Computing with distributed chaos

Sudeshna Sinha
The Institute of Mathematical Sciences, CIT Campus, Madras 600 113, India

William L. Ditto
School of Physics, Georgia Institute of Technology, Atlanta, Georgia 30332

(Received 19 November 1998)

We describe and discuss in detail some recent results by Sinha and Ditto [Phys. Rev. Lett. **81**, 2156 (1998)] demonstrating the capacity of a lattice of threshold coupled chaotic maps to perform computations. Such systems are shown to emulate logic gates, encode numbers, and perform specific arithmetic operations, such as addition and multiplication, as well as yield more specialized operations such as the calculation of the least common multiplier of a sequence of numbers. Furthermore, we extend the scheme to multidimensional continuous time dynamics, in particular to a system relevant to chaotic lasers. [S1063-651X(99)05306-4]

PACS number(s): 05.45.−a, 89.70.+c

I. INTRODUCTION

A recurring theme of research into chaotic systems over the past 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. Recent successful implementations of this concept have included the exploitation of chaotic behavior for control [1], synchronization [2], encoding information [3], and communications [4]. It is the purpose of this paper to expand this list to include computation, thereby describing a different direction in harnessing chaos.

In this article we describe in detail some recent results by Sinha and Ditto [5] demonstrating the capacity of nonlinear lattices to do computations through the emergent collective properties or the emergent responses of distributed chaos. Here we utilize the complex dynamics of the individual units, as well as their interactive couplings and adaptive processes (implemented in particular as a threshold mechanism) to do computations.

The possibility of universal computing can be demonstrated in different (complementary) ways.

(i) First, in principle, it is shown that a universal Turing machine (UTM) can be simulated by the system [6,7].

(ii) The second, more concrete, approach is to show that the system can emulate a device with which a UTM can be constructed. In particular, since any logic gate can be obtained by adequate connection of NOR gates (i.e., any boolean circuit can be built using a NOR gate), one can try to show that the responses of the dynamical system emulate a NOR gate [8].

(iii) Finally, one can do some prototypical arithmetic operations, such as the particular example of the addition operation, to demonstrate the computational ability of the dynamical system.

In this article we will use the last two approaches to demonstrate ''computing'' via emergent dynamics. First, we describe below our dynamical systems candidate for the ''hardware.''

II. THRESHOLD COUPLED CHAOTIC LATTICES

Now we describe the rich spectrum of phenomena (ranging from spatiotemporal fixed points to exact cycles of all orders) arising from a class of dynamical systems incorporating threshold coupling on a lattice of chaotic elements [9–11]. Here time is discrete, labeled by τ, space is discrete, labeled by i, $i=1, N$, where N is system size, and the state variable $x_\tau(i)$ (which in physical systems could be quantities such as energy, voltage, velocity, pressure, or concentration) is continuous. Each individual site, indexed by their spatial location i in the lattice, evolves under a suitable nonlinear map $f(x)$. For instance, the local map $f(x)$ can be chosen to be the logistic map, which has widespread relevance as a prototype of low-dimensional chaos:

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

$x\in[0,1]$, with the nonlinearity parameter a chosen in the chaotic regime. Specifically $a=4.0$ throughout. Another possible choice in the local dynamics is the circle map, which is relevant in systems involving nonlinear oscillatory behavior [11]. Further, the local dynamics can also be a suitable section or a stroboscopic sampling of a continuous time series arising from coupled ordinary differential equations (ODEs).

Now on this nonlinear lattice a *self-regulatory threshold dynamics* is incorporated. The adaptive mechanism is triggered when a site in the lattice exceeds the critical value x_*, i.e., when a certain site $x_\tau(i)>x_*$. The supercritical site then relaxes (or ''topples'') by transporting its excess $\Delta=x_\tau(i)-x_*$ to its neighbors. So this adaptive connecting mechanism ''opens'' whenever an element exceeds the (prescribed) threshold.

For the specific case of *unidirectional* transport in one-dimensional arrays we have the following scenario: After triggering a response the signal (excess of threshold) is transferred to one neighbor to the right (or left):

$$x_\tau(i)\to x_*,$$

$$x_\tau(i+1)\to x_\tau(i+1)+\Delta. \tag{1}$$

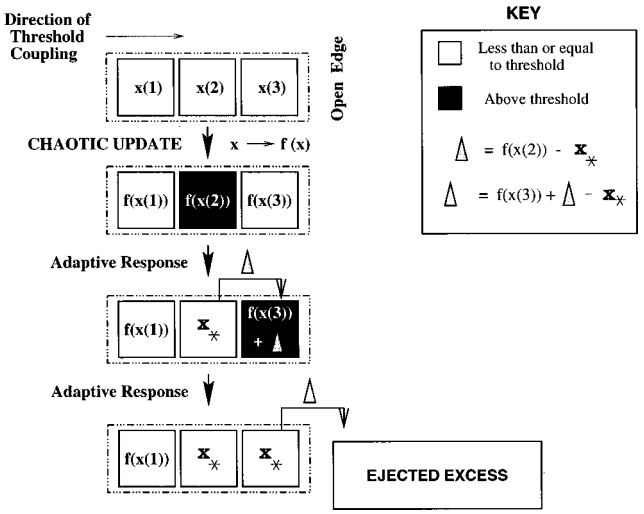

FIG. 1. Flow chart of a lattice of threshold coupled chaotic elements over one dynamical update, i.e., over one chaotic update and the subsequent adaptive response. Here the lattice size is equal to 3. After two relaxation steps (''avalanching'') all sites are undercritical.

The relaxation activity continues until all $x \leqslant x_*$, after which the next chaotic update of the on-site maps take place. That is, the chaotic evolution is slower than the adaptive response, allowing the system to relax completely before each chaotic update (see Fig. 1 for a schematic diagram of the dynamics).

The dynamics above then induces a unidirectional nonlinear transport down the array by initiating a domino effect (reminiscent of the ''avalanches'' arising in self-organized sandpiles [12]). The boundary is open so that the ''excess'' may be conducted out of the system. This kind of threshold mechanism imposed on local chaos makes the above scenario especially relevant for synapses of nerve tissue (note that individual neurons display complex chaotic behavior and have step-function-like reponses to stimuli).

Note that each synchronous iteration of the local maps of the elements in the lattice represents a single click of the dynamical clock. Thus our basic unit of time, which will henceforth be called a *dynamical (or chaotic) update*, consists of one forward iteration of the chaotic local maps of the system followed by relaxation of all lattice sites to their final (fully relaxed) state.

Now the nature of the threshold coupling is such that it actually works as a ''control'' mechanism (quite unlike, say, the usual diffusive coupling). The system is naturally always being ''reset,'' so to speak, by virtue of the adaptive self-regulation. The value of the threshold governs the dynamics of the array, showing the presence of many ''phases'' in x_* space [10]. As the threshold x_* is tuned the state of the lattice (as well as the emitted excess from the open boundary) evolves in cycles of varying orders. When there is no significant external random noise these cycles are exact. So this system, by variation of a *single* parameter (the threshold value), extracts a wide repertoire of dynamics from distributed chaos. Further, all threshold values yielding cycles of a desired order can be obtained *rigorously* through exact analysis [10].

We describe below some details of the dynamical phases. We will use the knowledge of these phases to ''program''

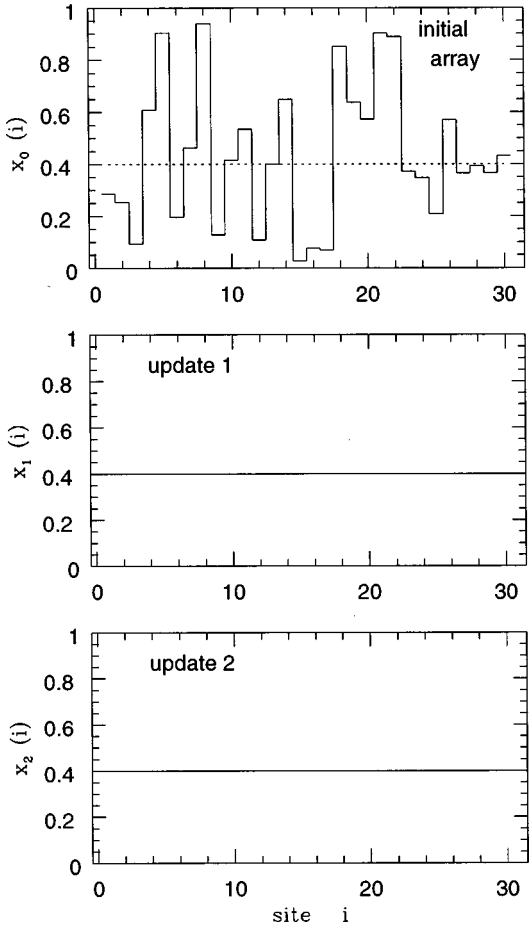

FIG. 2. Spatial profile of a lattice of 30 threshold coupled chaotic elements over three dynamical updates, starting from a random initial configuration. At the end of one update (i.e., after one chaotic iteration of the local maps and the subsequent adaptive response) the random lattice gains a spatially uniform profile [all $x(i)=x_*$]. The excess emitted in the first update is $15.7302\times\delta$. The emitted excess thereafter settles down to a constant value of $30\times\delta$. The value of threshold here is $x_*=0.4$ and the unit of excess emission $\delta=4x_*(1-x_*)-x_*=0.56$.

our system to do what we require.

As noted before, the most significant parameter in the system is the critical x_* [$x_* \in (0,1)$] and by tuning x_* one obtains the following phases. The first phase is the fixed point region that occurs when $x_*<0.75$. In this x_* region, $f(x_*)=4x_*(1-x_*)>x_*$ and therefore the element is always adapted back to $x=x_*$, with the duration of the ''avalanches'' being equal to N [10]. In this parameter regime then, the adaptive mechanism suppresses the underlying chaos in the lattice and yields effective spatiotemporal invariance. Further, the elements emit one unit of excess per dynamical update (see Fig. 2 for an illustrative example).

When $x_*=0.75$ we still have a coherent state with all $x(i)=x_*$, but now the avalanches are of size zero as there are never any active sites in the lattice, $x=0.75$ being a fixed point of the map $f(x)$. So when all the maps are at their fixed points the dynamics is trivial and there is no avalanching or cascade of excess.

When $0.75<x_*<1.0$ the temporal evolution of the lattice is attracted to a cycle whose periodicity depends on x_* [10]. Further, excess is emitted from the open edge of the lattice at

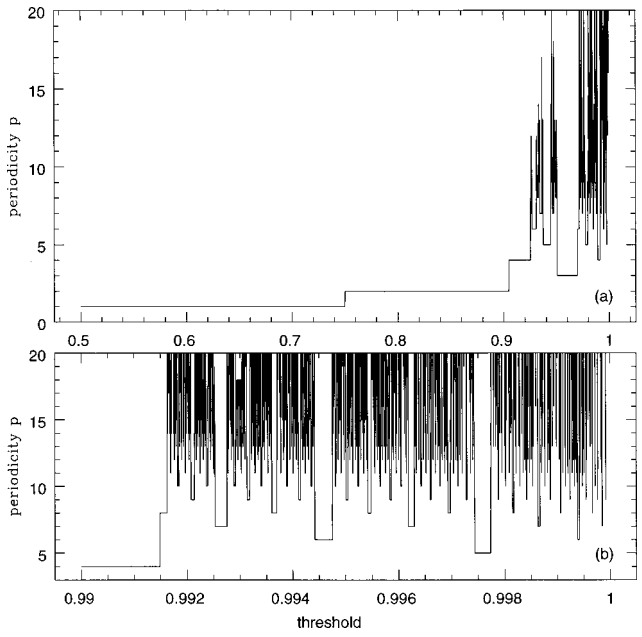

FIG. 3. Order p of the cycle obtained at various values of the threshold parameter x_*, for the case of unidirectional transport on a lattice of logistic maps, depicted for the ranges (a) $x_* \in [0.5,1.0]$ and (b) $x_* \in [0.995,0.9999]$ (p up to 20 is shown in both figures).

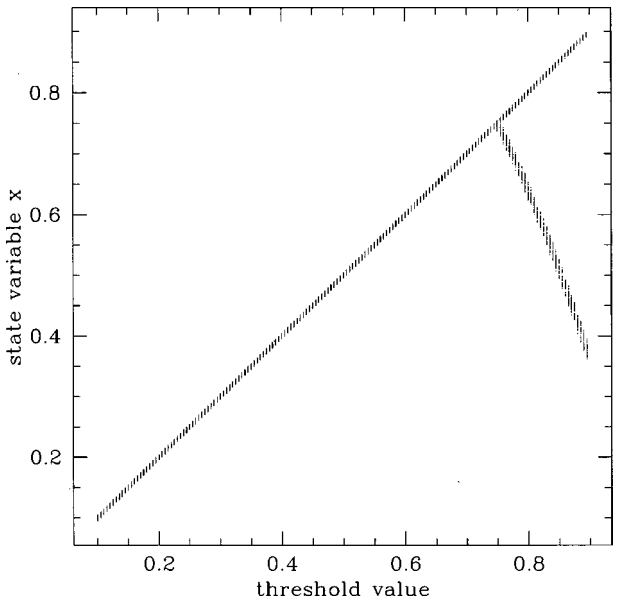

FIG. 4. State of the chaotic element under threshold mechanism in the range $0.0 < x_* \leqslant 0.9$ under the influence of uniform random noise in the state variable x as well as in the value of threshold x_*. The strength of the additive noise here was 0.01 for both the state variable and threshold setting. Its clear that the fixed point and the two-cycle region remain largely unchanged, with the attractor being broadened only slightly.

the same periodicity. For example, by tuning x_* one obtains the following dynamical phases: for $0.75 < x_* \leqslant 0.905...$ we get cycles of order 2, for $0.905... < x_* < 0.925$ we get order 4, for $x_* \sim 0.93$ we get order 6, for $x_* \sim 0.935$ we get order 7, for $x_* \sim 0.95$ we get order 10, for $x_* \sim 0.99$ we get order 4, and so forth. From Fig. 3 (which indicates the order of the cycle supported at various values of x_*) it is clearly evident that the system yields a rich repertoire of cyclic patterns. Now we will use the temporal characteristics of these cyclic states, emerging from chaotic dynamics through adaptive coupling, to do computations.

The transience times for settling onto some attractor are very short in this dynamics. For instance, Fig. 2 shows a random chain of 30 elements taking *one* dynamical update (i.e., one chaotic iteration followed by threshold response) to reach the attractor. This is indeed very typical. The evolution to the attractor here is so quick because a sizable part of state space can be ''grabbed'' instantly. Further, the domino effect (''snowballing'') ensures that elements down the chain will almost certainly ''trigger'' as well. Once triggered the element is immediately trapped into the desired cycle (and small noise does not nudge the trajectory away from this cycle, for a large range of threshold).

Figure 4 shows the above dynamics in the fixed point and the two-cycle region under the influence of noise. Here we consider random fluctuations in the state variable x, as well as in the threshold setting x_*. Clearly the only change discernible in the fixed point and two-cycle region is the slight broadening of the attractor. Note that while the attractor here is somewhat blurred, the period of excess emission still is exactly 1 for thresholds $x_* < 0.75$ and 2 for thresholds $0.75 < x_* \leqslant 0.905...$. So evidently the basic dynamical behavior in this region of threshold parameter space is robust with respect to noise. In the sections below we will use this two-cycle region for logic gate construction and the fixed point region for encoding numbers and doing arithmetic operations.

Note certain properties that underscore the vital significance of chaos in the emergent dynamics here. (a) If the same threshold dynamics was imposed on a random lattice, we would not recover any of the above-mentioned phases. For these phases to occur we crucially require a *deterministic dynamics*. (b) The *ergodic* properties of chaotic dynamics guarantees that the system always falls into the desired cycle and does not get trapped in any corner of phase space. (c) Generic locally chaotic maps (ranging from circle maps to tent maps) will yield an infinite number of periodicities under variation of the threshold [10].

Thus the chaotic element (our potential processing unit) under suitable cut-off on the state variable yields excess emission at different periodicities. In a way then, one is obtaining an ''output'' of varying periods from a chaotic ''input'' and only a chaotic input can yield all possible orders via one threshold variable. Now we will do computations using these varied patterns generated by threshold-coupled chaos [13].

III. CONSTRUCTION OF GATES

First we will demonstrate that our dynamical system can easily emulate logic gates. We will define our inputs suitably through input states and the output via easily obtainable, collective properties. For instance, for inputs, one can interpret the state of an element as (i) $x \sim x_*$ is 1 and (ii) $x < x_*$ is 0. An interesting dynamical response that can be used to characterize the output is the *excess transported out of the open boundary of the array as a result of the avalanching process*

TABLE I. NOR gate. The two inputs are I_1 and I_2 and the output is O.

I_1	I_2	O
0	0	1
0	1	0
1	0	0
1	1	0

initiated by adaptive relaxation. Thus one can imagine a readout (or *lead*) at one end of the array specified by the inputs, registering the ''excess signal'' that represents the output.

Now we operate in the x_* regime where the chaotic elements emit excess as two-cycles under threshold coupling. Here a two-element unit can have two possible states (note that we always consider attractor states, not transients).

(i) The coherent configuration. This occurs in the range $0.75<x_*<0.905$ and emits excess following threshold coupling, from the open edge, in the cyclic sequence 0202...[in units of $\Delta_1=f^2(x_*)-x_*$].

(ii) The out-of-phase configuration. This occurs in the range $0.835...<x_*<0.905$ and ejects excess in the sequence 0101... [in units of $\Delta_2=f(f(x_*)+\Delta_1)-x_*$].

Exploiting the phenomena described above, a particular realization of a NOR gate is achieved as follows. We work in the parameter window around $x_*\sim 0.84$, where (i) coherent and out-of-phase configurations are coexisting attractor states and (ii) the units of emitted excess for the two states are very different, with $\Delta_2 \ll \Delta_1$. This clearly distinguishes the adaptive response of the coherent configurations and the out-of-phase configurations.

Now the two inputs I_1 and I_2 of the logic gate imply two elements in specified states, whose collective response after a chaotic update (namely, the excess signal from the open boundary of the two-element chain) should emulate the output given in the NOR look-up table (Table I).

(i) If the inputs are $I_1=0$ and $I_2=0$, the states of the two elements comprising the gate are both $x<x_*$. The response of this coherent array (00) after a chaotic update leads to an emission of $2\Delta_1$ from the open edge (i.e., from element 2).

(ii) If the inputs are $I_1=0$ and $I_2=1$, the gate is comprised of the out-of-phase array (01), whose response after a chaotic update is to eject a total excess of 0 from the open edge.

(iii) If the inputs are $I_1=1$ and $I_2=0$, the gate is comprised of the out-of-phase array (10), whose response after a chaotic update is to eject a total excess of $1\Delta_2\sim 0$ from the open edge.

(iv) If the inputs are $I_1=1$ and $I_2=1$, the gate is comprised of the coherent array (11), whose response after a chaotic update is to eject a total excess of 0 from the open edge.

Defining the output from the collective response of the chain as 1 if ejected amount is much greater than 0 and 0 if ejected amount is approximately 0, it is clear that the *input-to-output association corresponds to that of a NOR gate*. See Fig. 5 for a schematic of this NOR gate construction.

Any Boolean circuit can be constructed by a suitable connection (i.e., coupling) of this basic two-element unit, whose responses emulate a NOR gate.

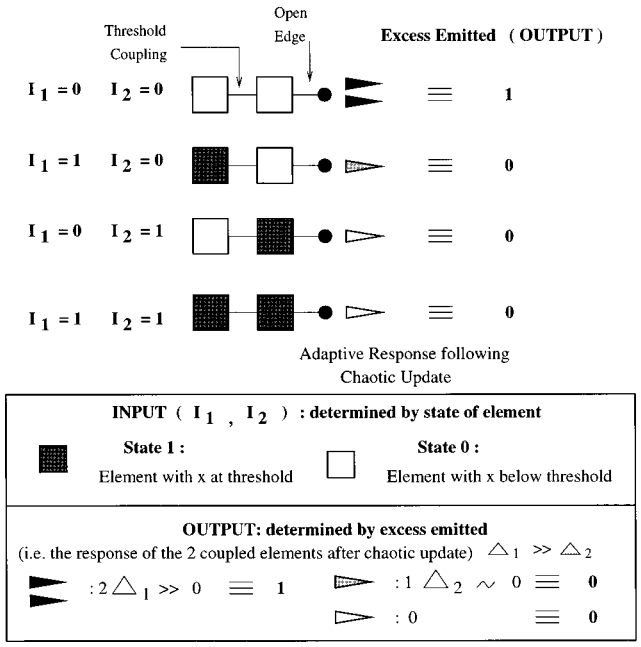

FIG. 5. Schematic diagram of the inputs and (dynamical) output emulating a NOR logic gate (cf. Table I).

Since the above response pattern is realized in a reasonably wide parameter window it is robust to small noise in the threshold setting, thus yielding a *stable* and *easily implementable* NOR gate. Further, the logic gates are robust to noise in state variable x and and one can employ a fuzzy definition for the input and output and still obtain the necessary input-output association (see Fig. 4). Note that one requires sufficiently strong nonlinearities in the local map in order to obtain the emission pattern necessary for the construction of gates, for instance, only $f(x)=ax(1-x)$ with $a>3$ can yield the required response.

Now there are some nontrivial issues in Boolean circuit implementation, foremost among which is the issue of wiring of gates, and the consequent spatial arrangement of processing units. In order to construct logic gates that are amenable to easy concatenation we now present an alternate prototype. Here each gate is realized by a *single* chaotic element and all *inputs and outputs are equivalently defined* such that the output of a gate can easily ''flow'' into the input. So the spatial arrangement of processing units comprising a gate array is quite simply realized in this prototype. We describe details below.

Alternate NOR gate: Using a single chaotic element under threshold mechanism

A single chaotic element can act as a gate as follows. Let the initial state of the gate element be equal to threshold x_*. Now the *inputs are stimulations* (kicks) to the state of the element. Thus inputs I_1 and I_2 make the state of the gate element

$$x_{\text{gate}}=x_*+I_1+I_2.$$

Also note that by this definition of input we naturally have a situation symmetric in I_1 and I_2.

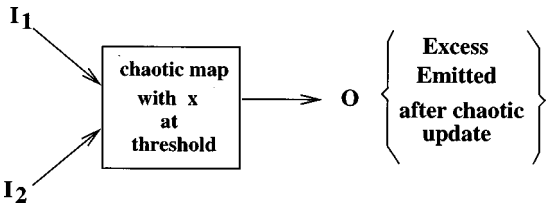

FIG. 6. Schematic diagram of the inputs and output of a single chaotic element emulating a logic gate.

The output, as before, is the excess emitted after a chaotic update due to the threshold mechanism: $O \equiv f(x_{\text{gate}}) - x_*$ if $f(x_{\text{gate}})$ is in excess of threshold x_* and $O \equiv 0$ if not (see Fig. 6 for a schematic).

Now output O can easily serve as input for a connected gate element, as what is emitted can directly add on to the state of the other gate element. So *wiring gates together is simply the usual threshold coupling of chaotic elements* [Eq. (1)] with the *excess emission from the open edge giving the output of the concatenation of gates*.

Now for a NOR gate we require the following characteristics to be true: (i) When $I_1 = 0$ and $I_2 = 0$, $O = 1$ unit. (ii) When $I_1 = 0$ and $I_2 = 1$ unit, $O = 0$. (iii) When $I_1 = 1$ unit and $I_2 = 0$, $O = 0$. (iv) When $I_1 = 1$ unit and $I_2 = 1$ unit, $O = 0$. This is realized in the range $x_* = 0.696 - 0.75$. We discuss details below.

When both I_1 and I_2 are 0, we have

$$x_{\text{gate}} = x_* + 0 + 0$$

and

$$f(x_{\text{gate}}) > x_* .$$

So the element emits one unit of excess, where one unit is equal to $4x_*(1-x_*) - x_*$. When either I_1 or I_2 is 1, we have

$$x_{\text{gate}} = x_* + 1 + 0 = x_* + 0 + 1$$

and

$$f(x_{\text{gate}}) < x_* .$$

So the element emits zero units of excess. When both I_1 and I_2 are 1, we have

$$x_{\text{gate}} = x_* + 1 + 1$$

and

$$f(x_{\text{gate}}) < x_* .$$

So the element emits zero units of excess. This clearly follows the NOR input-to-output association pattern.

We now give a specific example with threshold of gate element $x_* = 0.70$. Now one unit (i.e., the excess emitted by a chaotic element under threshold x_* after a chaotic update) is $4x_*(1-x_*) - x_* = 0.14$.

(i) For $I_1 = 0$ and $I_2 = 0$, $x_{\text{gate}} = x_* + 0 + 0 = 0.7$ and $f(x_{\text{gate}}) = f(0.7) = 0.84 > x_*$ and so output $O = 1$.

(ii) For $I_1 = 1$, $I_2 = 0$ and $I_1 = 0$, $I_2 = 1$, $x_{\text{gate}} = x_* + 1 + 0 = x_* + 0 + 1 = 0.84$ and $f(x_{\text{gate}}) = f(0.84) = 0.5376 < x_*$ and so output $O = 0$.

(iii) For $I_1 = 1$ and $I_2 = 1$, $x_{\text{gate}} = x_* + 1 + 1 = 0.98$ and $f(x_{\text{gate}}) = f(0.98) = 0.0784 < x_*$ and so output $O = 0$.

One can thus obtain a clearly defined NOR gate with a single chaotic element under the range of threshold indicated above. Now we have the added advantage that the *input and output have equivalent definitions* (i.e., one unit is the same quantity for input and output). Further, the output of one gate element can now easily couple to another gate element as input, so that gates can be ''wired'' directly. Also this response pattern is robust as it can be obtained in a reasonably wide range of threshold.

IV. ARITHMETIC OPERATIONS

Now we will adopt an alternate approach to the computing question. We will try to do some specific arithmetic operations through our emergent dynamics.

Our potential *processors* are a very large number of uncoupled elements evolving under their natural chaotic dynamics, as in the chaotic lattice described above. When they are not ''computing'' the elements are uncoupled (i.e., we can think of the default threshold to be $x_* = 1.0$, which leads to no interelement transfer, as $x \in [0,1]$ for the logistic map).

Specification of the input/operation consists of providing threshold parameters for some elements for threshold coupling ($x_* < 1$), which leads to an avalanche of emitted excess providing communication of information among these elements. Tapping the emergent collective excess from a specified open edge (readout) yields the answer. After each operation the processors is ready for the next instruction (which is another threshold parameter for the element). Any computing (recall von Neumann) is simply a sequence of such instructions.

So our *hardware* consists of lattices of logistic maps capable of interacting through threshold coupling. Our *programming* consists of fixing the threshold of the response of the lattice elements such that it performs a desired operation to yield the answer.

A. Encoding scheme 1

First, we describe how numbers are *encoded through their excess emission*. Excess emission is a direct function of the threshold values. In the threshold range $0 < x_* < 0.75$, a chaotic element under adaptive threshold response emits excess after each chaotic update in order to relax back to x_*. The amount of excess emitted per dynamical update is an unimodal nonlinear function of the threshold, over the range $0 < x_* < 0.75$, going from 0 at $x_* = 0$ to a maximum value $E_{\max} = 9/16$ at $x_* = \frac{3}{8}$ and then back to 0 again at $x_* = \frac{3}{4}$. In our encoding scheme the amount of excess emission (in specified units) directly gives the value of the integer. We define the unit of excess emission to be $\delta = E_{\max}/N$, where N is the largest integer we wish to encode. Then an integer m is encoded by an excess emission of $m\delta$. In order to encode from integer 0 to N, the necessary capacity of resolution of emitted excess must be $E_{\max}/N = \delta$. Clearly, greater precision in measuring the excess and threshold setting allows larger numbers to be encoded.

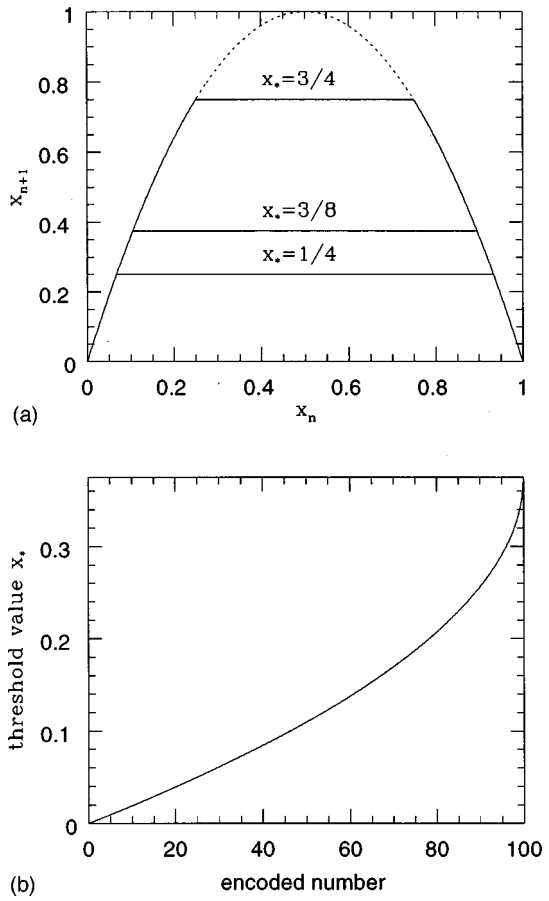

FIG. 7. (a) Dynamics of a single chaotic element updating under the logistic map $f(x)=4x(1-x)$ followed by adaptive threshold response. Here three threshold values x_* are displayed. The difference between the solid and dotted lines gives the amount of excess emitted in the dynamical update. (b) Look-up graph of encoded number vs threshold value x_*. The encoded number is given by the emitted excess Δ (which is a function of x_*) through the following relation: the encoded number is equal to Δ/δ, where $\Delta=f(x_*)-x_*$ and $\delta=E_{\max}/N$, with N being the largest number encoded ($N=100$ here).

Since the map is deterministic, one can *determine exactly* the threshold that yields a given excess (where the excess varies from 0 to N units). So we can obtain a look-up table associating the value of the threshold with the value of an integer for an arbitrarily large set of integers (see Fig. 7). The same encoding can handle real numbers as well (with precision determined by the excess and threshold setting resolution). So the *same* element can encode an arbitrarily large set of numbers, under varying threshold (with the threshold levels being sent to it as part of the ''software'').

Typically stronger nonlinearities yield a larger range of excess emission. For instance, the parabolic form of the logistic map $f(x)=ax(1-x)$, at $a=4$, has the highest maxima and thus yields the largest difference between the map and the effective truncated map after the adaptive response (shown in Fig. 7). So the range of excess emission $[0,E_{\max}]$ is determined by the above-mentioned difference, with $E_{\max}$ being exactly equal to $(a-1)^2/4a$ for the logistic map. Thus the range ($[0, \frac{9}{6}]$ for $a=4$, $[0, \frac{1}{3}]$ for $a=3$, and $[0, \frac{1}{8}]$ for $a=2$, down to 0 for $a<1$) clearly decreases with decreasing strength of nonlinearity a.

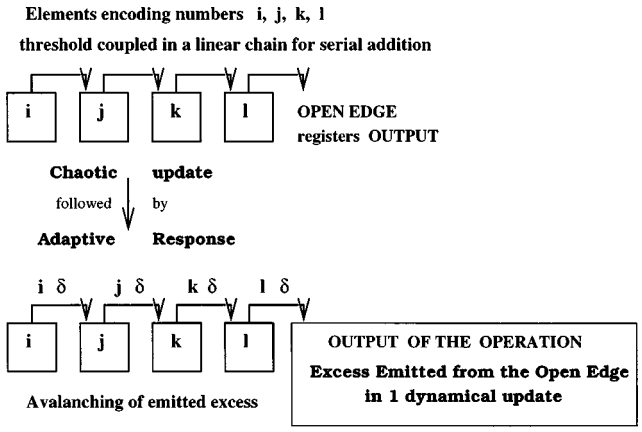

FIG. 8. Threshold coupled chaotic elements emulating an adding machine: Here we are adding four integers *i,j,k,l*, each encoded by an element with threshold fixed such that it emits *i,j,k,l* units of excess, respectively (where the unit of excess emission is δ). These elements encoding the terms are then threshold coupled for the addition operation. The ejected excess from element i $(=i\times\delta)$ drives element j and so on, up to element l, from whose open boundary the collective excess is emitted to the output lead. This emitted excess is exactly the sum $i+j+k+l$ (in units of δ). For serial addition we have a linear chain configuration and the computing time is equal to the number of terms in the sum (which is 4 here).

B. Addition

For the addition operation, we have to *threshold couple the elements encoding the terms in the addition.* That is, the excess emitted from one element drives the next (specifically, the excess ejected from an element is fed to the next element). The collective excess emitted from the open edge of the array [i.e., from the last term (element) of the sum] goes to a lead, which registers the answer. (See Fig. 8 for a schematic diagram.) Since a linear chain of threshold coupled elements, after chaotic update, gives rise to an avalanche that sweeps across the lattice gathering excess from all the elements, the excess emitted from the open edge of the lattice is then the sum total of all the individual excess amounts and can directly be associated with the result of the addition operation.

Now the ''computation time'' can be considered (naturally) to be the time the dynamical system takes to adaptively relax and emit excess from the open edge, which is the ''answer.'' Then one readily seems that the computing time is simply equal to the number of terms in the addition. Further, this operation is *commutative* as the ordering of the terms (elements) does not influence the answer.

Note that the range of parameters under which this encoding/addition works is very broad. Also in this range noise does not significantly degrade the performance of computation.

C. Parallelized operations

Importantly, one can do the addition operation in *parallel* (i.e., synchronously/concurrently) by having a *branching* topology of the lattice. ''Parallelization'' in this prototype relies on the fact that the relaxation takes place *synchronously* for all processing units and is a local phenomenon. Thus, to

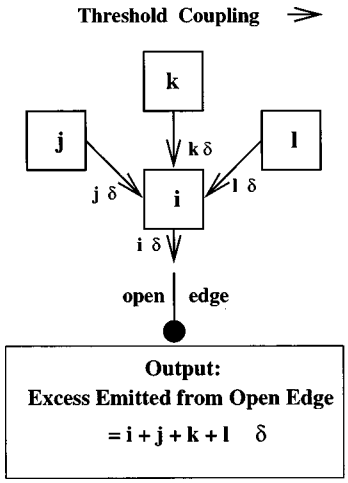

FIG. 9. Schematic diagram of the parallelized operation employing a branching lattice, where each branch is an element encoding a term in the addition. The excess emitted from the open edge yields the resulting sum. The adaptive relaxation process (equivalent to the computing time) occurs in only two avalanching steps.

add several numbers, one can employ a branched lattice where each branch is an element encoding a term in the sum. The duration of the ''avalanche'' of excess to the open end is equal to the length of the longest branch in the network; since the computing time is the time taken by the units to relax and emit excess to the readout, this time can be drastically shortened by appropriate networking. In serial wiring of units the length of the longest (only) branch is equal to the number of terms added and so the computing time is proportional to the number of terms in the addition, as we saw earlier. On the other hand, parallel circuits can yield relaxation in much shorter times, utilizing the simultaneous avalanching in the different branches. (See the schematic diagram, Fig. 9, and an example in Fig. 10, where the addition operation on 15 terms is parallelized to yield a computing time of 4, instead of 15 as in serial addition.)

Thus branching networks of dynamical elements can serve as a massively parallel machine, with several ''input leads'' flowing concurrently into a processing unit, from whose open edge the resulting net answer is collected. This parallelization potential is derived from the synchronicity of all the dynamical processes and the locality of the threshold and relaxation rule.

D. Multiplication

Multiplication can be performed (as an extension of addition), invoking the same parallel computational approach through branching lattices. For instance, to do $m \times n$ we have a lattice with n branches, each branch being a copy of the element encoding m. The total ejected excess will be the answer $m \times n$.

Alternately, in order to do $m \times n$, we can take the element representing m and *collect the ejected excess over n time steps*, i.e., the quantity accumulated over n units of the local chaos clock. This quantity will be equal to $m \times n$. In this scheme then, one waits for n time steps and then retrieves the result, which is the collective excess. That is, in order to multiply, we are now exploiting the temporal evolution of the adaptive response.

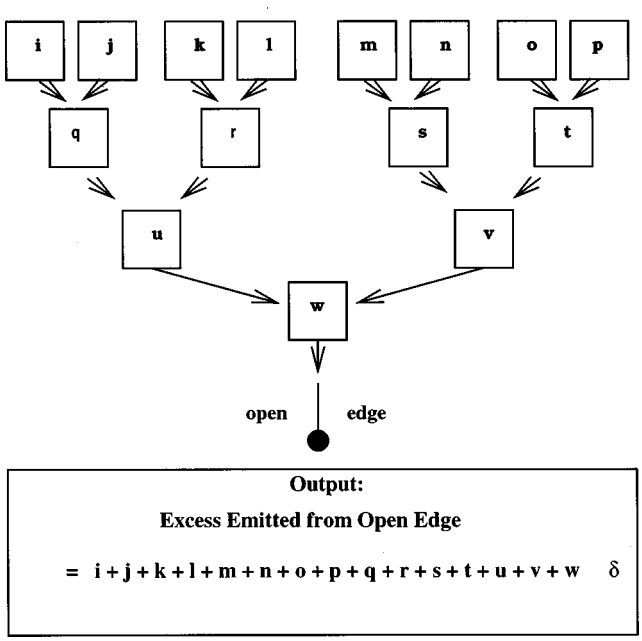

FIG. 10. Schematic diagram of the parallelized addition operation on 15 numbers *i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,* employing a treelike branching lattice. The excess emitted from the open edge yields the resulting sum. The adaptive relaxation process (equivalent to the computing time) occurs in only four avalanching steps (i.e., equal to the longest branch in the network).

The latter method of implementing multiplication will cost more time than the former method (specifically, n times more), while requiring less ''space,'' in the sense of needing far fewer elements or ''processors'' (specifically, n times less). So depending on the resources available, i.e., whether one has many elements or very quick dynamics, one can choose either way of doing the operation [14].

E. Encoding scheme 2

We denote the threshold yielding excess emission at periodicity k as x_*^k. Now in order to encode an N-bit binary number whose representation is $a_N a_{N-1} \cdots a_2 a_1$, we use N chaotic elements, each encoding a bit. If the value of the kth bit is 1 (i.e., $a_k = 1$), its threshold is set at $x_*^{2^{N-k}}$, such that it emits excess periodically with period 2^{N-k}. For instance, if the bit farthest from the decimal point $a_N = 1$ it will be encoded by an element whose period is 1, while if $a_1 = 1$ it will be encoded by an element with periodicity 2^{N-1}. If the value of a bit is 0, the element representing the bit has threshold set at 0, thus emitting no excess.

To obtain the value of the number $a_N \cdots a_1$, we have to threshold couple the N elements representing the bits, with a_N having the open edge to the readout. We evolve this chain over one period of the longest period 2^{N-1}, i.e., over 2^{N-1} dynamical updates. The collective excess emitted will be equal to the value of the number, namely, $\Sigma_{k=1,N} a_k 2^{k-1}$. This encoding scheme exploits chaos as it employs many different periods and only a locally chaotic element can yield all of them under varying threshold. The scheme can be modified easily to encode any other base expansions, such as decimals, as well. (See Fig. 11 for an example of this encoding scheme.)

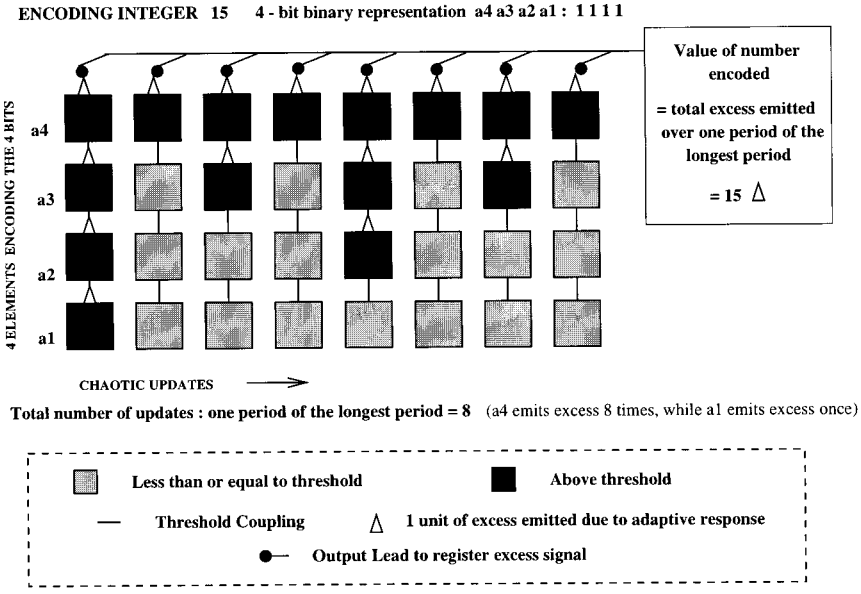

FIG. 11. Encoding an integer by a chain of threshold coupled elements.

Table II shows the temporal pattern of excess emission necessary for encoding a four-2 bit binary number 1111. For four bits, each encoding 1, we have four elements with threshold set such that excess emission occurs periodically every first, second, fourth, and eighth chaotic updates, respectively. The total excess emitted is equal to $8+4+2+1=15$, which gives the value of the number.

As another example, in order to encode, say, the integer 5 (101 in binary) we employ a chain of three elements, each encoding a bit. The first bit has threshold x_*^4, followed by one with $x_*=0$, and the last element (with the open edge) has x_*^1. So over one period of the longest period $2^{3-1}=4$, i.e., over four dynamical updates, the element encoding a_3 (value 1) emits one unit δ of excess 4 times, while the element encoding a_2 (value 0) emits 0 excess and the element encoding a_1 (value 1) emits one δ once. Thus the total emitted excess is equal to $4\times1+0+1\times1=5$ and directly gives the value of the encoded binary number.

Note that one has to take care in chosing the same unit of excess emission for all cycles, i.e., the amount an element with threshold x_*^k emits after k steps should be the same for all k. The threshold values for which a requisite set of cycles emit the same excess can be determined exactly. The number of bits that can be encoded is limited by the resolution of excess emission and threshold setting.

For addition now we can again threshold couple the chains of elements representing the terms in the sum. After evolution over 2^{N-1} chaotic updates, the coupled elements will yield from the open boundary an amount equal to the result of the addition. This operation commutes and any number of terms can be threshold coupled together (i.e., ''added'') in series (linear chain configuration) or parallel (branching chain configuration). The relaxation time (which determines computing speed) for serial addition of m N-bit numbers is less than or equal to $N\times m$, while for parallel addition it is less than or equal to $N+1$. A specific example of the serial and parallelized addition operation is demonstrated in Figs. 12 and 13.

TABLE II. Chaotic updates τ at which excess emission occurs for the elements encoding four binary bits 1 1 1 1, displayed over one period of the longest period equal to $2^{4-1}=8$.

n \ τ	1	2	3	4	5	6	7	8
1	×							
2	×				×			
3	×		×		×		×	
4	×	×	×	×	×	×	×	×

F. Least common multiple

We have shown how the collective response of distributed chaos can emulate logic/arithmetic operations. There is also scope for devising ''dynamical algorithms'' that exploit the

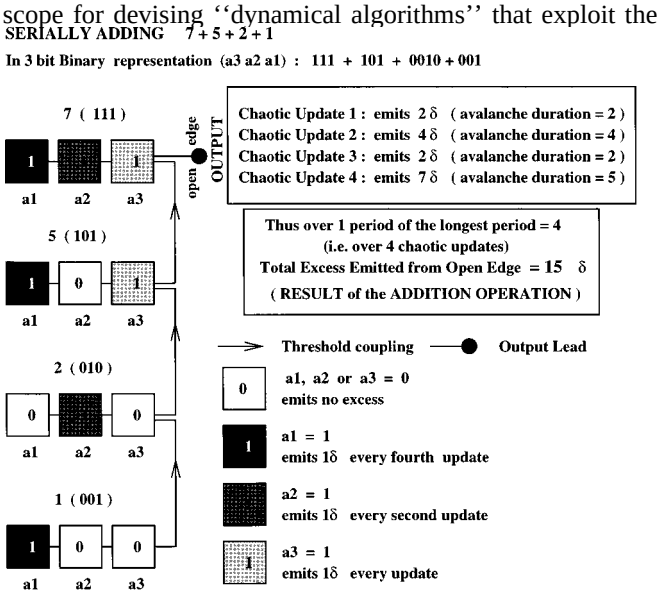

FIG. 12. Serial addition operation of four integers 7, 5, 2, and 1, where the terms of the addition are encoded by a chain of three elements each. These are threshold coupled in a linear configuration. Now after four dynamical updates (since the longest period is equal to $2^{3-1}=4$) the entire lattice emits a total excess of 15 units, which is the result of the operation $7+5+2+1=15$.

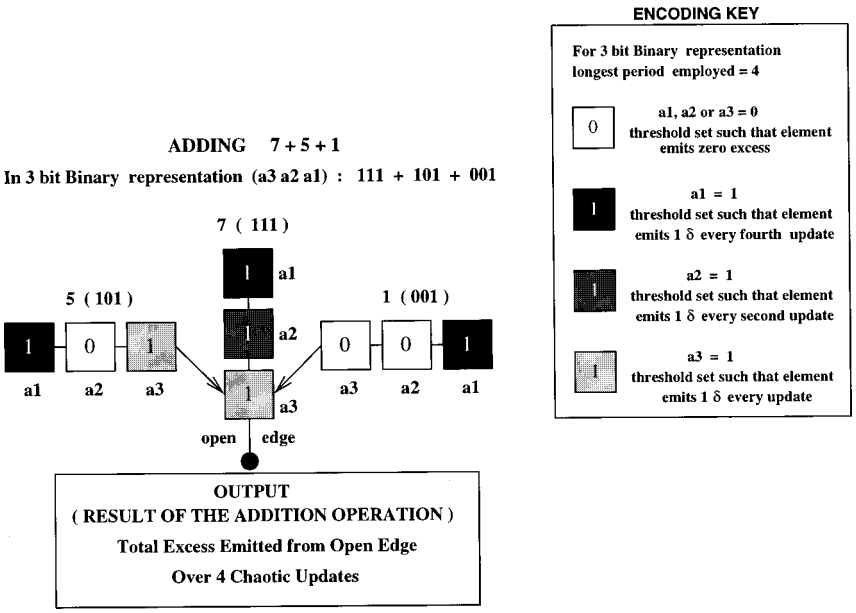

FIG. 13. Parallelized addition operation of three integers 7, 5, and 1, where the terms of the addition are encoded by a chain of three elements each. These are threshold coupled in a branching configuration. Now after four dynamical updates (since the longest period is equal to $2^{3-1}=4$) the entire branching lattice emits a total excess of 13 units, which is the result of the operation $7+5+1=13$. Here the complete relaxation process in each dynamical update requires a maximum of four avalanching steps (i.e., equal to the longest branch in the network).

richness of chaos to compute other numerical quantities. For instance, a dynamical algorithm for finding the least common multiple (LCM) can be realized as follows.

In order to find the LCM of n integers $k_1,k_2,\ldots,k_n$ we use n chaotic elements as ''input.'' These n input elements have their threshold fixed at values such that they emit excess cyclically with periods $k_1,k_2,\ldots k_n$. The periodicity of excess emission thus represents the value of the terms of the LCM.

Now the deterministic chaos allows us to obtain exact generating equations for threshold values supporting a certain periodicity [10]. Thus one can obtain a look-up table relating periodicity k of the excess emission of an element to threshold x_* in order to represent any positive integer.

Now, these input elements are coupled in parallel to one ''master'' element whose threshold is fixed at $x_*<0.75$ and that has the open edge leading to the answer. So the excess ejected from the input elements *synchronously stimulate* the master element, which in turn emits excess from its open boundary with periodicity equal to the *LCM of all the input stimulus periods*. Thus one obtains the required answer, i.e., the LCM of the terms, by simply measuring the period of the master element's response. Note that one can handle many terms in *parallel* here by stimulating the master element synchronously with different periodic impulses. Figure 14 shows a schematic of this calculation.

Alternately, one can have a simpler method for computing the LCM of two numbers by exploiting the diverse responses obtained under varying frequencies of output measurement, that is, with respect to varying intervals of excess emission sampling. In order to compute the LCM of two integers k_1 and k_2, we encode one of them, say k_1, by a chaotic element via the look-up graph described above (i.e., set the threshold such that it emits excess at period equal to k_1). Then one measures the response (i.e., the output) every k_2^{th} step, that is, measures the excess emission periodically with period equal to the second number (i.e., equal to k_2). The resulting excess

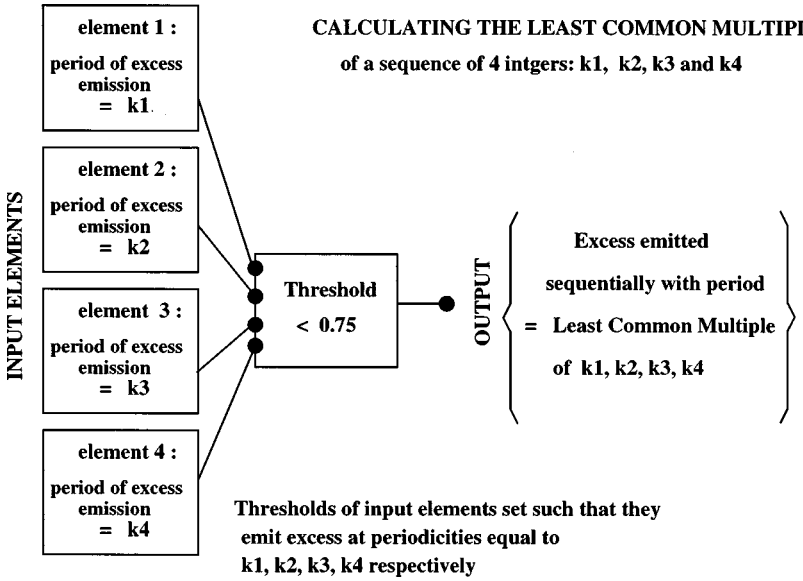

FIG. 14. Schematic diagram of the calculation of the LCM of three integers.

will then be registered every kth step, where k is the LCM of k_1 and k_2.

By this alternate method we can calculate the LCM of only two numbers in one operation. The LCM of n integers can be found by $n-1$ operations involving two numbers each. If done in parallel (in a binary tree structure) the calculation of the LCM of n integers will take of the order of $\log_2 n$ times the computation time for the calculation of the LCM of two numbers.

These simple but intriguing examples demonstrate that dynamic algorithms hold the potential for computing a range of specialized mathematical operations. Thus we begin to see the first glimpse that dynamics can perform computation not just by emulating logic gates or simple arithmetic operations but by performing more sophisticated operations through self-organization rather than composites of simpler operations.

V. APPLICATION TO LASER SYSTEMS

Now we present evidence of continuous time multidimensional systems yielding dynamical characteristics that can be exploited for computations in a manner similar to that outlined above. Consider a collection of threshold coupled chaotic Lorenz systems, where each unit is given by a set of three coupled ODEs

$$\dot{x} = \sigma(y - x),$$

$$\dot{y} = rx - y - xz,$$

$$\dot{z} = xy - bz. \tag{2}$$

We can implement the threshold action on any of the three variables.

It is known that there exists a correspondence between the laser and Lorenz system as follows: The z variable corresponds to the normalized inversion and the x and y variables correspond to normalized amplitudes of the electric field and atomic polarizations, respectively. The three parameters for the corresponding coherently pumped far-infrared ammonia laser system are $\sigma = 2$, $r = 15$, and $b = 0.25$. These parameter values have been obtained by detailed comparison with experiments [15]. Specifically, we choose the parameters of the Lorenz system to be the ones relevant to the IR NH_3 laser and henceforth we will refer to it as a laser system.

We can impose the threshold mechanism on any one of the three variables of the laser system, i.e., one demands that any variable x, y, or z must not exceed a prescribed threshold value x_*. Figures 15–18 show some representative results of this threshold action for a range of threshold values. It is clear that the threshold mechanism yields fixed points (Fig. 15) and limit cycles of varying sizes (Figs. 16 and 17).

Now low threshold values lead to fixed points in phase space, while larger thresholds generate cycles. Specifically, all thresholds $x_* \leq r-1$ imposed on the z variable and $x_* \leq \sqrt{b(r-1)}$ imposed on the x and y variables yield fixed points. Larger thresholds yield limit cycles whose sizes increase with increasing threshold (see Figs. 16 and 17 for examples). When the threshold is very large (close to the bounds of the attractor) the system under threshold mechanism yields broad cycles, like ribbons in phase space.

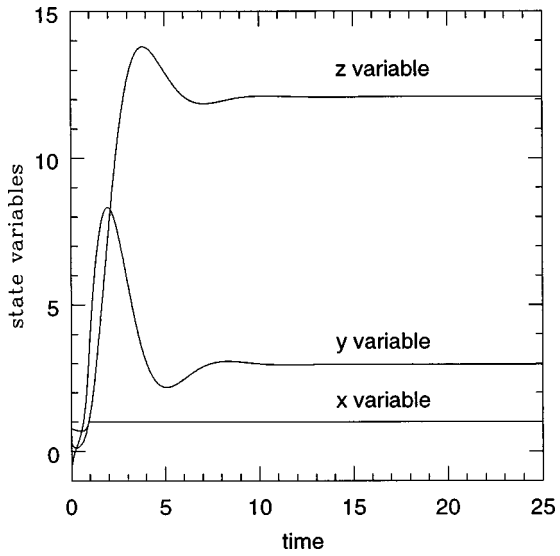

FIG. 15. Chaotic laser [with the parameters $\sigma = 2$, $r = 15$, and $b = 0.25$ in Eq. (2)] with variable x under threshold mechanism, with threshold value $x_* = 1$. The coupled ODE's evolved via fourth-order Runge-Kutta method with step size equal to 0.01. The threshold mechanism is implemented at intervals of 0.01. The three state variables are seen to rapidly evolve to the fixed point.

Note that the above holds for threshold implementation at reasonably short intervals. If the threshold condition is checked infrequently, one obtains fuzzy cycles (like ribbons in phase space) instead of exact cycles. The ''width'' of these broad limit cycles is inversely proportional to the interval at which the threshold mechanism is implemented.

Arithmetic operations with the laser system using encoding scheme 1

We find that threshold coupled laser systems can successfully encode and emulate addition/multiplication, using the

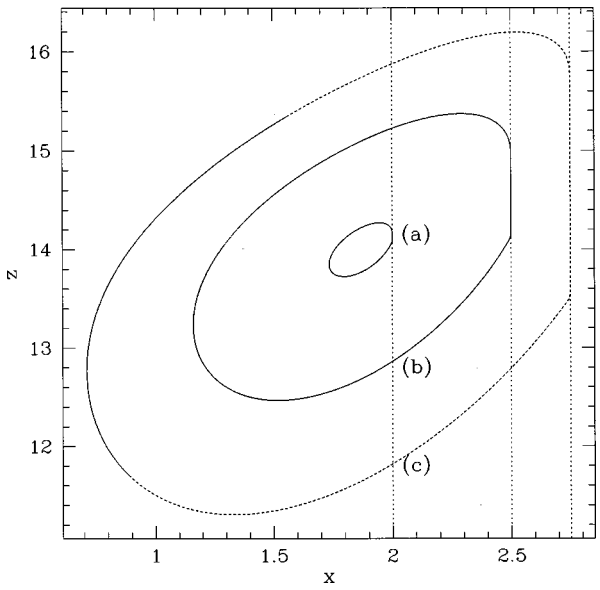

FIG. 16. Chaotic laser (with $\sigma = 2$, $r = 15$, and $b = 0.25$), with variable x under threshold mechanism, with threshold value equal to (a) 2.0, (b) 2.5, and (c) 2.75. The chaotic orbit yields limit cycles of increasing size for these thresholds. The dotted lines indicate the three different values of the threshold.

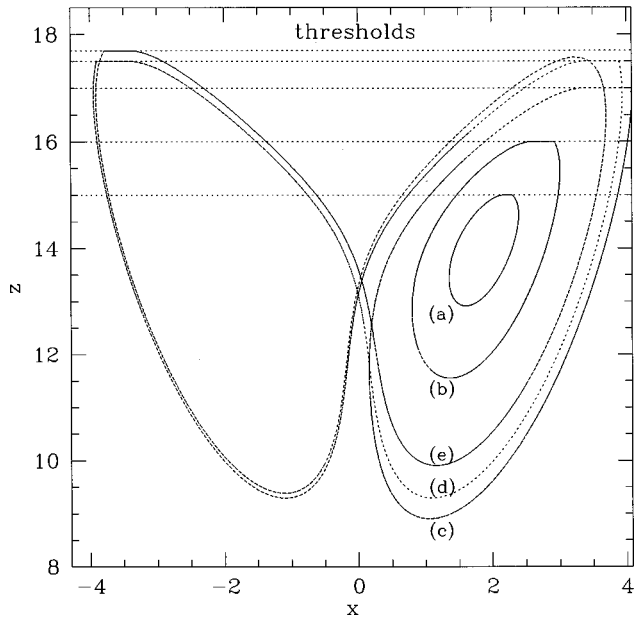

FIG. 17. Chaotic laser (with $\sigma=2$, $r=15$, and $b=0.25$), with variable z under threshold mechanism, with threshold equal to (a) 15, (b) 16, (c) 17, (d) 17.5, and (e) 17.7. The chaotic orbit yields limit cycles whose size is determined by the value of the threshold. The dotted lines indicate the threshold cutoffs.

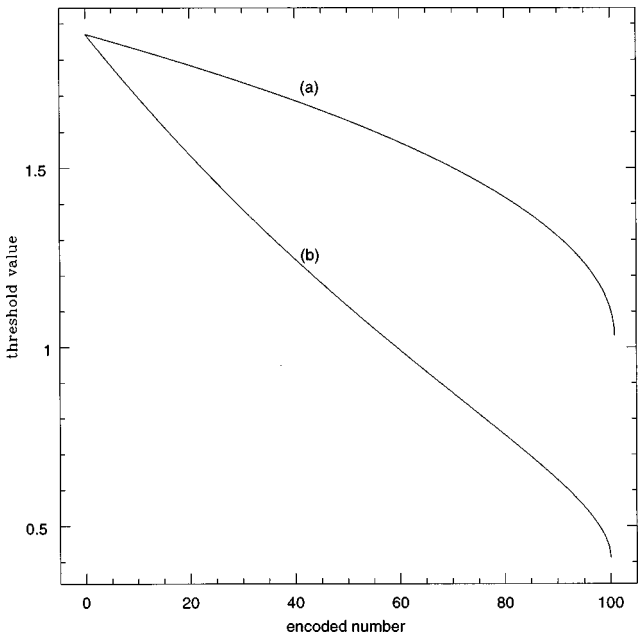

FIG. 19. Look-up graph of encoded number vs threshold values for the chaotic laser system with threshold mechanism implemented on the (a) y variable and (b) x variable (again the largest number encoded is 100).

threshold condition on any of the three variables x, y, and z (see Figs. 18 and 19). For instance, using the threshold mechanism on the z variable of the laser system, one obtains a large range of excess emission. The dependence of excess emission on threshold is linear (see Fig. 18). This makes encoding via scheme 1 particularly easy.

The other interesting thing is that the threshold interval giving encoding/addition is proportional to r, where r is the nonlinearity parameter corresponding to pump rate. So higher pump rates give bigger ranges of operation.

To perform addition on m numbers we set the threshold of m connected chaotic laser units such that each encodes a term in the sum. The excess emitted from a unit drives its neighboring one, with the unit encoding the last term of the sum having the open edge with the lead registering the output. After a chaotic update an avalanche sweeps across the threshold coupled units (as demonstrated in Fig. 20), giving rise to an excess emission from the open edge, which can directly be associated with the result. The addition operation is then achieved simply as follows: Input the threshold values from the look-up table to encode the numbers to be added and then register the emitted excess from the open

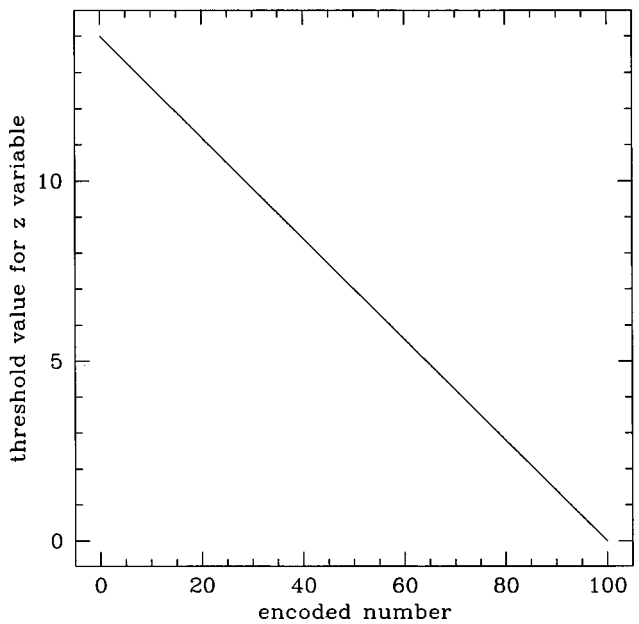

FIG. 18. Look-up graph of encoded number vs threshold values for the chaotic laser system with threshold mechanism implemented on the z variable (here the largest number encoded is 100).

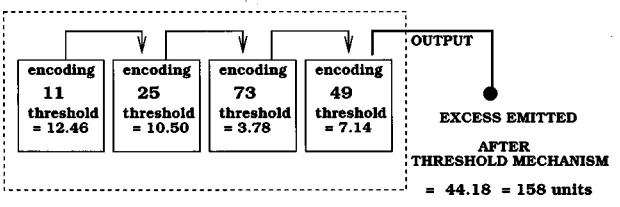

FIG. 20. Threshold coupled chaotic laser units emulating an adding machine: Here we are adding four integers 11, 25, 73, 49, each encoded by a chaotic laser unit with threshold fixed from the lookup graph of Fig. 17, such that they emit 11, 25, 73, 49 units of excess, respectively (here the unit of excess emission is $\delta=0.2796$). These elements encoding the terms are then threshold coupled for the addition operation. The ejected excess from the element encoding 11 ($=11\times\delta$) drives the element encoding 25 and so on, up to the element encoding 49, from whose open boundary the collective excess is emitted to the output lead. This emitted excess ($=44.18$) is exactly the sum $11+25+73+49=158$ (in units of δ). For serial addition we have a linear chain configuration and computing time is equal to the number of terms in the sum (which is 4 here).

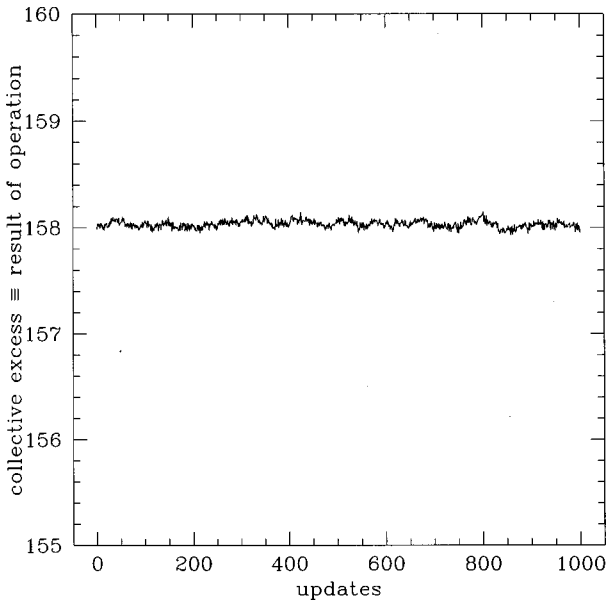

FIG. 21. Here we have noisy chaotic laser units threshold coupled to emulate an adding machine. The strength of uniform random noise in the state variables of the various units is 0.01. We are adding the same set of four integers 11, 25, 73, 49, as in Fig. 19. Each term is again encoded by a chaotic laser unit, with the collective excess ejected at the readout providing the result of the operation. The figure shows this quantity over several dynamical updates. Clearly the emitted excess fluctuates very minimally around the sum $11+25+73+49=158$ (in units of δ). Thus the result of the addition operation is robust to small noise in the system.

edge at the end of one dynamical update. The dynamics of the lattice is such that this emitted excess is the required answer. We can have a linear or branching chain of chaotic laser units for serial and parallel addition. The operation is again *commutative*, as the ordering of the terms (elements) does not influence the answer.

The result of the operation is reasonably insensitive to noise. For instance, Fig. 21 shows the same addition done in the presence of additive random noise. Clearly the ''result'' of the operation fluctuates only very slightly about the correct answer.

Multiplication can be performed (as an extension of addition), invoking the same parallel computational approach through branching lattices. For instance, to do $m\times n$ we can have a lattice with n branches, each branch being a copy of the unit encoding m. The total ejected excess will be the answer $m\times n$. Alternately again, we can do $m\times n$ by taking the element representing m and collecting the ejected excess over n time steps, i.e., the quantity accumulated over n units of the local chaos clock. This evidence of computational ability from continuous time multidimensional systems indicates that our scheme seems to have definite possibilities of expansion and opens up concrete experimental possibilities with ultrafast optics.

VI. DISCUSSION

We have presented here a purposefully simple dynamical system and shown how we can program it to perform both general and specific computations. While, in some sense, every physical system can be thought of as an ''analog'' computer, the trick is to make specific analog computers, such as dynamical systems, perform the computations we desire. To our knowledge, it is surprising that chaotic systems can be programmed to perform such a wide variety of computations.

We tried to include a diverse enough group of computations (logical operations, arithmetic operations, and specialized dynamical algorithms) to demonstrate that dynamical systems can be simply and flexibly programmed to compute. Further, we have provided a specific application to a laser system. In light of these results, it can be envisaged that a high-speed chaos computer might be constructed that exploits fast chaotic lasers for computation. The advantage of using chaos computing in this case would be that programming could be accomplished by slight changes to the dynamical system to perform high-speed addition, multiplication, etc.

Our general strategy here was to investigate the opportunities provided by nonlinear dynamics to constitute an effective computing medium, exploiting the determinism of dynamics on the one hand and its richness on the other. In contrast to efforts to bring computational models and physics closer together starting from the computer model end (such as taking the digital dynamics and adding physical properties and constraints to it [16] or efforts to use the theory of computation to describe/quantify the complexity of physical systems [17]), we start from the physics end and explore the possibilities chaos has to offer to computation. While certain ''physicslike'' models, for instance, cellular automata, have been investigated extensively as candidates for computing, chaotic dynamics was still to be explored as a computing medium. Here we have demonstrated the possibility of computing with chaos, which *a priori* may seem surprising.

Note that nonlinearity in the processing units is clearly necessary for various Boolean/arithmetic implementations, though these units need not necessarily be chaotic. However, only chaotic dynamics will ensure the capacity to get *all* the different applications from the *same* processing units. That is, we can ''control'' the chaotic map to the dynamics required for the application at hand and only the fully chaotic case can be ''pruned'' to all possible behaviors, as applications demand.

It is evident that dynamical computing has potential and flexibility, arising from the *wide range of behaviors each module is capable of, through the variation of a single (''programmable'') parameter*. That is, the chaotic elements present a range of possibilities with the *same* collection of elements (i.e., using the same hardware) by simply changing the threshold (which is fed in as input and is part of the software). Specific applications of this versatility are the encoding schemes and the dynamical algorithm for finding the LCM. (See Tables III and IV for a summary.) Further, the knowledge of the dynamics of the nonlinear system constituting the hardware allows us to exactly specify the thresholds that yield the required inputs or operations (like a ''machine language'') and this makes programming of our system simple and direct.

Interestingly, note that from another viewpoint, our system has the capability of ''changing'' its hardware, through its software, as the chaotic elements constituting the processors can change their behavior depending on the threshold value they receive (which is part of the ''program''). There-

TABLE III. Analogs of computing devices and operations in the chaotic network.

Computing devices/operations	Analogs in the chaotic lattice
Basic hardware: processors	Chaotic elements in a lattice/chain/array/network
Communication of data/information	Transport induced by threshold mechanism couples the elements through the cascade of emitted excess (''avalanching'')
Programming	Setting threshold value
Input (''source'')	Feeding in a stream of threshold parameters
Output (an external ''visible'' state of the computing system)	Collective response of the system (can simply, clearly, and consistently be associated with the result of the operation)
The ''value'' sent by the system to the output lead (''sink'')	Excess emitted from the open boundary of the array
Analog features	Continuous state variables and threshold parameters system dynamics ''emulates'' the operation
Digital features	Discrete elements binary representations employed in arithmetic operations
Parallel operation	Highly branching arrays evolving synchronously

fore, they can serve as ''programmable hardware'' [18].

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 [19,20] and quantum computing [21,22], which also aim at discovering alternative ways of exploiting physical phenomena, well understood in the context of physics, to do computations. There too it is still not clear that these computing systems, first presented as *alternate computing paradigms*, can perform better than digital computers (although they hold great promise) [23]. Indeed, we choose our coupled logistic map lattice not from speed or optimization concerns but from a ''proof-of-concept'' concern, much like Aldeman demonstrated the feasibility of DNA computing in 1994, by solving a seven-node Hamiltonian path problem, a special case trivial to solve by conventional computer. In contrast to the DNA and quantum paradigms, which are geared to handle *specific* problems suited specially to itself, 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 [24].

In our work we have tread the middle ground between very abstract mappings of dynamical systems onto the UTM and very concrete realizations of specific computing devices with complicated systems. For instance, it can be straightforwardly shown that coupled map lattices are equivalent to synchronous concurrent algorithms (SCAs) [7,25]. Along similar lines it can easily be shown that our dynamical system is also equivalent to a SCA. While this is assuring, as we now know that our system will work ''in principle,'' it was not our aim here to merely state this. Instead, we have demonstrated explicitly how chaos can yield specific arithmetic/

TABLE IV. Potential advantages of computing with distributed chaos.

Extensive range of possibilities with the same collection of elements
cycles of *all* orders can be emulated by simply changing the threshold
Versatile and flexible: each element is capable of a very wide range of behaviors
through variation of a single (programmable) parameter
exploiting the richness of chaotic dynamics
Inherently highly parallelizable
Controlled, potentially general purpose, applications possible:
as the chaotic elements can be made to yield *exact* cycles of *any* desired order
Simple and direct:
(only *one* adjustable parameter, the threshold, yields *all* arithmetic/logic operations)
Implementation simple:
do not have to monitor each element individually,
simply tap the response from one specified open edge
this response is associated with the answer in a clear and consistent way

logic operations. Alternately, there are certain concrete approaches to the computing question. For instance, certain chemical systems can be very delicately tuned to yield some logic gates [8]. There are many crucial parameters in such models (involving both the construction of the apparatus as well as the geometric configuration and timing of the input/output waves). Fine adjustments of these lead to the desired phenomena. In contrast to such attempts, here we have a very simple and general scenario, with only *one* adjustable parameter defining both arithmetic and logic operations and giving robust responses that can emulate the answer/output.

Finally, we would like to discuss this ''computing-with-chaos'' principle in relation to the two existing computing cultures: namely, the conventional algorithmic way (on which the structure of working general purpose computers is based) and the neural net, ''experience-acquisition'' style [26]. We have tried to emulate, through the spatiotemporal responses of distributed chaos, what an algorithmic computing machine is capable of doing. Our computing paradigm then enjoys the advantage of being *direct* and *controlled*. In fact, it is quite amazing how adaptive coupling allows one to use chaos in such a controlled manner. Chaos computing is then implementable very consistently. The system while evolving chaotically, processes information reliably (and ''predictably'').

Clearly our ''dynamic computing'' is very different from neural computing in style and content. Neural nets do not have any natural intrinsic dynamics. We, on the other hand, are computing with chaos. Loosely speaking, this is like saying that the analog of the constituent ''neurons'' in neural nets is functionally simple, while in our model it has natural chaotic response, and thus is behaviorally far richer.

Neural nets are closely tailored to specific tasks, whereas here we have a bunch of potentially general purpose processors, which can handle different arithmetic/logic operations (communicated through a stream of parameters). So our computing paradigm is more versatile.

Further, there is nothing in our computing that is analogous to a sequence of ''weight adjustments'' to match target truth tables [26]. Our system does not, in the style of neural nets, try to adjust its internal coupling to deliver the desired response. Instead, our knowledge of the hardware, namely, the determinism of the chaotic evolution, allows us to *exactly* specify the coupling that will yield the required operation. Thus we have tried to exploit our knowledge of the physics of the constituent hardware to enable us to ''program'' the chaotic elements at the ''machine level'' (for instance, we exploit exact solutions for ''look-up tables'' to implement encoding). Our computing then needs *no ''learning time'' for tasks* and is consequently faster. While there will be considerable effort (and anticipated limitations of techniques) in determining the specific physics of the problem at the design level, having ''constructed'' or ''synthesized'' the computing device, one will not need any additional overheads for performing basic encoding/logic/arithmetic operations (as these will be ''hard wired'' so to speak).

Finally, note that our computing principle shares one (very advantageous) feature of neural computing. It is also inherently highly parallelizable, in the sense that the distributed elements can evolve synchronously, i.e., the system can execute several operations concurrently [26]. In summary, we have demonstrated that extended chaotic systems are capable of performing computations through a rich variety of emergent spatiotemporal properties.

[1] T. Shinbrot, C. Grebogi, E. Ott, and J. Yorke, Nature (London) **363**, 411 (1993).

[2] L. M. Pecora and T. L. Carroll, Phys. Rev. A **44**, 2374 (1991); W. L. Ditto and L. M. Pecora, Sci. Am. (Int. Ed.) **269**, 62 (1993).

[3] S. Hayes, C. Grebogi, E. Ott, and A. Mark, Phys. Rev. Lett. **73**, 1781 (1994).

[4] G. D. Van Wiggeren and R. Roy, Science **279**, 1198 (1997).

[5] S. Sinha and W. L. Ditto, Phys. Rev. Lett. **81**, 2156 (1998).

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

[7] A. V. Holden *et al.*, Chaos **2**, 367 (1992).

[8] A. Toth and K. Showalter, J. Chem. Phys. **103**, 2058 (1995).

[9] S. Sinha and D. Biswas, Phys. Rev. Lett. **71**, 2010 (1993).

[10] S. Sinha, Phys. Rev. E **49**, 4832 (1994); Int. J. Mod. Phys. B **9**, 875 (1995).

[11] S. Sinha, Phys. Lett. A **199**, 365 (1995).

[12] P. Bak, C. Tang, and K. Wiesenfeld, Phys. Rev. Lett. **59**, 381 (1987); Phys. Rev. A **38**, 364 (1988).

[13] There is an additional feature that can potentially lend flexibility to the response of the chaotic network. The variation in the number of updates after which the emitted excess is ''measured'' also lends rich variety to the emergent responses. This feature is exploited later in an arithmetic application.

[14] To do the multiplication $m \times n$, one can also feed in the excess emerging from the open edge of the lattice representing m back to the first element for n time steps. After that one can simply measure the excess ejected from the open edge, which gives the ''answer.''

[15] 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. **61**, 223 (1995).

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

[17] 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 Young 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.

[18] G. Taubes, Science **277**, 1935 (1997).
[19] L. M. Aldeman, Science **266**, 1021 (1994).
[20] R. Pool, Science **268**, 498 (1995).
[21] 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, 1994), p. 124.
[22] A. Steane, Rep. Prog. Phys. **61**, 117 (1998).
[23] 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 that 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).
[24] 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.
[25] SCA is a model used in computer science to give a unified theory for a number models of parallel computing (including neural nets) [7].
[26] I. Aleksander and H. Morton, *An Introduction to Neural Computing* (Champman and Hall, London, 1990).