

Transactions Briefs

Chaos Computing: Implementation of Fundamental Logical Gates by Chaotic Elements

Toshinori Munakata, Sudeshna Sinha, and William L. Ditto

Abstract—Basic principles of implementing the most fundamental computing functions by chaotic elements are described. They provide a theoretical foundation of computer architecture based on a totally new principle other than silicon chips. The fundamental functions are: the logical AND, OR, NOT, XOR, and NAND operations (gates) and bit-by-bit arithmetic operations. Each of the logical operations is realized by employing a single chaotic element. Computer memory can be constructed by combining logical gates. With these fundamental ingredients in hand, it is conceivable to build a simple, fast, yet cost effective, general-purpose computing device. Chaos computing may also lead to dynamic architecture, where the hardware design itself evolves during the course of computation. The basic ideas are explained by employing a one-dimensional model, specifically the logistic map.

Index Terms—Chaos computing, new computing paradigm, one-dimensional (1-D) chaotic systems.

I. INTRODUCTION

Recently, ideas of computer architecture based on totally new principles other than silicon chips have been proposed. They include quantum [1], [2], DNA/RNA [3], [4], and various forms of nanocomputing. Nanocomputing deals with the computing aspect of the broader field of nanotechnology, which studies nanoscale phenomena typically in order of 10^{-7} m = 0.1 micron or smaller in one dimension. Examples of nanocomputing are organic molecular computing [5], atomic scale circuitry [6], and carbon nanotube transistors [7]. While practical, everyday-use of these proposals as computing devices is yet to be seen, these ideas have stimulated the scientific community due to their fundamental nature, novelty, and potentials for new forms of information processing and applications. Some of the new technologies are hoped to complement or replace the current computer architecture based on silicon chips. In this brief, we introduce the fundamental theory of another new architecture paradigm called *chaos computing*.

In September 1998, Sinha and Ditto proposed a new computing scheme based on chaotic dynamical systems [8]. They showed that certain computing functions, such as logical NOR (that is, NOT OR) and integer arithmetic operations, can be achieved by using a two chaotic element model. Later, they developed a single chaotic element model, achieving the same types of computing operations [9]. Simply stated, a chaotic element receives external input, say, X and Y . As a dynamical system, the element can exhibit chaotic internal behavior influenced by the input. It then can output, for example, Z , to the outside of

the element. Desirable input-to-output mapping can be achieved by utilizing the complex dynamics of the individual elements, as well as their interactive couplings and adaptive processes, implemented in particular as a threshold mechanism.

In general, the study of chaotic systems, or chaos in short, has attracted much attention for the past 30 years or so. Chaos represents a deterministic dynamical system that is nonlinear, sensitive to initial conditions (the so-called butterfly effect), and exhibits sustained irregularity. A simple example of a chaotic system is a pseudo-random number generator. It employs a nonlinear deterministic equation such as $x_{n+1} = cx_n$ modulo m , where c and m are constants. Although this underlying rule is simple, the resulting solutions, i.e., the pseudo-random numbers, are very irregular (the more irregular, the better the random numbers). Also, a small change in the initial condition (seed) can yield a significantly different sequence of random numbers. We exploit these unique characteristics of chaotic systems to implement chaos computing as we will see.

Chaos is all around us and found in many disciplines, such as physics, chemistry, biology, medicine, and engineering. For example, chaotic phenomena are found in lasers, electronic circuits, chemical systems, brains and hearts. Many researchers have worked in the field because of both theoretical and practical importance of the subject, and challenging and fascinating features of extreme nonlinearity [10], [11]. In particular, 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 improved performance. Successful implementations of this concept have included the exploitation of chaotic behavior for control [12], synchronization [13], [14], encoding information [15], and communications [16].

In this brief, we will show how we can directly implement the most fundamental computer functions by chaotic elements. In particular, we will illustrate: 1) the basic logical operations, AND, OR, NOT, XOR (Exclusive OR), and NAND (NOT AND) gates; 2) bit-by-bit arithmetic operations such as addition; and 3) construction of computer memory based on integrated-circuit design. These three types of items are necessary and sufficient ingredients of the most basic components to build a computer. This brief lays out the first major step of theoretical foundations of the chaos computer. The general nature of the technique will allow a universal form of computing, rather than a narrowly specialized problem domain. With abundant chaotic systems in engineering and nature, this may lead to a simple, yet very fast and cost effective computer. The basic ideas are explained by employing a one-dimensional (1-D) model, specifically the logistic map. Extensions to higher dimensional models are discussed in separate papers [17], [18].

We note that the basic logical operations, AND, OR, NOT, XOR, and NAND, can be constructed by combining the NOR operation proposed in [8], [9]. For example, the AND operation can be realized by $\text{AND}(X, Y) = \text{NOR}(\text{NOR}(X, X), \text{NOR}(Y, Y))$. However, such conversion processes are inefficient in comparison with the direct implementation of the AND operation, considering perhaps such operations may be performed billions of times. The direct implementation of each logical operation discussed in this brief, whichever employed, will guarantee the best efficiency for that operation. We also note that once we achieve bit-by-bit arithmetic addition, other types of bit-by-bit operations can be implemented as derivatives of addition. For example, subtraction can be done as the 2-s complement operation

Manuscript received August 16, 2000; revised December 12, 2001 and May 24, 2002. The work of W. L. Ditto was supported by the Office of Naval Research under Grant N00014-00-1-0020. This paper was recommended by Associate Editor R. Sridhar.

T. Munakata is with the Computer and Information Science Department, Cleveland State University, Cleveland, OH 44114 USA.

S. Sinha is with the Institute of Mathematical Sciences, C.I.T. campus, Chennai 600113, India.

W. L. Ditto is with the Biomedical Engineering Department, University of Florida, Gainesville, FL 32611-6131 USA

Digital Object Identifier 10.1109/TCSI.2002.804551

of addition; multiplication can be done as repeated addition, and division as repeated subtraction.

II. BASIC LOGICAL OPERATIONS (GATES) WITH CHAOTIC ELEMENTS

As shown in Fig. 1(a) and (b), we consider a single chaotic element (represented by a circle) for each kind of the logical operations. Each element is a chaotic dynamical system, whose state may be represented by a value of x (for example, $0 \leq x \leq 1$). An element receives either two external input signals I_1 and I_2 or one external input signal I , and yields one output signal O . The former type, having two external input signals I_1 and I_2 , is employed for the logical AND, OR, XOR, and NAND operations. The latter type, having one external input signal I , is employed for the NOT operation. Each of I_1 , I_2 and O can assume either 0 (representing either binary 0 or logical False) or 1 (binary 1 or logical True). The logical operations are defined by patterns of input-to-output mapping, represented by the truth table. For example, the (I_1, I_2, O) triplet values for the AND $(I_1, I_2) \rightarrow O$ mapping are: (0, 0, 0), (0, 1, 0), (1, 0, 0) and (1, 1, 1). The (I, O) values for the NOT $(I) \rightarrow O$ are (0, 1) and (1, 0). Also shown in Fig. 1(c) is configuration for bit-by-bit arithmetic addition, which will be discussed later. In this configuration, two chaotic elements are employed; there are two external inputs, I_1 and I_2 , for each element, and two external outputs, O_1 and O_2 .

The external inputs and output (I/O) such as I_1, I_2 and O described in the preceding paragraph, where each assumes either 0 or 1, are “interpreted” or “virtual” values. In chaotic dynamical systems, we have corresponding “actual” input values denoted as X, X_1, X_2 , and so on, and actual output values denoted as Z, Z_1, Z_2 , etc. The “actual” values here are normalized rather than true physical values, and they range over $[0, 1]$ including fractions. The relationships between interpreted and actual values, (interpreted \leftrightarrow actual), are: (0 \leftrightarrow 0) and (1 \leftrightarrow δ). In this brief, δ is a positive constant derived during a process in chaotic dynamical systems, and will be explained later. The interpreted and actual values are somewhat analogous to logical and physical values in conventional computers. The logical 0 and 1 may correspond to electric voltage or current, which assumes a value other than 0 and 1 of a physical device.

In our implementation, we demand that the relationships between interpreted and actual values will be *uniform* for both input and output, as well as among the various logical operations such as AND, OR, etc. This requires that constant δ assumes the same value throughout a network of chaotic elements for any input and output value combinations. 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. With this conversion from interpreted to actual, the truth table for our basic logical operations can be defined. For example, for two inputs (X_1, X_2) , there are four possible input value combinations: $(X_1, X_2) = (0, 0), (0, \delta), (\delta, 0), (\delta, \delta)$. The corresponding outputs for, for example, the AND gate should be 0, 0, 0, δ . Our problem is to design chaotic elements that perform these desired logical operations, that is, elements that yield appropriate output for given input.

Each element is a small chaotic dynamical system—a “chaotic chip” or “chaotic processor” so to speak. It changes its state x internally based on a chaotic function $f(x)$. More specifically, x_n , the x value at discrete time n , determines x_{n+1} , the x value at discrete time $n + 1$, as $x_{n+1} = f(x_n)$. This can further be extended to later time steps, as for example, $x_{n+2} = f(x_{n+1}) = f(f(x_n)) = f^2(x_n)$, and so on, or, without writing a specific time, we can say that if the state is x at a certain time, it will change to $f(x)$ at the next time step. We use these two representations, $x_{n+1} = f(x_n)$ and $x \rightarrow f(x)$, interchangeably. The element can also accept inputs from outside and add the input to the internal chaotic state. Finally, the element can emit a

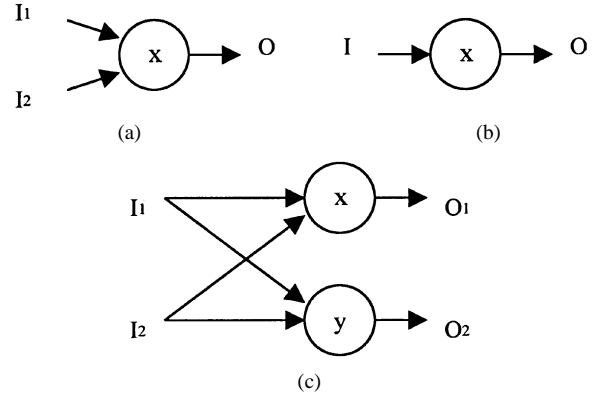


Fig. 1. Three types of input/output configurations: (a) logical AND, OR, XOR, and NAND; (b) logical NOT; (c) bit-by-bit arithmetic addition.

signal to outside as its output based on its state. More specifically, given a threshold value x^* , if $f(x) \leq x^*$ no external output, i.e., $Z = 0$; otherwise $Z = \delta \equiv f(x) - x^*$ will be emitted to outside. Utilizing all these characteristics plus the chaotic behavior of function $f(x)$ itself, we implement the fundamental logical operations as follows in three steps.

Step 1) Initialization of the state to x_0 and addition of external inputs

$$x = x_0 + X_1 + X_2 \text{ for the AND, OR, XOR,}$$

and NAND operations;

$$x = x_0 + X \text{ for the NOT operation.}$$

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

Step 3) Threshold mechanism to obtain output, i.e., the output Z is

$$Z = 0 \text{ if } f(x) \leq x^*; \quad Z = \delta = f(x) - x^* \text{ if } f(x) > x^*.$$

Having laid out our three-step procedure for a chaotic element, the next step is to design our system in such a way that it yields the desired input-to-output mapping. More specifically, given a chaotic function $f(x)$, we want to have the three parameters, δ, x_0, x^* , to be consistent with the procedure and also to achieve the required mapping. Take, for example, the case of $Z = \text{AND}(X_1, X_2)$. There are four possible input value combinations: $(X_1, X_2) = (0, 0), (0, \delta), (\delta, 0), (\delta, \delta)$, which should yield outputs of: 0, 0, 0, δ , respectively. For our current purpose, however, only the following three cases are sufficient, combining the second and third input combinations, $(0, \delta)$ and $(\delta, 0)$, into Case 2).

Case 1) Both X_1, X_2 are 0. In this case, x in Step 1) is $x_0 + 0 + 0 = x_0$, and $f(x)$ in Step 2) is $f(x_0)$. Since Z in Step 3) should be 0 in this case, we must have $f(x_0) \leq x^*$.

Case 2) One of X_1, X_2 is 0, the other δ , i.e., $(X_1, X_2) = (0, \delta)$ or $(\delta, 0)$. In this case, x in Step 1) is $x_0 + \delta$, and $f(x)$ in Step 2) is $f(x_0 + \delta)$. Since Z in Step 3) is 0 in this case for AND, we must have $f(x_0 + \delta) \leq x^*$.

Case 3) Both X_1, X_2 are δ . In this case, x in Step 1) is $x_0 + 2\delta$, and $f(x)$ in Step 2) is $f(x_0 + 2\delta)$. Since Z in Step 3) is δ in this case, first we must have $f(x_0 + 2\delta) > x^*$. Furthermore, $f(x) - x^* = f(x_0 + 2\delta) - x^*$ must end up with the value of $\delta > 0$ consistent with the value of δ in other places. Combining these two requirements, we write condition for Case 3) to be $f(x_0 + 2\delta) - x^* = \delta$.

It is necessary to have all the three conditions, $f(x_0) \leq x^*$, $f(x_0 + \delta) \leq x^*$, and $f(x_0 + 2\delta) - x^* = \delta$, to be satisfied simultaneously to implement $Z = \text{AND}(X_1, X_2)$, since mapping from (X_1, X_2) to Z must hold for all combinations of (X_1, X_2) . Conversely, when these three conditions are satisfied, $Z = \text{AND}(X_1, X_2)$ holds. That is, the three conditions are necessary and sufficient for $Z = \text{AND}(X_1,$

TABLE I

NECESSARY AND SUFFICIENT CONDITIONS TO BE SATISFIED BY A CHAOTIC ELEMENT TO IMPLEMENT EACH OF THE LOGICAL OPERATIONS. ROW 1 CORRESPONDS TO INPUT $(X_1, X_2) = (0, 0)$, ROW 2 $(0, \delta)$ OR $(\delta, 0)$, AND ROW 3 (δ, δ) . IN THIS TABLE, δ IS A COMMON POSITIVE CONSTANT FOR ALL THE OPERATIONS, WHEREAS, x_0 AND x^* DIFFER AMONG THE OPERATIONS

Operation	AND	OR	XOR	NAND	NOT
Conditions	$f(x_0) \leq x^*$	$f(x_0) \leq x^*$	$f(x_0) \leq x^*$	$f(x_0) = \delta$	$f(x_0) - x^* = \delta$
	$f(x_0 + \delta) \leq x^*$	$f(x_0 + \delta) - x^* = \delta$	$f(x_0 + \delta) - x^* = \delta$	$f(x_0 + \delta) = \delta$	$f(x_0 + \delta) \leq x^*$
	$f(x_0 + 2\delta) - x^* = \delta$	$f(x_0 + 2\delta) - x^* = \delta$	$f(x_0 + 2\delta) \leq x^*$	$f(x_0 + 2\delta) - x^* \leq x^*$	

X_2). Similar considerations for the remaining logical operations lead to Table I, a summary of necessary and sufficient conditions to be satisfied by each of the operations:

As noted earlier and in Table I, we demand δ to be a common positive constant for all the operations so that an output from one gate can directly be fed into another gate as input. On the other hand, x_0 and x^* differ among the operations, although the same symbols are used for simplicity. In this context, x_0 and x^* for AND, for example, can be denoted as $x_{\text{AND},0}$ and x_{AND}^* .

We also note that the conditions imposed in Table I are looser constraints than the ones where the inequalities were replaced with equations. Typically, we may start selecting function $f(x)$. This may be determined by characteristics of a physical device for actual implementation. Given a specific function, we may search for solutions in terms of δ , since this must be a constant throughout all the operations. We determine a pair of (x_0, x^*) for each of the operations consistent with δ , where δ may have to be chosen appropriately so that solutions for all operations exist. We note that our computing scheme under many chaotic systems will be robust for background noise.

III. CASE STUDY: THE LOGISTIC MAP

A basic procedure to construct logical gates using chaotic elements is prescribed in the previous section. In this section, we apply the procedure for a specific chaotic function called the logistic map. In this way, the scheme will be clearer. We also consider specific numeric examples.

The general form of the logistic map is given by the following equation:

$$f(x) = rx(1-x), \quad 0 < r \leq 4. \quad (1)$$

This is a well-known 1-D function for its historic background and the chaotic characteristics it can exhibit despite its simplicity [10]. The behavior of the steady-state solution of this map depends on the value of parameter r . For $0 < r < 1$, the steady-state solution is 0; for $1 \leq r < 3$, a fixed point; for $3 \leq r < 3.57$, periodic; and finally for $3.57 \leq r \leq 4$, chaotic. One can test out easily how the steady-state solutions of (1) are chaotic, i.e., the successive values of x are very irregular and a small change in the initial condition yields a significantly different sequence of x , somewhat similar to the pseudo-random numbers discussed before. This map has widespread relevance to physical and biological chaotic phenomena. For example, electrical circuits and nonlinear oscillators can exhibit such phenomena. In the following, we consider a special case of the logistic map by selecting $r = 4$. Although other values of r close to 4 can also be used, $r = 4$ guarantees that the function is chaotic

$$f(x) = 4x(1-x). \quad (2)$$

Under this consideration, the expressions in Table I become more explicit, for example, $f(x_0) = 4x_0(1-x_0)$.

TABLE II

NUMERIC EXAMPLES FOR x_0 AND x^* FOR $f(x) = 4x(1-x)$ AND $\delta = 1/4$

Operation	AND	OR	XOR	NAND	NOT
x_0	0	1/8	1/4	3/8	1/2
x^*	3/4	11/16	3/4	11/16	3/4

Numeric Examples

We select the constant δ , common to both input and output and all logical gates, to be 1/4. For AND operation, for example, selecting $x_0 = 0$ and $x^* = 3/4$ suffices the three conditions given in Table I as follows:

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

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

$$f(x_0 + 2\delta) - x^* = f(1/2) - 3/4 = 1 - 3/4 = 1/4 = \delta. \quad (5)$$

Table II shows such numeric values for x_0 and x^* for the logical gates under consideration.

IV. BIT-BY-BIT ADDITION AND MEMORY

We can directly obtain bit-by-bit arithmetic addition from two chaotic elements using the gate implementations discussed above. This is the most fundamental form of arithmetic operation. Other types of arithmetic operations can then easily be performed using this basic addition or similar operation. For example, addition of larger numbers (e.g., addition of two 32-b numbers) can be carried out by extending the bit-by-bit operation to higher number of bits. Subtraction can be done as addition of the complement numbers. Multiplication can be achieved as repeated addition or its variations; similarly, division can be done as repeated subtraction.

In Fig. 1(c), I_1 and I_2 are the two binary numbers to be added and O_1 is the first (rightmost) digit of the sum, and O_2 is the carry of the answer to the next digit. These values can easily be determined by employing the operations discussed above as: $O_1 = \text{XOR}(I_1, I_2)$ and $O_2 = \text{AND}(I_1, I_2)$. Furthermore, 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 [19], [20].

V. IMPLEMENTATION

Successful employment of chaos computing requires two major steps: 1) development of a theoretical foundation and 2) physical implementation. Step 1) is the major focus of this brief. We will defer most of Step 2) principally because it involves extensive issues very specific to the nature of the chaotic system employed. Since chaotic systems can range from electronic and optical to biological it is not possible to discuss all these very different implementation scenarios here. While this brief has laid down the general framework it is well beyond its scope to discuss specific implementations comprehensively

as well. We note that, in general, certain time lags between development of fundamental theories and implementation are very common, particularly for totally new ideas. In the following, we will only sketch some major implementation issues and their possible future directions.

We start with discussing the current state of characterizing chaotic systems and communicating with them. “Communicating” here means to *input* controlled signals (for example, in form of a wave of electrical and laser pulses) into a chaotic system with appropriate parameter setting, then detect *output* signals (for example, optical, electric, etc.) transmitted from the system. This is exactly the basis of the idea of chaos computing too. There have been several reports of controlled use of various chaotic physical devices. For instance, chaotic electrical circuits have been extensively characterized and used for communications (see [21] for a review). Chaotic lasers too have been widely reported. For example, Weiss *et al.* reported an exact correspondence between the far-infrared megahertz ammonia laser and a chaotic Lorenz-like system [22], [23], and more recently, Van Wiggeren *et al.* demonstrated communication with chaotic lasers [16]. In short, such wide ranging experiments with chaotic systems suggests that the chaotic computing schemes discussed in this brief will be feasible.

Future tasks and possibilities include the following.

1) *Selection and design of the electro physical system:* There are abundant physical system candidates for chaos computing, such as electrical circuits and nonlinear oscillators mentioned above. In general, selecting appropriate forms of input and output and a material of the system element may require extensive search and experiment. For example, ultrafast semiconductor lasers may be considered as candidates for all-optical implementations of chaos computing. The frequency and intensity of the laser beam may depend on what material is used as the system element.

2) *The scale of the system:* Chaotic systems may be realized at various scales such as macroscopic, microscopic and nanoscale. At macroscopic level, each system is at a visible scale, say, 1 mm to 10 cm. A system of this size may be easy to deal with, thus it may give a good starting point. The system can conceivably be much smaller than the macroscopic level, for example, at the order of a micron or 10^{-6} to 10^{-7} m. It can be further smaller at, for example, molecular level (10^{-9} m) or at the atomic level (10^{-10} m). Of course, how to deal with a chaotic system of this scale itself is a major issue. For the past couple of years, there has been significant development on nanotechnology [7]. Chaos computing at microscopic and nanoscale levels will be closely related to nanotechnology. In addition to the problem of dealing with manipulation of the chaotic element of this size, quantum effect would have to be considered at the nanoscale level. Such a scheme may overlap on both chaos and quantum computing.

3) *Fabrication of chaotic computing networks:* For an experimental study of the above two items, we may start with a system consisting of a single chaotic element for simplicity. The next phase will be a network of multiple chaotic elements. How to connect and operate on these elements, i.e., the system architecture, will be the next major problem.

It is not appropriate to discuss a specific order of expected computing speed in detail at this incipient stage. However, it is conceivable that choosing fast chaotic dynamics, such as ultra-fast optical components operating in the gigahertz or terahertz regime, will enhance computational speeds beyond those currently available. For instance, in principle, using semiconductor lasers or fiber lasers yielding chaotic sub nanosecond/picosecond pulses, one could perhaps reach speeds of 10^{10} to 10^{12} logic operations per second [24], [25]. It is premature at this stage to make exact comparisons with other computing paradigms. The latter point is also true to some extent in other new computing paradigms such as DNA and quantum. That is, which computing paradigm is superior to which one in what application domains is still an unsettled question.

VI. DYNAMIC ARCHITECTURE

It turns out that chaos computing can have a unique feature called dynamic architecture. Imagine a chaotic computing physical system, which is the “hardware.” Because of the characteristics of chaos, we can achieve various logic gates, adders and memory as discussed in this brief by applying small changes to the parameters on the *same* hardware. The flexibility of achieving various operation types by chaos computing may lead to several new computing architecture concepts. Almost all other computing paradigms do not have such flexibility, which may be termed as *static* architecture. For example, most gates are currently built as fixed hardware on a silicon chip and cannot be changed to other logic gates.

One exception to this is the architecture generally called field-programmable gate array (FPGA). There are several different types for FPGA architectures. One type employs fuses and antifuses as switches to make or break circuit connections for one-time programming [26]. We may call this type *one-time configurable* architecture. In some other FPGA one can set a specific architecture configuration every time it is used on the same hardware. We may call this *multi-time configurable* architecture. Furthermore, some of more recent FPGA systems allow the hardware changes or evolves during computation, by “rewiring” tiles or computer elements [27]. Generally, systems that change their hardware configurations during the course of computation can be termed as *dynamic* architecture. In particular, the last example may be called *dynamic rewiring* architecture.

Chaos computing is particularly apt to all these types of variable architectures because of its flexibility for changing logic gates by slightly modifying its parameter values. Note that in chaos computing the architecture can be modified by changing the type of each gate, for example, from an AND gate to an OR gate, rather than rewiring the circuit during computation. Such scheme by chaos computing may be called *dynamic logic* architecture. Let G_0 be a specific chaos computing network configuration at time 0. It can be changed to another configuration G_1 during computation. It can further be changed to another configuration G_2 , and so on. Such configuration changes can be implemented either by a predetermined schedule, *predetermined dynamic logic architecture*, or by the outcome of computation, *outcome-dependent dynamic logic architecture*.

In the context of computation this scheme of changing the types of gates during computation may appear strange. One analogy may be to computer languages which allow program changes during execution. Conventional languages such as Fortran and C do not allow program changes during execution. So-called symbolic languages however, such as Prolog and Lisp, typically employed in artificial intelligence, allow program changes during execution. In this context, chaos computing is analogous to symbolic languages while other common computing paradigms can best be compared to conventional languages. Practical applications of such form of dynamic architecture are yet to be explored, but certainly this will open up a totally new concept of computing architecture.

VII. DISCUSSION

One of the most common questions concerning chaos computing is the cost associated with the computation. We can consider two kinds of cost: one is the “construction cost” of a physical device, and the other the “operational cost.” It is premature to make any concrete discussion on either type of cost without a specific implementation scenario. However, we think that chaos computing in general will be relatively cost effective for the following reasons. First, we have many potential chaotic systems to choose from, considering the cost, speed, and so on. For the construction cost, chaos computing will likely not have to rely on extraordinary environments such as extreme low temperature,

high pressure, high magnetic field, etc. For example, generating a laser beam in general is inexpensive (e.g., a CD drive). The operational cost should not be significant either. The reason is that the device is intrinsically a chaotic physical system, a feature that is at the heart of the chaos computing idea. For the example discussed here it means that the state of the device is naturally updated according to the given function $f(x)$ in Table I. So Step 2) of the implementation takes place automatically and does not present additional computation overheads, an advantage that conceivably hold for Step 3) of the threshold mechanism.

The basic idea here then, 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 standard square wave pulses but allowed to operate with their natural dynamics to perform operations. This makes for possibly more robust and cost-effective 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 natural temporal patterns emulating the different gates. The proposed methods (which constitute a nonfeedback control, utilized as a programming scheme) once designed, needs no further run-time effort.

Another common question is why employ chaos. Precisely speaking, what is crucially needed is a dynamical device whose state evolution is characterized by a strongly nonlinear function. This is true in chaotic physical systems and so they are natural candidates for the scheme. Importantly note that this computing scheme yields the outputs for all the different gates for all sets of inputs from the *same* physical element, i.e., using the same updating function. This can only be achieved through strongly nonlinear (thus typically chaotic) characteristics.

In this brief we focus on *sequential* computing employing a chaotic element with one variable x . This can be extended to *parallel* computing by various schemes. One approach is basically the same as many traditional parallel computers—execute many “processors,” in this case chaotic elements, at the same time. By simultaneously executing L chaotic elements, parallelism of degree L can be achieved. Another approach is to utilize the high dimensionality, or the number of variables, associated with some chaotic systems. For example, three variables are associated with the far-infrared ammonia laser described in the previous section. The normalized amplitude of electric field, the atomic polarization, and the normalized inversion can be assigned as x_1 , x_2 and x_3 , respectively. This specific system can implement parallelism of degree 3. In general, a system with n variables can implement parallelism of up to degree n . There are systems as high as $n = 120$; for example, certain neuronal cells can be realistically described by 120 variables [28], [29].

Furthermore, by combining the above two schemes in a hybrid approach with L n -dimensional elements, we can achieve parallelism of degree nL [17], [18].

The technique described here is universal in that it allows construction of a general-purpose computer for a wide spectrum of applications, rather than a machine for a narrowly specialized problem. This architecture may be particularly fit for simulation of chaotic phenomena such as weather forecasting or biomedical problems such as the analysis of the brain and heart. Harnessing some of the abundant chaotic phenomena in engineered and natural systems for the development of a simple, fast and cost-effective chaotic computer will be an exciting endeavor.

ACKNOWLEDGMENT

The authors would like to thank Dr. D. W. M. Marr of the Colorado School of Mines for his valuable comments.

REFERENCES

- [1] R. Feynman, “Quantum mechanical computers,” *Found. Phys.*, vol. 16, no. 6, pp. 507–531, 1986.
- [2] E. Rieffel and W. Polak, “An introduction to quantum computing for nonphysicists,” *ACM Comput Surv.*, vol. 32, no. 3, pp. 300–335, 2000.
- [3] L. M. Adleman, “Molecular computation of solutions to combinatorial problems,” *Science*, vol. 266, pp. 1021–1024, 1994.
- [4] S. A. Kurtz, S. R. Mahaney, J. S. Royer, and J. Simon, “Biological computing,” in *Complexity Theory Retrospective II*, L. A. Hemaspaandra and A. L. Selman, Eds. New York: Springer-Verlag, 1997, pp. 179–195.
- [5] C. P. Collier, E. W. Wong, M. Belohradsky, F. M. Raymo, J. F. Stodart, P. J. Kuekes, R. S. Williams, and J. R. Heath, “Electronically configurable molecular-based logic gates,” *Science*, vol. 285, pp. 391–394, 1999.
- [6] H. C. Manoharan, C. P. Lutz, and D. W. Eigler, “Quantum mirages formed by coherent projection of electronic structure,” *Nature*, vol. 403, pp. 512–515, 2000.
- [7] A. Bachtold, P. Hadley, T. Nakanishi, and C. Dekker, “Logic circuits with carbon nanotube transistors,” *Science*, vol. 294, pp. 1317–1320, 2001.
- [8] S. Sinha and W. L. Ditto, “Dynamics based computation,” *Phys. Rev. Lett.*, vol. 81, pp. 2156–2159, 1998.
- [9] —, “Computing with distributed chaos,” *Phys. Rev. E*, vol. 60, pp. 363–377, 1999.
- [10] T. Munakata, *Fundamentals of the New Artificial Intelligence: Beyond Traditional Paradigms*. New York: Springer-Verlag, 1998, ch. 7.
- [11] W. L. Ditto and T. Munakata, “Principles and applications of chaotic systems,” *Comm. ACM*, vol. 38, no. 11, pp. 96–102, 1995.
- [12] T. Shinbrot, C. Grebogi, E. Ott, and J. Yorke, “Using small perturbations to control chaos,” *Nature*, vol. 363, pp. 411–417, 1993.
- [13] L. M. Pecora and T. L. Carroll, “Driving systems with chaotic signals,” *Phys. Rev. A*, vol. 44, pp. 2374–2383, 1991.
- [14] W. L. Ditto and L. M. Pecora, “Mastering chaos,” *Sci. Amer.*, vol. 269, no. 2, pp. 78–84, 1993.
- [15] S. Hayes, C. Grebogi, E. Ott, and A. Mark, “Experimental control of chaos for communication,” *Phys. Rev. Lett.*, vol. 73, pp. 1781–1784, 1994.
- [16] G. D. vanWiggeren and R. Roy, “Communications with chaotic lasers,” *Science*, vol. 279, pp. 1198–1200, 1998.
- [17] S. Sinha, T. Munakata, and W. L. Ditto, “Flexible parallel implementation of logic gates using chaotic elements,” *Phys. Rev. E*, vol. 65, no. 036216, pp. 1–9, 2002.
- [18] —, “Parallel computing with extended dynamical systems,” *Phys. Rev. E*, vol. 65, no. 036214, pp. 1–7, 2002.
- [19] T. C. Bartee, *Computer Architecture and Logic Design*. New York: McGraw-Hill, 1991.
- [20] M. M. Mano, *Computer System Architecture*, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1993.
- [21] R. N. Madan, Ed., “Chua’s circuit: A paradigm for chaos,” in *World Scientific Series on Nonlinear Science*. ser. B. Singapore: World Scientific, 1993, vol. 1.
- [22] C. O. Weiss, R. Vilaseca, N. B. Abraham, R. Corbalan, G. J. de Valcarcel, J. Pujol, U. Hubner, and D. Y. Tang, “Models, predictions, and experimental measurements of far-infrared NH₃-laser dynamics and comparisons with the Lorenz–Hanken model,” *Appl. Phys. B*, vol. 61, pp. 223–242, 1995.
- [23] U. Hubner, N. B. Abraham, and C. O. Weiss, “Dimensions and entropies of chaotic intensity pulsations in a single-mode far-infrared NH₃ laser,” *Phys. Rev. A*, vol. 40, no. 11, pp. 6354–6365, 1989.
- [24] I. Fischer, G. H. M. van Tartwijk, A. M. Levine, W. Elsasser, E. Gubel, and D. Lenstra, “Fast pulsing and chaotic itinerancy with a drift in the coherence collapse of semiconductor lasers,” *Phys. Rev. Lett.*, vol. 76, no. 2, pp. 220–223, 1996.
- [25] Q. L. Williams, J. Garcia-Ojalvo, and R. Roy, “Fast intracavity polarization dynamics of an erbium-doped fiber ring laser: Inclusion of stochastic effects,” *Phys. Rev. A*, vol. 55, no. 3, pp. 2376–2386, 1997.
- [26] J. P. Hayes, *Computer Architecture and Organization*, 3rd ed. Boston, MA: McGraw-Hill, 1998.
- [27] A. Agarwal, “Raw computation,” *Sci. Amer.*, vol. 281, no. 2, pp. 60–63, 1999.
- [28] B. J. West, Ed., *Patterns, Information and Chaos in Neuronal Systems*. Singapore: World Scientific, 1993.
- [29] R. Traub and R. Miles, *Neuronal Networks of the Hippocampus*. New York: Cambridge Univ. Press, 1991.