Parallel computing with extended dynamical systems

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

We discuss the scope of parallelism based on extended dynamical systems, in particular, arrays of chaotic elements. As a case study we demonstrate the rapid solution of the Deutsch-Jozsa problem, utilizing the collective properties of such systems.

DOI: 10.1103/PhysRevE.65.036214 PACS number(s): 05.45.−a

I. INTRODUCTION

The controlled computational capability of networks of chaotic elements was demonstrated recently with the direct and flexible implementation of fundamental logic and arithmetic operations [1,2]. The general strategy in these efforts was to investigate the opportunities provided by nonlinear dynamics to build an effective computing medium, exploiting the determinism of dynamics on one hand, and its richness and complexity on the other. Here we will discuss the scope of dynamics-based parallelism. As a case study, we will attempt the rapid solution of the Dentsch-Jozsa (DJ) problem [3], utilizing collective dynamical properties of strongly nonlinear extended systems.

The rapid solution of the Deutsch-Jozsa problem by quantum methods was one of the first dramatic demonstrations of the power of quantum computing. In this work, we will utilize collective properties of extended nonlinear dynamical systems to reduce computational effort in the solution of this benchmark problem. We first review the problem below.

The Deutsch-Jozsa problem can be stated as follows. Let U_f be a device that computes a function f. Given an input i, U_f will, after some time, output the value $f(i)$. In general terms, the class of computational tasks that is being considered here involves being given U_f and then using it to determine some property $G[f]$ in the shortest possible time. [G is some function of the sequence $f(0), f(1), \ldots$.]

In particular, consider a k digit binary integer variable i, i.e., a string of length k with entries 0 or 1. The entire $N=2^k$ possible combinations of 0's and 1's are valid inputs for the function. The function $f(i)$ is defined on this k-bit domain space to a 1-bit range space [$f(x)=\{0,1\}$]. Generally, there are $2^N=2^{2^k}$ functions from the N strings to {0, 1}, since each of the N strings can be mapped to either 0 or 1. For example, for $k=3$, there are $N=2^3=8$ strings, 000, 001, . . . , 111, and $2^8=256$ functions. Consider two functions such that (1) $f(i)=C$ is constant for all the N possible input values, i.e., all outputs are 0 or 1; (2) $f(i)=B(i)$ is 0 for $N/2$ input values and 1 for the other half, i.e., the function is balanced as the N outputs are a sequence of equal numbers of 0's and 1's (in any order).

There are only two constant functions possible: one gives 0 for all the N input values; the other gives 1 for all the N input values. There are a large number of different balanced functions possible, each corresponding to a distinct output sequence of 0's and 1's. Specifically, the number of balanced functions for a k-bit DJ problem is given by straightforward combinatorics to be $C^N_{N/2}=N!/(N-N/2)!(N/2)!$, where $N=2^k$, since $N/2$ out of N strings map to 0 and the rest to 1. In the case of $k=3$, i.e., $N=8$, there are $C^8_1=70$ balanced functions.

The problem posed by Deutsch-Jozsa was to determine from a sequence of outputs whether the function generating the outputs was constant or balanced [3]. The computational effort in solving this problem can be put as follows: What is the minimum number of function calls required before you are sure if you have a constant or a balanced function? The standard mathematical theory that is used to study the possibilities and limitations of computing, based on Turing machines (which can be viewed as an abstract model of today's computers) would solve the problem by executing U_f repeatedly to obtain the values of sufficiently many outputs in order to determine the class of function with certainty. In the worst case, for instance, where the first $N/2=2^{k-1}$ outputs are 0 and the next $N/2$ outputs are 1, a Turing machine would take $2^{k-1}+1$ tries to obtain the first output of 1. Hence to deduce the function class with certainty, one can need up to $2^{k-1}+1$ function calls. The difficulty in solving this problem using a standard Turing machine then grows exponentially with the number of bits in the input string.

Treating this DJ problem as a case study here, we will indicate in general the parallelized problem solving conceivable with extended complex systems. The general strategy of using extended systems for parallelizing tasks employs arrays of dynamical systems, with the size of the array being determined by the number of synchronous subtasks the primary task can be broken into. This is a general form of parallelism and can be applied to a range of computing applications.

The most straightforward way of implementing the DJ problem in this conventional parallelism is to let each device take one input and return the output, and after all the inputs have been executed in parallel one can combine the outputs by an OR operation. If the result is 1, it is balanced; otherwise it is constant.

One can also conceive of further reduction of computational effort in solving the DJ problem, through the observation of some collective physical property or response of the extended system, i.e., the problem is set up in such a way

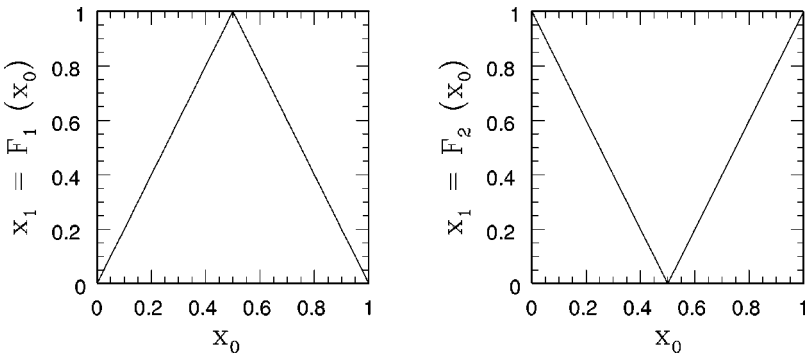

FIG. 1. The two complimentary functions F_1 and F_2.

that a collective property directly indicates the ''answer'' to the problem, without necessitating individual measurements. We give a few illustrative implementations of this idea explicitly below.

II. LATTICE OF NONLINEAR MAPS

Consider, for instance, an array of size $2^k=N$, where each element i, $i=1,\ldots,N$, evolves under some suitable map. Such an array can yield N outputs simultaneously, with each output being encoded by the state of an element, e.g., the jth output is encoded by the state $X_n(j)$ of the jth element at some time n. Thus, this array can process a k-bit domain space (which has 2^k outputs corresponding to the 2^k possible input combinations) concurrently.

So each spatial element is a ''dynamical device,'' the evolution of whose state is governed by some appropriate iterated map F of an interval onto itself ($X\in[0,1]$ specifically). The output is encoded in the state of the element after some specified transience time n. For instance, one can use the following encoding scheme to generate 0's and 1's: when the iterate is left of center, i.e., $X<1/2$, the map returns 0, and when it is right of center, i.e., $X\geqslant 1/2$, it returns 1.

For the constant function C we can have each local map given by

$$F_C(X)=rX, \tag{1}$$

where $r\ll 1$.

For all $r<1$ this system will rapidly evolve exponentially to the fixed point at $X^*=0$. So the state X of all elements in the array, which determines $f(i)$, is *always* ~ 0. That is $X_n(j)\leqslant 1/2$ for all j thus encoding 0. This is equivalent to obtaining a constant (0 in this example) for all outputs, which is the action of the constant function.

To implement the balanced function we can use the tent map

$$F_B(X)=1-2|X-1/2|. \tag{2}$$

Since the invariant probability density of the map is flat and uniform [4], the iterates are equally probable in both halves. Therefore, starting an array with uniform random initial conditions will yield after transience, on average, an equal number of 0's and 1's, as the $X_n(j)$'s at any n will be equidistributed on the right and left. The state of the array is dynamically changing and at different times different elements will be right or left of center. In principle then, probabilistically speaking, all possible balanced functions can be attained by the state of the array at different times n.

Alternatively, we can implement the problem as follows. The inputs, which are binary strings of length k: $a_1a_2\ldots a_k$, can be encoded as binary fractions X lying in the interval [0, 1],

$$X=0.a_1a_2a_3\ldots=\sum_{j=1}^{k} a_j 2^{-j}, \tag{3}$$

where a_j is either 0 or 1. Without loss of generality, we arrange binary numbers in the function domain in increasing sequence. For example, for $k=3$ the numbers are arranged as 000, 001, . . . , 111.

The output from each spatial element can be the coarse-grained first forward iterate of the maps $X_1=F(X_0)$. If $X_1\geqslant 1/2$, U_f returns 1; otherwise, the return is 0. That is, the first digit of X_1 in binary fraction representation determines the output $f(i)$, since this is 1 if $X_1\geqslant 1/2$ and 0 otherwise. Thus we obtain a function $f(x)$ from the k-bit domain space ($X_0=0.a_1a_2\ldots a_k$) to a 1-bit range space {01} given by the first digit of $X_1=F(X_0)$.

For the constant function C we can again have each local map given by

$$F_r(X)=rX, \tag{4}$$

where $r<1/2$.

For all realizations of the k-bit string (the inputs i), each being encoded as some number $X\in[0,1]$, the map with $r<1$ will exponentially rapidly evolve to $X^*=0$. Specifically, say, $r=2^{-m}$ in Eq. (2). Then the action of F on $X=\Sigma_{j=1}^{k}a_j2^{-j}$ yields

$$F_{2^{-m}}(X_0)=X_1=\sum_{j=1}^{k} a_j 2^{-j-m}=0.000\ldots a_1a_2\ldots a_k. \tag{5}$$

Thus the effective action of F here is to create m 0's after the binary point for the first dynamical iterate. The first digit of any X_1, which determines $f(i)$, is then always 0 if $m\geqslant 1$, i.e., $r\leqslant 1/2$. This is equivalent to obtaining a constant 0 for all inputted strings, which is the action of the constant function.

TABLE I. In general, 2^N functions can be generated by the functions F_1 and F_2. For $k=3$, i.e., $N=8$, this number is 256. The first column gives the number of 1's obtained from the eight inputs. The balanced function gives half of the eight outputs to be 1 and has 70 possible combinations. The first and last entries yield constant functions, as they have either all 0's or all 1's.

Number of 1's obtained	Combinations possible
0	1 (constant function 0)
1	$C_1^8=8$
2	$C_2^8=28$
3	$C_3^8=56$
4	$C_4^8=70$ (balanced function)
5	$C_5^8=56$
6	$C_6^8=28$
7	$C_7^8=8$
8	1 (constant function 1)

To obtain different balanced functions we evolve the elements $X(j)$, $j=1$, N through two different nonlinear evolution function, F_1 and F_2 (see Fig. 1):

$$F_1(X)\equiv F_B(X)=1-2|X-1/2| \tag{6}$$

and

$$F_2(X)=1-F_1(X)=2|X-1/2|. \tag{7}$$

When $F_1(X)\geqslant 1/2$ it returns 1, and otherwise returns 0. When $F_2(X)>1/2$ it returns 1, and otherwise 0. Each of the individual evolution map functions, F_1 and F_2, have equal probability of returning 0 or 1 for any random input (initial condition). The section of the interval yielding 0 for F_1 gives 1 for F_2, and vice versa. In this sense the functions are complimentary, and cover both the possibilities, i.e., an input state can give an output of either 0 or 1 depending on the function chosen. This is evident from Fig. 1. By using such complementary functions one can implement all 2^N possible functions, including all possible balanced functions. Tables I and II summarize the combinatorial properties of these functions [5].

TABLE II. The table shows the number of different combinations that generate the different balanced functions for $k=3$, i.e., $N=8$, via the complementary functions. The first column gives the number of elements evolving under the dynamical map F_1. The rest of the elements evolve under F_2. The second column gives the number of distinct output combinations possible in each case (each yielding a different balanced function), and they all add up to a total of 70.

Number of elements evolving via F_1	Combinations possible
8	1
6	16
4	36
2	16
0	1

Thus the dynamics is as follows. The elements of the lattice $[X_0(j), j=1, \ldots, N]$ evolve one forward iterate by F_1 or F_2. The state of the lattice after a dynamical step $[X_1(j), j=1, \ldots, N]$ encodes all the outputs simultaneously. Figure 2 schematically shows the local maps of the array implementing the constant function and that implementing two particular balanced functions. All possible balanced functions can be obtained this way.

If we do not need the values of the individual outputs $f(i)$ but simply have the task of finding out if any one of them is nonzero (as in the DJ problem), we can employ some collective physical property to solve the task. Such measurements are often reasonably easy in extended systems, and can give the required result directly and rapidly, bypassing the 2^k individual measurements and subsequent OR operation necessary otherwise.

For instance one can observe some mean-field-like property:

$$h=\frac{1}{N}\sum_{j=1}^{N} f\{X(j)\}. \tag{8}$$

When $f(X)\equiv X$, the value of this mean field $h \equiv (1/N)\Sigma_j X(j)$ is ~ 0 for the constant function C, and is $\sim \frac{1}{2}$ for any one of the $C_{N/2}^N$ distinct balanced functions (implemented either via evolution function F_B or via combinations of F_1 and F_2) [6]. Thus the mean field bears a very distinct signature of the two classes of functions, and can be used to decipher, with certainty, which one of the two functions we have.

Note that a coarse estimate of the mean field is adequate here, since the mean-field values corresponding to the constant and balanced classes of functions are so far apart. Thus if one designs a device where such mean-field-like quantities can be obtained (even to fairly low precision) via one direct measurement, we can bypass the 2^k individual output measurements and a subsequent OR operation, and obtain the result directly.

Clearly, noise is not a problem in this implementation (and in the implementations listed subsequently), since the scheme involves only coarse-grained quantities. Also, the evolved state encoding the answer involves very short evolution times (just the first iterate here, for instance) and this allows fast operations and also ensures that errors do not expand.

In summary, by finding suitable collective properties that bear the clear and unambiguous signature of the dynamics simulating the different classes of functions, and by designing devices that allow such collective properties to be measured directly, one can reduce computational effort in the DJ class of problems.

III. ARRAY OF OPTICAL DEVICES: EXAMPLE OF A HYBRID APPROACH

Hybrid schemes incorporating processors of dimension d in arrays of length L yield parallelism of degree dL. Here we present a specific realistic implementation of a hybrid scheme, using a set of optical devices, such as a unidirec-

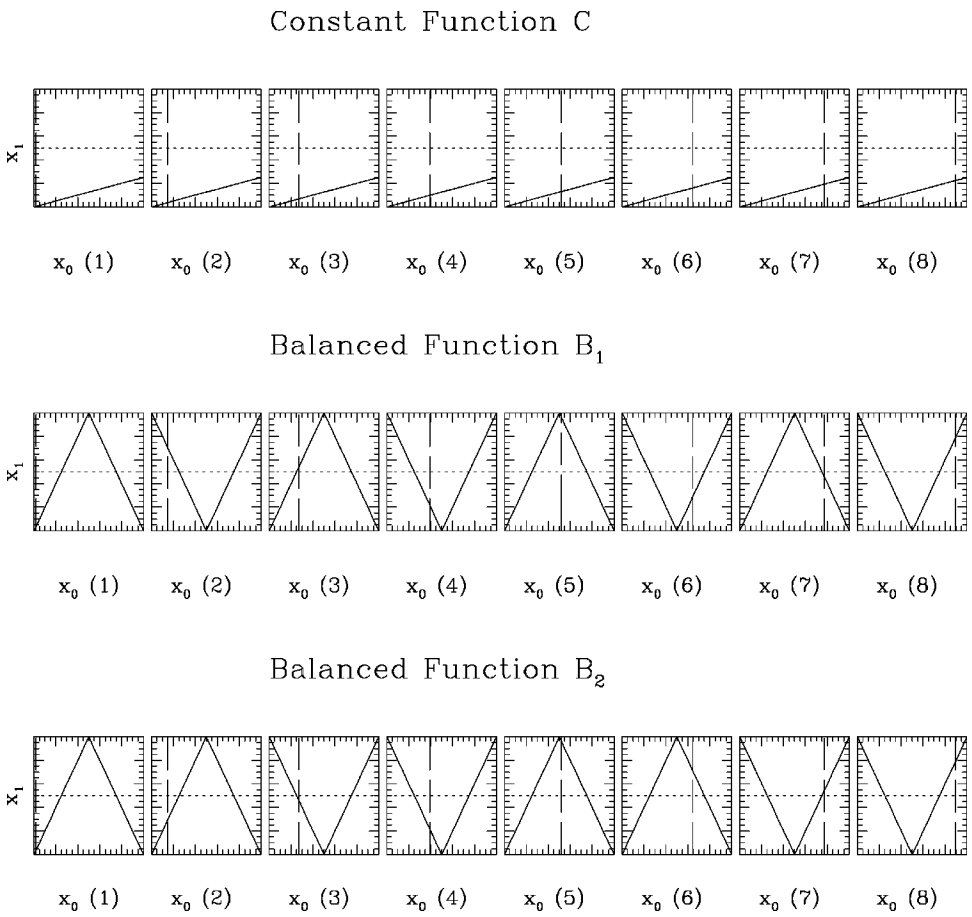

FIG. 2. The DJ problem for the case of $k=3$, i.e., $2^3=8$ inputs, implemented by spatially distributed parallelism. The inputs are encoded as binary fractions [via Eq. (3)] and have values 0, $\frac{1}{8}$, $\frac{1}{4}$, $\frac{3}{8}$, $\frac{1}{2}$, $\frac{5}{8}$, $\frac{3}{4}$, $\frac{7}{8}$ encoding 000, 001, 010, 011, 100, 101, 110, 111, respectively. For the constant function all eight elements, each encoding an input in its initial state (indicated by the vertical dashed line), evolves via the return map $X_1=rX_0$ with $r=\frac{1}{8}$ here. It is clear that for all the inputs, all the output states X_1 (given by the intersection of the vertical dashed line with the rX_0 line) lie below the horizontal dotted line indicating the value of $\frac{1}{2}$. Since all $X_1<1/2$, all eight outputs are 0. For the two balanced functions, on the other hand, the elements evolve via $X_1=F_1(X_0)$ [given by Eq. (6)] or $F_2(X_0)$ [given by Eq. (7)], and the inputs can now give $X_1\geqslant 1/2$ for half the cases. The first balanced function B_1 gives an output of 1 for the four inputs: 001, 010, 100, 111, and zero for the other four while B_2 gives output 1 for the four inputs: 100, 101, 110, 111, and 0 for the others.

tional ring cavity, containing an active medium of two-level atoms homogeneously broadened and interacting with a coherent electromagnetic wave. The envelope of the electric field E_n for successive round trips, indexed by n, obeys the mapping [7]:

$$E_{n+1}=A+BE_n\exp(i|E_n|^2). \tag{9}$$

The parameter A is proportional to the coherent external field and B is an attenuation factor. This map has been observed in hybrid optical bistable devices with delayed feedback [8] and also in an all-optical bistable device using a single-mode optical fiber as a nonlinear medium in a ring cavity pumped by a train of mode locked pulses [9].

Thus each device j can encode the input through its initial conditions $E_0(j)$. Since $E_0(j)$ is complex valued, it can encode *two* inputs, one through its real part and another through its imaginary part. The same scheme of encoding inputs namely, as a binary fraction, as given in Eq. (1), can be followed. All the N outputs are encoded in the coarse-grained real and imaginary parts of the evolved state $E_n(j)=a_n(j)+ib_n(j)$, $j=1,\ldots,N/2$.

When the parameters of the devices are around $A\sim 3$ and $B\sim 0.3$, the evolution is chaotic, with the states of the different units $E_n(j)$, $j=1,N/2$, fluctuating wildly from element to element at any particular snapshot of time n. Say the $a_n(j)$'s and $b_n(j)$'s lying above a prescribed cutoff encode an output of 1 and the rest encode 0. The coarse-grained evolved state $E_n(j)$, $j=1,N/2$, can then yield all possible balanced functions through a suitable choice of n and the cutoff.

When parameter $A<1$ the elements evolve to fixed points. So all the units (after transience) will have identical values. This is the analog of the constant function.

Consider the specific example of $k=3$, i.e., $2^3=8$ inputs. Since for a k-bit problem one needs 2^{k-1} devices (as each device processes two inputs simultaneously), we employ four optical elements: $E_n(j)=a_n(j)+ib_n(j)$, with $j=1,\ldots,4$. The inputs are encoded as initial $a_0(j)$, and $b_0(j)$, $j=1,\ldots,4$. Thus, in binary fraction notation, input

000 gives $a_0(1)=0.000$, input 001 gives $b_0(1)=0.001$, input 010 gives $a_0(2)=0.010$, input 100 gives $b_0(2)=0.100$, etc.

One can obtain a constant function with system parameters $A<1$, $B\sim 0.3$. This will give an output (after short transience time) to be $a(1)=a(2)=a(3)=a(4)=a_{\text{fixed}}$ and $b(1)=b(2)=b(3)=b(4)=b_{\text{fixed}}$. So any cutoff greater than a_{fixed} and b_{fixed} will give an output of 0 for all j at all times n.

To obtain balanced functions one can set the parameters to be $A=3$, $B=0.3$, which gives a chaotic mapping. For instance, a balanced function can be obtained by the electric fields at $\tau=5$, $E_5(j)$, $j=1,\ldots,4$, which gives $a(1)=3.694\,76$, $a(2)=2.091\,72$, $a(3)=3.339\,39$, $a(4)=2.128\,27$, and $b(1)=0.399\,02$, $b(2)=-0.057\,57$, $b(3)=-0.582\,17$, $b(4)=0.281\,57$. The mean value of a is 2.813 54 and the mean value of b is 0.0102. Using the mean as the cutoff, we have the eight outputs corresponding to the eight inputs to be 1, 1, 0, 0, 1, 0, 0, 1. Similarly with the state at $\tau=20$, one can obtain a balanced function with outputs 1, 0, 0, 1, 1, 0, 0, 1, etc.

Now one can extract a small fraction of the output of each device and mix it to give a mean field,

$$h_n=\frac{1}{N}\sum_{j=1}^{N}|E_n(j)|^2, \tag{10}$$

where $E_n(j)$ is the electric field of the jth unit. The setup could be designed to yield this mean-field output directly, in which case the above averaged quantity can be obtained through one measurement.

After transience, if we measure the mean field of the devices [given by Eq. (9)] we find that for the constant function this mean field is constant in time with values bounded from above by 2, while for any balanced function, where the mappings are necessarily chaotic, the mean field fluctuates and has values bounded from below by 8. So the mean field bears a clear and unambiguous signature of the nature of the functions. If the mean field has a value of less than 2 this is clearly a constant function, and *vice versa*. So, if the task at hand is only to determine whether the function is balanced or constant, one can do this with certainty through one direct measurement of the mean field.

IV. THRESHOLD COUPLED ARRAY

Consider another example of a network of $2^k=N$ chaotic elements $x(1),x(2),\ldots,x(N)$, where each chaotic element has two basic dynamical phases [10,2].

(1) *Chaotic update*. Synchronous global changes from time t to $t+1$ of the elements in the network. This is governed by a chaotic evolution map, say, to be specific, the logistic map: $F(x)=rx(1-x)$ with $r=4$, which that maps interval [0, 1] to itself.

(2) *Adaptive phase*. Between chaotic updates there is an adaptive phase that consists of local changes triggered by elements in the network having a state x greater than some critical threshold value x^*, i.e., $x>x^*$. When this occurs, the overcritical element, say the ith element $x(i)$ relaxes (reverts) to the threshold value spilling the "excess" $Z=\{x(i)-x^*(i)\}$ over to its nearest neighbor down the array,

$$x(i)\rightarrow x^*(i),$$

$$x(i+1)\rightarrow x(i+1)+Z.$$

This excess can snowball into an "avalanche" of excess down the array, in a kind of domino effect, with the cumulative excess being emitted from the "open end" of the network, i.e., sent outside the system. When an element has state $x(i)\leqslant x^*(i)$ its state remains at $x(i)$, and no interelement transfer takes place.

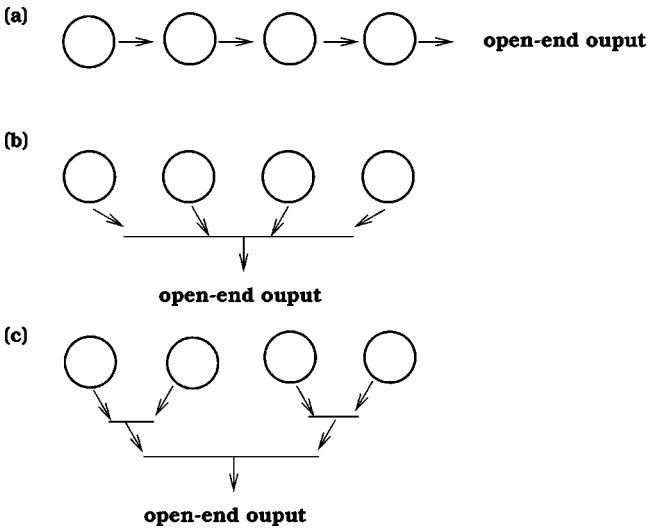

FIG. 3. A mesh of four threshold coupled elements ($N=4$): (a) sequentially connected—here the adaptive phase takes $N=4$ adaptive steps; (b) parallel connected—here the adaptive phase takes one adaptive step, as all elements relax simultaneously emitting excess to the open end; and (c) connected in a binary tree—here the adaptive phase takes $\ln_2 N=2$ steps.

The adaptive phase can be sequential, partially parallel, or entirely parallel (see Fig. 3). In the case of sequential connections, the entire adaptive process takes $O(N)$ adaptive steps. When the elements are connected in parallel, all individual leads are wired together to give a collective open-end readout. Thus, the relaxation of all the elements is simultaneous, and the adaptive step is of $O(1)$. The clock of the dynamical system is set by the chaotic update though, since that time scale is independent of the topology of the connections and threshold settings. The adaptive time (which varies for different threshold settings, and topologies of the mesh) is required to be faster and the adaptive process occurs to completion between the chaotic updates.

We illustrate this threshold coupled array with a simple example. Let us consider two coupled elements, i.e., $N=2$, with thresholds set at $x^*(1)=0.5$ and $x^*(2)=0.25$. The input state at time $t=0$ is given as $x(1)=0.25$, $x(2)=0.5$. At $t=1$ chaotic update takes place on those input values and an adaptive phase follows. At $t=2$ again the next chaotic update occurs, followed by the adaptive phase, and so on. Table III shows step-by-step changes of the system over three units of time. Output from $x(2)$ represents the cumulative excess

TABLE III. Step-by-step changes of an $N=2$ system with two coupled elements $x(1)$ and $x(2)$, over three units of time. The thresholds here are set at $x^*(1)=0.5$ and $x^*(2)=0.25$. Output from $x(2)$ represents the cumulative excess emitted from the open end and is equal to 1.0 for all times for these threshold values. The states of the individual elements $x(i)$ are equal to $x^*(i)$ for all times after the first transient step at $t=1$.

	$x(1)$	Output from $x(1)$	$x(2)$	Output from $x(2)$
$t=0$; input	0.25		0.5	
$t=1$; chaotic update	0.75		1.0	
Adaptive phase	0.5	0.25	1.25→0.25	1.0
$t=2$; chaotic update	1.0		0.75	
Adaptive phase	0.5	0.5	1.25→0.25	1.0
$t=3$; chaotic update	1.0		0.75	
Adaptive phase	0.5	0.5	1.25→0.25	1.0

emitted from the open end. After the transient step of $t=1$, the dynamics reaches the steady state, i.e., for $t=2,3,\ldots,\infty$, the emitted excess is identical and the states of the individual elements before a chaotic update are all at $x(i)=x^*(i)$. In fact, this steady-state configuration of $x(i)=x^*(i)$ before a chaotic update is obtained for all thresholds $x^*\leqslant\frac{3}{4}$ after short transience [10].

Such an array can yield N outputs simultaneously, with each output being encoded by the state of an element in the array, i.e., the ith output is coded in the state of unit i: $x(i)$. Thus this array can process a k-bit domain space (which has 2^k possible inputs) concurrently.

The entire output sequence is obtained from the state of this network after transience. We operate in the threshold

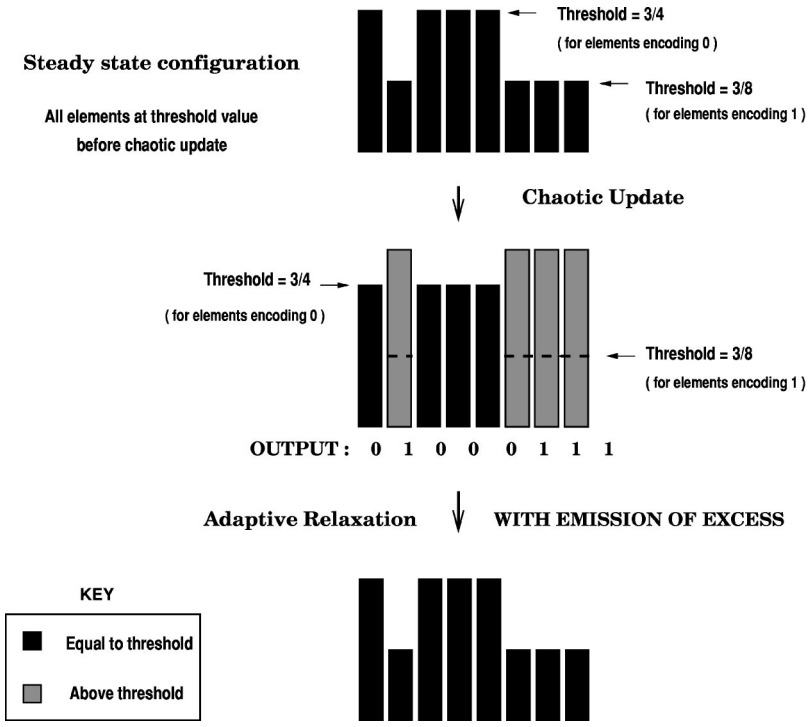

FIG. 4. The DJ problem for the case of $k=3$, i.e., $2^3=8$ inputs, implemented by a network of eight threshold coupled logistic maps, with the state of the ith element encoding the output to the ith input. To implement different balanced functions, the thresholds of the elements in the array are set at $x^*(i)$, $i=1,\ldots,8$, with $x^*(i)$ being $x_0^*=\frac{3}{4}$ for half the elements (randomly chosen) and $x_1^*=\frac{3}{8}$ for the remaining half. Here we depict the implementation of a particular balanced function with output sequence 0, 1, 0, 0, 0, 1, 1, 1. Since inputs 2, 6, 7, 8 give output 1, the elements 2, 6, 7, 8 in the array have threshold set at $x_1^*=\frac{3}{8}$ and the remaining four elements giving output 0 have thresholds at $x_0^*=\frac{3}{4}$. The steady-state configuration just before a chaotic update has all elements at threshold value $x(i)=x^*(i)$. After chaotic update, the value of x of elements 2, 6, 7, 8 is $f(3/8)=15/16>x_1^*$ and so they fire, initiating an avalanche. The elements 1, 3, 4, 5 have states $f(\frac{3}{4})=\frac{3}{4}\leqslant x_0^*$ and so they do not trigger any response. The excess emission from the array at the end of the avalanching process is four units of excess (1 unit$=\frac{15}{16}-\frac{3}{8}=\frac{9}{16}$). So simply by noting that this array emits excess, we can know with certainty that it is analogous to a balanced function, since the array for the constant function (with all thresholds at 1) emits no excess at all at any time.

region $x^* \leqslant 3/4$. For such systems, after transience (which is usually very short) the array is in a steady-state configuration, i.e., the configuration at the end of a chaotic update plus adaptive relaxation (i.e., just before the subsequent chaotic update) is always the same: $x(i)=x^*(i)$ for $i=1,\ldots,N$, where $x^*(i)$ is the threshold for the ith element. In such a configuration, after a chaotic update the elements that are overcritical in the network, i.e., with $f(x^*)>x^*$, encode an output of 1. So the elements that trigger a response, or ''fire,'' in some sense give output 1. The units that are undercritical after the chaotic update [i.e., with $f(x^*)\leqslant x^*$] encode an output of 0 (Fig. 4).

To obtain a constant function we simply have to set threshold $x^*=1$ for all the elements. Since the dynamical values are bounded by 1, no element of this system will then ever be over the threshold at any time. So they will never ''fire,'' and consequently always encode 0. The emitted excess is consequently also identically zero. So no emitted excess from the system is a signature of a constant function.

To obtain a balanced function, we have to set half of the elements (randomly chosen) to have thresholds x_0^* and the other half to have thresholds x_1^*. There are $C_{N/2}^N$ ways of doing this, implementing the different balanced functions. The elements with thresholds x_0^* encode an output of 0, and those with thresholds x_1^* encode 1 ($x_0^*, x_1^* \leqslant 3/4$). As mentioned before, after short transience this array is in a steady configuration; $x(i)=x^*(i)$ for $i=1,\ldots,N$, with $x^*(i)$ being either x_0^* or x_1^*.

Now, to obtain output 0, i.e., to not trigger an adaptive response after chaotic update, we must demand that $f(x^*) \leqslant x^*$. So one can set the threshold to be $x_0^*=3/4$, or $x_0^*=0$. Since $f(x_0^*)=x_0^*$ for the cases of both $x_0^*=3/4$ and $x_0^*=0$ after chaotic update, the elements with thresholds at x_0^* will not be above threshold and consequently cannot start an avalanche [11].

The rest of the $N/2$ elements, encoding output 1, can have a threshold set at some x_1^*, which will generate large excess emission after a chaotic update. For instance, we can choose $x_1^*=3/8$, which maximizes the quantity $\Delta=f(x_1^*)-x_1^*=4x_1^*(1-x_1^*)-x_1^*$, and generates an excess emission of $\Delta=9/16$ from each element. All possible balanced functions can be obtained by suitably choosing the thresholds of the N elements, and all such arrays will yield an excess emission of $(N/2)\Delta$ from the open edge.

If we do not need the individual outputs $f(i)$, but simply have the task of finding out if any one of them is nonzero, we can just examine the excess emission from the open end of the array. This collective excess is exactly equal to the number of input elements in the array yielding an output of 1 (in units of Δ). So for all the $C_{N/2}^N$ distinct balanced functions we will have an excess emission with common value $(N/2)\Delta$ and this will immediately tell us that there are $N/2$ outputs that are 1.

Thus, with one measurement of the collective excess we can deduce with certainty whether or not the network yields any one of the innumerable balanced functions, as all balanced functions will yield an excess of exactly $(N/2)\Delta$ while the constant function yields no excess emission [12].

Even under noise the difference in the collective excess of balanced functions and constant functions is $\sim(N/2)\Delta$. Since Δ is chosen to be large, these responses are clearly different. Thus, deduction of the class of functions under noise is as easily done as in the noise-free case. Such schemes may be extended to closely related extended systems such as models of sandpile and percolation phenomena.

In summary, we have used extended dynamical systems to obtain different strategies for parallelizing the DJ problem. In particular, we have exploited the collective properties of such systems to reduce computational effort. The success of these schemes underscores the potential for problem solving using the parallelism inherent in extended dynamical systems.

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

[2] S. Sinha and W. Ditto, Phys. Rev. E **60**, 363 (1999).

[3] D. Deutsch and R. Jozsa, Proc. R. Soc. London, Ser. A **439**, 553 (1992); R. Cleve *et al.*, *ibid.* **454**, 339 (1998).

[4] E. Ott, *Chaos in Dynamical Systems* (Cambridge University Press, Cambridge, England, 1993).

[5] There are alternate ways of realizing the different balanced functions, of course. For instance, instead of the two basic functions given in Eqs. (6) and (7), one can use combinations of the fixed functions $F_1=0$ and $F_2=1$ to implement the different balanced functions.

[6] The other constant function, giving all outputs to be 1 can be obtained with equal ease. The mean field then will have a value ~ 1. So the difference between the mean fields of this constant function and any balanced function is again $\frac{1}{2}$.

[7] K. Ikeda, Opt. Commun. **30**, 257 (1979); K. Ikeda *et al.*, Phys. Rev. Lett. **45**, 709 (1980).

[8] H. M. Gibbs *et al.*, Phys. Rev. Lett. **46**, 474 (1981); F. A. Hopf *et al.*, Phys. Rev. A **25**, 2172 (1982).

[9] H. Nakatsuka *et al.*, Phys. Rev. Lett. **50**, 109 (1983).

[10] S. Sinha and D. Biswas, Phys. Rev. Lett. **71**, 2010 (1993); S. Sinha, Phys. Rev. E **49**, 4832 (1994); Phys. Lett. A **199**, 365 (1995); Int. J. Mod. Phys. B **9**, 875 (1995).

[11] Note that these units can ''topple'' during an avalanche initiated by other neighboring units, though.

[12] The other constant function, giving all outputs to be 1, can be obtained with equal ease. The collective excess will then have a value $\sim N\Delta$. So the difference between the collective excess of this constant function and any balanced function is again $(N/2)\Delta$.