obtain the two outputs, i.e., the outputs $x_{\rm out}^1$, $x_{\rm out}^2$ (encoding O_1 and O_2 in Table IX). If $x_i(n)\leqslant x_i^*$, then $O_i=0$, and if $x_i(n)>x_i^*$ and $x_{\rm out}^1=x_i(n)-x_i^*=\delta_i$, then $O_i=1$.

In this representative example of implementing the AND operation in parallel for two distinct sets of inputs, namely, $O_1\equiv{\rm AND}(I_1^1,I_2^1)$ and $O_2\equiv{\rm AND}(I_1^2,I_2^1)$, the 16 rows in Table IX corresponding to the 16 possible combinations for values of I_1^1, I_2^1, I_1^2, I_2^2 must hold true. As before, this reduces to nine distinct cases, as the I/O relations are symmetric with respect to exchange of I_1^1, I_2^1 and I_1^2, I_2^1, and so we can combine for instance, rows 5 and 9 in Table IX into one case,

TABLE IX. The truth table for $O_1\equiv{\rm AND}(I_1^1,I_2^1)$ and $O_2\equiv{\rm AND}(I_1^2,I_2^2)$, where (I_1^1,I_2^1) is the first set of inputs and (I_1^2,I_2^2) the second set of inputs.

I_1^1	I_2^1	I_1^2	I_2^2	O_1	O_2
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	0	1	0	0
0	1	1	0	0	0
0	1	1	1	0	1
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	1

TABLE X. Necessary and sufficient conditions to be satisfied by a chaotic element in order to implement the logical operations AND in parallel on two sets of inputs.

Initial state	${\rm AND}(I_1^1,I_2^1)$	${\rm AND}(I_1^2,I_2^2)$
x_{10},x_{20}	$x_1(n)\leqslant x_1^*$	$x_2(n)\leqslant x_2^*$
$x_{10},x_{20}+\delta_2$	$x_1(n)\leqslant x_1^*$	$x_2(n)\leqslant x_2^*$
$x_{10},x_{20}+2\delta_2$	$x_1(n)\leqslant x_1^*$	$x_2(n)-x_2^*\sim\delta_2$
$x_{10}+\delta_1,x_{20}$	$x_1(n)\leqslant x_1^*$	$x_2(n)\leqslant x_2^*$
$x_{10}+\delta_1,x_{20}+\delta_2$	$x_1(n)\leqslant x_1^*$	$x_2(n)\leqslant x_2^*$
$x_{10}+\delta_1,x_{20}+2\delta_2$	$x_1(n)\leqslant x_1^*$	$x_2(n)-x_2^*\sim\delta_2$
$x_{10}+2\delta_1,x_{20}$	$x_1(n)-x_1^*\sim\delta_1$	$x_2(n)\leqslant x_2^*$
$x_{10}+2\delta_1,x_{20}+\delta_2$	$x_1(n)-x_1^*\sim\delta_1$	$x_2(n)\leqslant x_2^*$
$x_{10}+2\delta_1,x_{20}+2\delta_2$	$x_1(n)-x^*\sim\delta_1$	$x_2(n)=x_2^*+\delta_2$

rows 6 and 10 in Table IX into one case, etc. A summary of the conditions to be satisfied in order that the AND operation be implemented in parallel for any two sets of inputs is given in Table X.

It is necessary to have all the conditions described above to be satisfied simultaneously to implement $O_1\equiv{\rm AND}(I_1^1,I_2^1)$ and $O_2\equiv{\rm AND}(I_1^2,I_2^2)$ in parallel, since the mapping from $(I_1^1,I_2^1,I_1^2,I_2^2)$ to (O_1,O_2) must hold for all combinations of I_1^1, I_2^1, I_1^1, I_2^2. Conversely, when these conditions are satisfied, $O_1\equiv{\rm AND}(I_1^1,I_2^1)$ and $O_2\equiv{\rm AND}(I_1^2,I_2^2)$ hold for all I_1^1, I_2^1, I_1^2, I_2^2. That is, these conditions are necessary and sufficient for parallely implementing AND for any two sets of inputs. Similar considerations for the other parallel logical operations can be formulated straightforwardly.

Now in this neuronal model, suitable solutions for the necessary and sufficient conditions tabulated in Tables IX and X can be found. These exist around $x_{10}=0.5375$, $x_{20}=1.0835$, $x_1^*=0.231$, and $x_2^*=1.454$, with $\delta=0.0115$ and $n=20$. Note, however, that the range of solutions is smaller (and consequently, less robust) than in the previous example of parallelized bit-by-bit addition, as many more conditions have to be satisfied simultaneously now than in the previous case.

In summary, here we have doubled computing speeds, as a single 2D chaotic element can act as two processors in parallel, and thus we gain processors without increase in physical size.

V. PARALLEL LOGIC WITH A CHAOTIC LASER MODEL

Consider a chaotic Lorenz-like system, described by a set of three coupled ordinary differential equations

$$\dot{x}_1=\sigma(x_2-x_1),$$

$$\dot{x}_2=rx_1-x_2-x_1x_3,$$

$$\dot{x}_3=x_1x_2-bx_3. \qquad (2)$$

It is known that there exists a correspondence between a coherently pumped far-infrared (FIR) ammonia laser and the

TABLE XI. Truth table of the output values, the x_1, x_2, and x_3 variables must encode in order to yield the logic responses of $O_1 \equiv \mathrm{OR}(I_1,I_2)$, $O_2 \equiv \mathrm{XOR}(I_1,I_2)$, and $O_3 \equiv \mathrm{AND}(I_1,I_2)$, respectively. The outputs to the four possible input combinations of I_1 and I_2 are shown.

I_1,I_2	O_1	O_2	O_3
0,0	0	0	0
0,1	1	1	0
1,0	1	1	0
1,1	1	0	1

Lorenz system above as follows: the x_3 variable corresponds to the normalized inversion, and the x_1 and x_2 variables to normalized amplitudes of the electric field and atomic polarizations, respectively. The three parameters for the corresponding FIR NH_3 laser model are $\sigma=2$, $r=15$, and $b=0.25$. These parameter values have been obtained by detailed comparison with experiments [19]. This laser operates in the megahertz region. We will now use this 3D system to implement three operations in parallel.

As before, the three variables (x_1,x_2,x_3) of this system at various points of time τ yield various gate outputs. As a specific representative illustration, let us implement $\mathrm{OR}(I_1,I_2)$, $\mathrm{XOR}(I_1,I_2)$, and $\mathrm{AND}(I_1,I_2)$ logic gates in parallel on a pair of inputs (I_1,I_2). Implementation of other logic gates proceeds in a similar fashion.

In our example the three-dimensional state of the system at some appropriate time τ should encode $\mathrm{OR}(I_1,I_2)$, $\mathrm{XOR}(I_1,I_2)$, $\mathrm{AND}(I_1,I_2)$, i.e., $x_1(\tau)$ should encode $\mathrm{OR}(I_1,I_2)$, $x_2(\tau)$ should encode $\mathrm{XOR}(I_1,I_2)$, and $x_3(\tau)$ should encode $\mathrm{AND}(I_1,I_2)$. We thus have to implement the input-output relations tabulated in Table XI. Since we seek the parallel implementation of OR, XOR, and AND on a common set of inputs (I_1,I_2) here, we have $I_k^1=I_k^2=I_k^3=I_k$, with $k=1,2$. As in the 2D case, we again employ three steps to implement our logical operations.

Step 1. Initialization of the state of the system to x_{10},x_{20},x_{30} and addition of external inputs,

$$x \rightarrow x_{i0}+x_{\mathrm{in}}^1+x_{\mathrm{in}}^2$$

for variables $i=1,2,3$. Here x_{in}^1 encodes input I_1 and x_{in}^2 encodes input I_2. As earlier, $I_k=0$ corresponds to $x_{\mathrm{in}}^k=0$ and $I_k=1$ corresponds to $x_{\mathrm{in}}^k=\delta$.

Step 2. Chaotic evolution for time τ, from the initial state given above, via Eq. (2).

TABLE XIII. Values of initial state: x_{10},x_{20},x_{30} and thresholds x_1^*,x_2^*,x_3^* for the three variables x_1,x_2,x_3, respectively, yielding the parallel logic operations of OR, XOR, and AND, with $\delta=1.0$. Here the chaotic evolution takes place over $\tau=0.5$.

x_{10}	x_{20}	x_{30}	x_1^*	x_2^*	x_3^*
−1.9	8.3	−9.5	8.4	16.0	32.3
−1.8	7.9	−10.0	8.5	16.4	32.6

Step 3. Threshold mechanism on the three variables at time τ to obtain the three outputs, i.e., the outputs x_{out}^1, x_{out}^2, x_{out}^3 (encoding O_1, O_2, and O_3, as in Table XI). If the ith variable $x_i(\tau) \leqslant x_i^*$, then the ith output $O_i \equiv 0$, and if $x_i(\tau) > x_i^*$ and $x_{\mathrm{out}}^i=\{x_i(\tau)-x_i^*\} \sim \delta_i$, then $O_i \equiv 1$.

In this system, a large range of initial states and threshold values satisfy the conditions of Table XII yielding the OR, XOR, and AND operations in parallel. For instance, two specific numerical examples are shown in Table XIII. Again we consider a common value of $\delta_1=\delta_2=\delta_3=\delta$ for ease of concatenation of dynamical gates. Note that a continuous band of solutions exist between the two solutions shown in Table XIII, all giving the relevant gate responses in parallel, with the same δ encoding the three outputs. Typically these solutions are robust under fluctuations in the dynamics, initial states, and threshold settings. Also note that the τ here is as low as one-tenths of the typical periodicity of the system, and so the operation is fast. Further, the short evolution times and large values of δ help to keep the operations robust (as in the 2D example). Table XIV displays the output from a typical noisy evolution of the elements. Clearly the output is always well within acceptable tolerance.

It is then conceivable that choosing faster chaotic dynamics, such as ultrafast optical components operating in the gigahertz regime, will enhance computational speeds beyond those currently available. For instance, in principle, using semiconductor lasers or fiber lasers yielding chaotic subnanosecond/subpicosecond pulses [20], one could perhaps reach speeds of 10^{10} logic operations per second.

VI. DISCUSSION

Our aim was to construct general multipurpose programmable hardware out of chaotic elements. Further, we aimed to increase computational speeds through the exploitation of parallel computing architectures that utilize the many dynamical states available to chaotic systems. The direct and flexible parallel implementation of the functions proposed

TABLE XII. Necessary and sufficient conditions to be satisfied by a 3D chaotic laser in order to implement the logical operations OR, XOR, and AND in parallel. Here variable x_1 is implementing the OR operation, variable x_2 is implementing the XOR operation, and variable x_3 is implementing the AND operation.

Initial state	OR	XOR	AND
x_{10},x_{20},x_{30}	$x_1(\tau) \leqslant x_1^*$	$x_2(\tau) \leqslant x_2^*$	$x_3(\tau) \leqslant x_3^*$
$x_{10}+\delta_1,x_{20}+\delta_2,x_{30}+\delta_3$	$x_1(\tau)-x_1^* \sim \delta_1$	$x_2(\tau)-x_2^* \sim \delta_2$	$x_3(\tau) \leqslant x_3^*$
$x_{10}+2\delta_1,x_{20}+2\delta_2,x_{30}+2\delta_3$	$x_1(\tau)-x_1^* \sim \delta_1$	$x_2(\tau) \leqslant x_2^*$	$x_3(\tau)-x_3^* \sim \delta_3$

TABLE XIV. Output values obtained for the parallel operations OR, XOR, and AND, under noise, for the representative example of the laser system with initial states and thresholds given by the first row of Table XIII. The strength of noise (both in the initial states and in the evolution) is given by η. In the case of output 0 (i.e., for input (1,1) in XOR, input (0,1) [$\equiv$(1,0)] in AND, and input (0,0) in OR, XOR, and AND) the $x(\tau)$ values are well below the threshold. So, under small noise these states still remain below threshold, and one always obtains output 0, as desired. The value of $x_{\rm out}\sim\delta$, encoding 1, is somewhat more sensitive to noise. Shown in the table are the values $\delta_{\rm logic}^{(I_1,I_2)}$, for a particular (generic) noise realization, for the cases yielding an output of 1 (namely, for input (0,1) [$\equiv$(1,0)] in XOR and OR, and for input (1,1) in OR and AND). Clearly they are all within an accuracy of 0.05 around $\delta=1.0$ [12].

η	$\delta_{\rm OR}^{(0,1)}$	$\delta_{\rm XOR}^{(0,1)}$	$\delta_{\rm OR}^{(1,1)}$	$\delta_{\rm AND}^{(1,1)}$
0	0.999	1.019	0.984	0.996
0.001	1.002	1.007	0.985	0.998
0.01	1.007	0.961	0.975	0.973

here can then serve as simple and cost effective key ingredients of a computing system, for instance, serving flexibly as the basis for bit-by-bit arithmetic addition, and as a basic component of computer memory. With these fundamental ingredients in hand it is conceivable to build simple, fast, cost effective, and general-purpose computing devices, which are more flexible than wired hardware.

It is illuminating to contrast our use of chaotic elements with the possible use of periodic elements on one hand, and random elements on the other. It is not possible to extract all the *different* logic responses from the *same* element in case of periodic components, as the temporal patterns are inherently very limited. So, periodic elements do not offer much flexibility or versatility. Random elements on the other hand have many different temporal sequences. But they are *not deterministic* and so one cannot use them to *design* components. Only chaotic dynamics enjoys both richness of temporal behavior as well as determinism. Here we have shown how one can select out temporal responses corresponding to different logic gate patterns from such dynamics, and this ability allows us to construct programmable gates [21].

The basic idea here is not to expend effort and energy trying to eliminate the complicated inherent dynamics, but to exploit it instead. The intrinsic dynamics of our components then need not be forced into unnatural patterns such as transistor-transistor logic pulses but are allowed to operate with their natural dynamics to perform operations. This makes for possibly more robust system behavior.

In some sense the knowledge of the dynamics has allowed us to reverse engineer and obtain what must be done in order to select out the temporal patterns emulating the different gates. The proposed methods (which constitute a nonfeedback control, utilized as a programming scheme) once designed, need no further run-time effort. For instance, here the same element can be made to operate as different gates by simply choosing a suitable threshold, which is made available as a look-up table. The thresholding does not change the natural dynamics but operates as a resetting of the state of the system.

It is not appropriate at this incipient stage to debate the optimality of computing with chaos. The interesting inference one can draw at this point is the feasibility of chaos as a candidate for direct and controlled computing and its evident potential. This is quite like the situation in the more ''mature'' fields of DNA [8] and quantum computing [9] where too its still not clear whether these computing systems, first presented as *alternate computing paradigms*, can perform better than digital computers (although they hold great promise) [22]. In contrast to the DNA and quantum paradigms, which are geared to handle *specific* problems suited specially to themselves, we are aiming at a general-purpose machine. Further, chaos computing has an advantage (unlike, say DNA computing, which is limited by slow biological processes) in that here one is quite free to design and exploit (almost) any fast dynamical system. So we can choose from a wide variety of chaotic systems, ranging from fast electronic circuits to fast lasers, and this will have direct relevance for the operational speeds attainable in experimental realizations [23].

In summary, we have demonstrated the basic principles for the direct and flexible implementation of all basic logical operations utilizing low dimensional chaos. Then we generalized the concept to high dimensional chaotic systems and demonstrated the parallelism inherent in such systems. The power of the scheme is evident when one notes that even a two-dimensional system, when utilized in the manner indicated here, *doubles* computational speeds. It is conceivable then that such architectures may serve as flexible ingredients of a fast and cost effective general-purpose computing device.

[1] C. Moore, Phys. Rev. Lett. **64**, 2354 (1990).

[2] A. V. Holden *et al.*, Chaos **2**, 367 (1992). In this work it was straightforwardly shown that coupled map lattices are equivalent to synchronous concurrent algorithms, which is a model used in computer science to give a unified theory for a number of models of parallel computing (including neural nets).

[3] H. T. Siegelmann and S. Fishman, Physica D **120**, 214 (1998); H. T. Siegelmann, A. Ben-Hur, and S. Fishman, Phys. Rev. Lett. **83**, 1463 (1999).

[4] D. Hansel and H. Sompolinsky, Phys. Rev. Lett. **68**, 718 (1992).

[5] N. Margolus, Physica D **10**, 81 (1984); T. Toffoli and N. Margolus, *Cellular Automata Machines: A New Environment for Modeling* (MIT Press, Cambridge, MA, 1987); T. Toffoli and N. Margolus, Physica D **47**, 263 (1990).

[6] J. P. Crutchfield and K. Young, Phys. Rev. Lett. **63**, 105 (1989); J. P. Crutchfield, Physica D **75**, 11 (1994). The above efforts by Crutchfield and co-workers were primarily focused on the *alternate* question: can the theory of computation help describe/quantify the complexity of physical systems? Here we

are approaching quite the opposite question. We want to *harness* (in an explicit realization) a very complex physical system to do computations for us (in a controlled, direct, and hopefully clean and simple fashion). So while Crutchfield's work has bearing on the computational effort required in modeling complex behavior, which is used to quantify the emergence of complexity, it does not exploit the computational capability of nonlinear processes to solve ''tasks'' extrinsic to it.

[7] S. Sinha and W. Ditto, Phys. Rev. Lett. **81**, 2156 (1998); Phys. Rev. E **60**, 363 (1999).

[8] L. M. Aldeman, Science **266**, 1021 (1994); R. Pool *ibid.* **268**, 498 (1995).

[9] P. W. Shor, in *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science*, edited by S. Goldwasser (IEEE Computer Society Press, Los Alamitos, CA, 1994), p. 124; A. Steane, Rep. Prog. Phys. **61**, 117 (1998); A. Barenco, Contemp. Phys. **37**, 375 (1996).

[10] M. M. Mano, *Computer System Architecture*, 3rd ed. (Prentice-Hall, Englewood Cliffs, NJ, 1993); T. C. Bartee, *Computer Architecture and Logic Design* (McGraw-Hill, New York, 1991).

[11] The interpreted and actual values are somewhat analogous to *logical* and *physical* values in conventional computers, where the logical 0 and 1 most often correspond to electric voltage or current in a physical device whose values are interpreted as 0 and 1.

[12] Note that the δ is defined up to some accuracy ϵ, i.e., $|x_{\rm out}-\delta|\leqslant\epsilon$. In these numerical examples $\epsilon\sim 0.05$.

[13] Our efforts are in contrast to the earlier efforts of A. Toth and K. Showalter, J. Chem. Phys. **103**, 2058 (1995), who obtained gates from chemical systems by delicately tuning many parameters (involving both the construction of the apparatus as well as the geometric configuration and timing of the input and output waves). Fine adjustments of these lead to the desired phenomena. In contrast, here we have one adjustable parameter defining all the gates from the same system.

[14] Patterns, *Information and Chaos in Neuronal Systems*, edited by B. J. West (World Scientific, Singapore, 1993), and references therein.

[15] R. Traub and R. Miles, *Neuronal Networks of the Hippocampus* (Cambridge University Press, New York, 1991).

[16] Realistically the degree of parallelism will be limited by the number of variables, which can be observed, and preferably manipulated. Note, however, that even indirect (perhaps, model based) assessments of the state variables would be quite adequate here, especially since the specific schemes we will propose are based only on coarse grained values, not on high precision characterizations. Of course the power of the scheme is evident when one notes that even a two-dimensional system, when utilized in the manner indicated, *doubles* computational speeds.

[17] D. R. Chialvo and A. V. Apkarian, J. Stat. Phys. **70**, 375 (1993).

[18] K. Aihara and G. Matsumoto, in *Chaos*, edited by A. R. Holden (Manchester University Press, Manchester, 1986), pp. 257–269.

[19] U. Hubner, N. B. Abraham, and C. O. Weiss, Phys. Rev. A **40**, 6354 (1989); C. O. Weiss *et al.*, Appl. Phys. B: Lasers Opt. **B61**, 223 (1995).

[20] I. Fischer *et al.*, Phys. Rev. Lett. **76**, 220 (1996); Q. L. Williams, J. Garcia-Ojalvo, and R. Roy, Phys. Rev. A **55**, 2376 (1997).

[21] G. Taubes, Science **277**, 1935 (1997).

[22] In fact after years of intense research, some fundamental problems still remain in DNA and quantum computing paradigms. For instance, quantum computers are exponentially more sensitive to external noise and to slight deviations of their components from their design specifications. Quantum computing relies heavily on the quantum correlations of entangled states, and these are extremely sensitive to thermal noise, which would tend to randomize each bit separately, making the whole system ''decohere.'' All pure states of a quantum system are connected by a continuous symmetry, so there are no discrete positions in which to stabilize such states. Moreover, any mechanism working ''locally'' on each qubit separately would inevitably destroy entanglement. Controlling large scale entanglement is thus a fundamental issue in quantum computing, which is yet to be solved. In DNA computing, there is the problem of really slow molecular biological operations, and the hazard of fracture in the DNA molecule. (Unlike biological computers, we are fortunately free to design and use very fast dynamical systems, ranging from electronic circuits to chaotic lasers). There is also the inability to transmit information from one molecule to another in a DNA computer, which limits their flexibility. Further DNA computing involves somewhat random operations, that is to say, inherently noisy components (unlike the determinism of our prototype).

[23] Of course, the speed of these devices will depend strongly on how they are implemented. To make the proposal work well we need to design elements where the threshold can be set/reset easily, and this implementation issue also sets the possible limitations.