Adaptation and Self-Repair in Parallel Computing Structures

B. A. Huberman
Xerox Palo Alto Research Center, Palo Alto, California 94304

and

T. Hogg
Physics Department, Stanford University, Stanford, California 94305
(Received 15 December 1983)

This paper reports a study of the dynamics of highly concurrent computing structures capable of learning and storing several input patterns. The existence of attractors in their dynamical behavior leads to a novel self-repairing mechanism. Quantitative experiments on adaptive processor arrays demonstrate the existence of attractive fixed points in their adaptation and recognition properties, as well as the immunity of these collective computation machines to random errors.

PACS numbers: 89.90.+h, 06.50.Mk, 89.70.+c

Complex systems, such as biological organisms and computing structures, lie between the realms of statistical mechanics and the physics of a few degrees of freedom. As such, they lack both the universality of the laws of large numbers and the simplicity implied by the geometry of low-dimensional phase spaces. Moreover, they are capable of exhibiting behavior which in many cases is fault tolerant and characterized by adaptation, learning, and recognition. All these properties underlie problems of current interest in very large-scale integration (VSLI),[1] neurobiology,[2] and cognitive processes such as vision[3] and speech[4] understanding.

Parallel computing structures, which are common in nature, provide an ideal experimental tool when implemented in actual machines. This allows for a detailed analysis of concurrent processes which are not often experimentally accessible in the real world.[5]

This paper reports results of a quantitative study of dynamical behavior in parallel computers which can be implemented with current technology. Their properties include learning, pattern recognition, and associative (content addressable) memory. We present a novel self-repairing mechanism, based on the existence of stable attractors, which is a conceptual departure from the usual multiplexing schemes.[6] We also introduce a tumbling index, Λ, governing the rate of discrete error recovery in these arrays. Finally, we discuss the applicability of our results to a variety of systems.

Consider a synchronous, rectangular array of identical processors each of which is locally connected to its neighbors. In addition to data values received through these connections, each processor

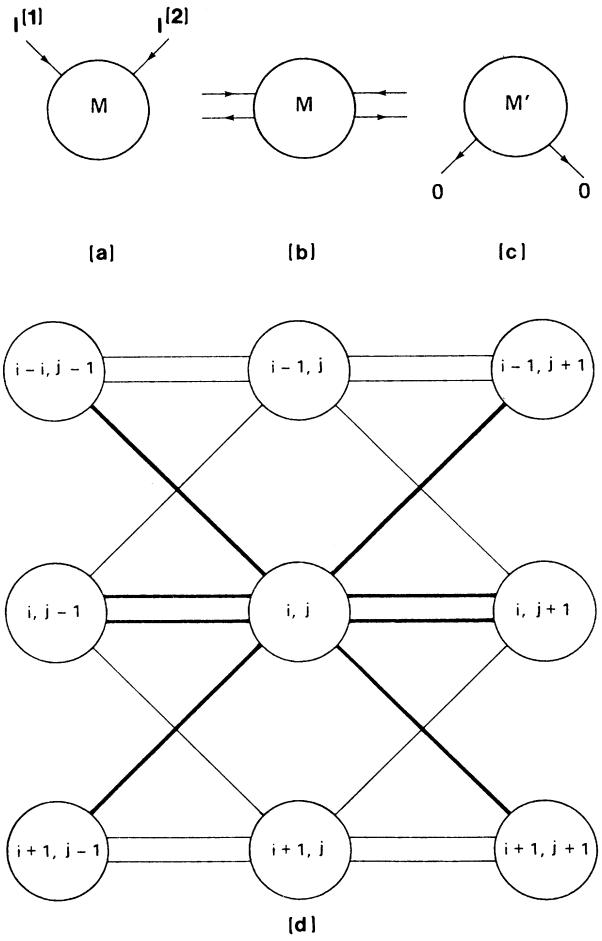

FIG. 1. Computation steps and unit cell of the array. (a) First the two input values are received from above. (b) The new output is then computed and sent to the left and right. (c) The internal memory value is modified and the new output sent down. (d) Unit cell of the array.

has an adjustable internal state, or memory, which allows it to adapt to its local environment. Overall input and output takes place at the edges of the array, with the upper edge of the array for input and the lower edge for output.

Our particular implementation[7] for an array with m rows and n columns provides each element with two integer inputs and a single integer output. All of the data values are constrained to lie in a specific interval, namely $[S_{\min},S_{\max}]$. The internal state of each element is represented by a single small integer. The local computation rules enhance differences in the data values, with a saturation process which keeps the values within the specified interval.

The sequence of local computations is illustrated in Figs. 1(a)–1(c). Specifically, let $M_{ij}(k)$ be the memory value contained in the processor located in the ith row and jth column after the kth time step. Let $I_{ij}^{(1)}(k)$ and $I_{ij}^{(2)}(k)$ be the input values to this element and $O_{ij}(k)$ be the output. The new output of each element is then computed as

$$O_{ij}(k+1)=\max\{S_{\min},\min(S_{\max},M_{ij}(k)[I_{ij}^{(1)}(k)-I_{ij}^{(2)}(k)])\} \tag{1}$$

which, except for the saturation, just multiplies the difference in the inputs by the value stored in the memory. The connections between elements are defined by the relations

$$I_{ij}^{(1)}(k)=O_{i-1,j-1}(k), \tag{2a}$$

$$I_{ij}^{(2)}(k)=O_{i-1,j+1}(k) \tag{2b}$$

for $1\leq i\leq m$ and $1\leq j\leq n$. Thus each element is connected to its neighbors along diagonals and each output is used as an input to two elements (except at the edges of the array). The unit cell of such a lattice is shown in Fig. 1(d). These connections cause the data to flow through the array at a rate of one row per time step. The external input signal to the array at step k is denoted by $\underline{S}(k)$, and $\underline{R}(k)$ is the resulting output vector. The boundaries of the array are specified by

$$O_{0j}(k)=S_j(k), \tag{3a}$$

$$O_{mj}(k)=R_j(k), \tag{3b}$$

$$O_{i0}(k)=O_{i,n+1}(k)=0 \tag{3c}$$

for the top, bottom, and side edges, respectively.

Adaptive behavior is obtained when the memory values M_{ij} are modified by comparing the output, which was previously computed according to Eq. (1) by the i,j element, to the outputs of its neighbors to the left and right, as shown in Figs. 1(b) and 1(c). If the i,j output is greater (smaller) than the other two, the value of M_{ij} is increased (decreased) by one, subject to the restriction that its magnitude remain in a fixed range $[M_{\min},M_{\max}]$. Typically, we used $M_{\min}=1$ and $M_{\max}=4$. This procedure further enhances differences in the inputs to each row of cells. By thus separating the adaptive process from the flow of data, the adapted array can be studied by sending inputs through it without changing the memory values.

To analyze the dynamics of such a complex system, we used the stroboscopic technique of d'Humieres and Huberman,[8] which is illustrated in Fig. 2 for a square connectivity. Other topologies, such as the one we study here with diagonal connections, can be equally treated with this technique. By presenting a periodic string of inputs and computing the distance between corresponding sampled outputs, we were able to study dynamical changes in the array while it adapted to the input sequence. Notice that this technique does not necessarily require direct access to the internal state of the array. Specifically, for periodic signals with period P we have $\underline{S}(k+P)=\underline{S}(k)$ for all k. Since a particular

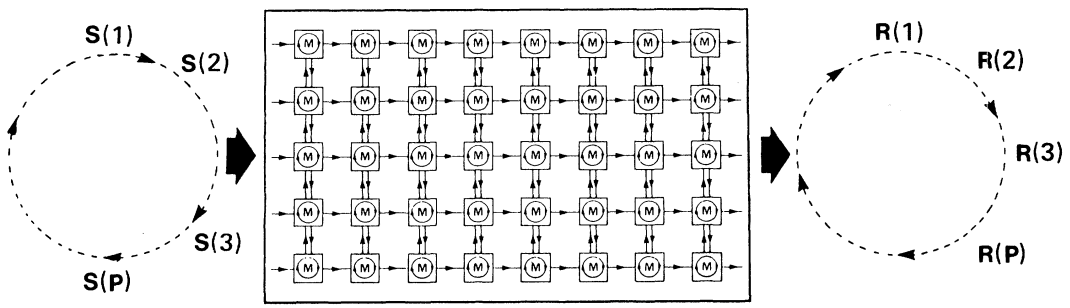

FIG. 2. Schematic description of the stroboscopic sampling of the array dynamics. The square boxes denote the elements of the array with memory M which can differ in each element.

signal requires m time steps to reach the output, the signal vector entering the array at step k, $\underline{S}(k)$, will produce a result at step $m+k$, $\underline{R}(m+k)$. The output distance for each period is defined by

$$d(t) = \max_{\{k\}} ||\underline{R}(m+k+P) - \underline{R}(m+k)||, \quad (4)$$

where k ranges over the set $\{tP+1, tP+2, \ldots, (t+1)P\}$, i.e., the times at which the t th period of the input signal enters the array.

Figure 3 shows the results of an experiment performed on a representative array consisting of 64 cells arranged in a square lattice which was subjected to a periodic, pipelined input consisting of four patterns chosen at random, i.e., $P=4$. Pipelining, besides being computationally efficient (i.e., high concurrency), prevents the array from operating on zeros between inputs. The saturation values were chosen to be $S_{\max} = -S_{\min} = 15$. As shown in Fig. 3(a) up to the location of the arrow, the system reaches a fixed point in a short time (of order five periods). In the adapted regime the stroboscopically sampled state of the array consists of a complicated configuration of memory values which are shown in Fig. 2(b).

These attractive fixed points produce recognition processes which, unlike those in digital filters,[9] are nonlinear because of saturation in the local computation rules. By fixing the values of the memory states and then sending inputs through the array in any order, we were able to determine which learned inputs produced different outputs. These processes map many different input patterns into each output and in general the output of the array is insensitive to small variations in the inputs. Thus, small distortions (as measured by their distance) of the learned inputs still produce the same outputs. Note that by using each output as an address in a data base, the array can be made into a flexible content-addressable memory.

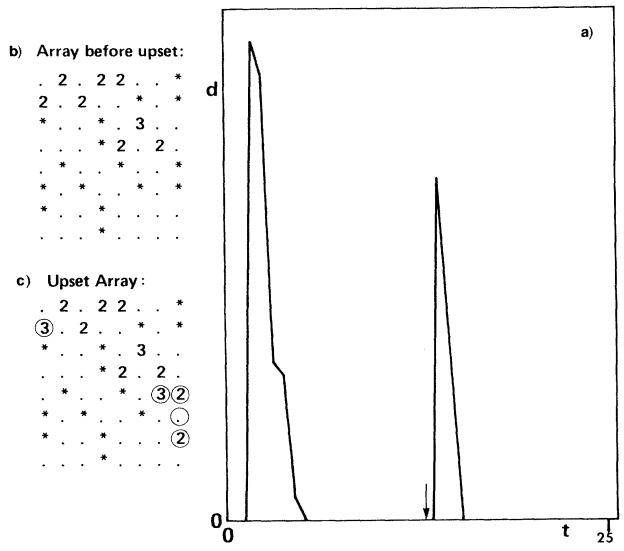

FIG. 3. (a) Distance $d(t)$ as a function of time in units of the input period for a square array with 64 cells. The arrow denotes the time at which errors were introduced. (b) State of the array before upset. The dots denote the value 1 and the asterisks 4. (c) State of the array immediately after five soft upsets.

The existence of stable attractors for these computing structures provides a novel mechanism of self-repair during the adaption process. The dynamics of the array causes small fluctuations in either data or memory values to relax towards the attractive fixed points. This general method of self-repair is very different from the standard techniques used for reliable computing. The latter are based on multiplexing schemes with majority rules and require an enormous amount of hardware per operation.[6] Our technique uses instead the stable collective modes of the computer. Hence the dynamics of this parallel machine is analogous to the behavior of a dissipative dynamical system with many degrees of freedom.

To study this effect, we introduced errors during the adaptive process either while the array was changing or after it converged to a fixed point. Such errors were introduced by randomly modifying some of the memory values and then allowing the array to proceed as before (soft upsets), or by permanently freezing the state of some of the elements as well as their outputs (hard upsets).[10] Figure 3(c) shows the state of the array immediately after five soft errors were introduced, with the circles indicating which elements were changed. The resulting rapid recovery is seen in Fig. 3(a) to the right of the arrow. The state of the array after it recovered was the same as shown in Fig. 3(b). Typically, soft upsets could be introduced in 20% of the memory elements of the array without causing it to relax to another attractor.[11]

This mechanism of self-repair suggests the existence of a tumbling index, Λ, which determined the average rate at which errors at the output change discretely in time. Unlike the familiar Lyapunov exponent, Λ characterizes a walk on an integer lattice. Such an index can be defined as an average of the rates of error decay over all array configurations which produce an output differing from the initial one, weighted by the probabilities with which the configurations occur. Further numerical experiments will be needed to determine

the optimal algorithm for measuring Λ.

In conclusion, adaptive computing networks can be designed which display self-organizing, fault-tolerant behavior with few wires, local computation, and a lattice structure. Such architectures, based on a rectangular lattice of identical processors, can be readily implemented on single VLSI chips. They also provide a fertile ground for studies of dynamics of many degrees of freedom. Furthermore, since these machines can be used to implement standard logic gates they are capable of universal computation. These computing structures can also be used for perceptual tasks such as vision and speech learning and recognition. Last but not least, given the traditional extrapolation of the behavior of automata to the functioning of the brain,[12] our mechanism of self-repair may provide a useful paradigm for studying its reliability.

We have benefitted from stimulating discussions with M. Gell-Mann, A. Perliss, and R. Spinrad. One of us (T.H.) would like to thank the Xerox Palo Alto Research Center for a predoctoral fellowship.

[1]C. A. Mead and L. A. Conway, *Introduction to VLSI Systems* (Addison-Wesley, Reading, Mass., 1980).

[2]J. C. Eccles, *The Understanding of the Brain* (McGraw-Hill, New York, 1973).

[3]M. Brady, Comput. Surv. 14, 3 (1982). See also D. H. Ballard, G. E. Hinton, and T. J. Sejnowski, Nature (London) 306, 21 (1983).

[4]J. Banatre, P. Frison, and P. Wuinton, Acta. Inf. 18, 431 (1983).

[5]See, for instance, *Essays on Cellular Automata,* edited by A. W. Burks (Univ. Of Illinois Press, Champaign, Ill., 1976); S. Wolfram, Rev. Mod. Phys. 55, 601 (1983); M. Y. Choi and B. A. Huberman, Phys. Rev. A 28, 1204 (1983).

[6]K. Brueckner and M. Gell-Mann (unpublished); J. von Neumann, in *Automata Studies,* edited by C. E. Shannon and J. McCarthy, Annals of Mathematics Studies Vol. 34 (Princeton Univ. Press, Princeton, N.J., 1956), pp. 43 ff. See also S. Winograd and J. Cowan, *Reliable Computation in the Presence of Noise* (MIT Press, Cambridge, Mass., 1963), and more recently, R. L. Dobrushin and S. I. Ortyukov, Prob. Inf. Transm. (USSR) 13, 203 (1977).

[7]We should mention that many other local rules produce the general behavior that we report below.

[8]D. d'Humieres and B. A. Huberman, J. Stat. Phys. 34, 361 (1984).

[9]R. W. Hamming, *Digital Filters* (Prentice-Hall, Englewood Cliffs, N.J., 1977).

[10]Error mechanisms in actual microelectronic circuits are discussed by J. McNuthy, Phys. Today 36, No. 1, 9 (1983).

[11]Hard errors also allow for relaxation to the attractor provided they are few in number.

[12]See, for instance, J. von Neumann, *The Computer and the Brain* (Yale Univ. Press, New Haven, Conn., 1958).