TWO NON-STANDARD PARADIGMS FOR COMPUTATION:

ANALOG MACHINES AND CELLULAR AUTOMATA

Kenneth Steiglitz

Dept. of Computer Science
Princeton University
Princeton, New Jersey 08544

1. Introduction

Serious roadblocks have been encountered in several areas of computer application, for example in the solution of intractable (NP-complete) combinatorial problems, or in the simulation of fluid flow. In this talk we will explore two alternatives to the usual kinds of computers, and ask if they provide some hope of ultimately by-passing what appear to be essential difficulties.

Analog computation, the first alternative to standard digital computation was all but abandoned in the early 1960's, but has a long and rich history. (See the textbooks [16,17], for example.) We will use a very general notion of what analog computation means — we will not restrict ourselves to the kind of analog computer that uses operational amplifiers, diodes, and so on. Rather, we will consider any physical system at all as a potentially useful computer, provided only that we can communicate with it. We will then attempt to formulate precisely the following question: Can analog computers solve problems using reasonable (non-exponential) resources that digital computers cannot?

The study of this question leads us to formulate a strong version of Church's Thesis: That any finite physical system can be simulated *efficiently* by a digital computer. By "efficiently" we mean in time polynomial in some measure of the size of the physical system. This thesis provides a link between computational complexity theory and the physical world. If we grant that P $\neq$ NP, and Strong Church's Thesis, we can then conclude that no physical device solving an NP-complete problem can do so efficiently. We will propose such a device and explore the physical implications of our argument. While we will be able to make certain statements, the important questions in this field are unresolved, especially those dealing with quantum-mechanical systems.

A *cellular automaton* (CA) is in general an n-dimensional array of cells, together with a fixed, local rule for recomputing the value associated with each cell. CA's were originally proposed by von Neumann as a mathematical model to study self-replication. More recently, they have received attention as possible

J. K. Skwirzynski (ed.), Performance Limits in Communication Theory and Practice, 173–192.

models of general nonlinear phenomena, and have been used for nonlinear image processing in the biomedical and pattern recognition fields [3]. We will mention recent work on the use of CA's to model fluid flow.

Cellular automata offer an ideal vehicle for studying highly parallel, pipelined computation structures, such as systolic arrays. We will describe the design and testing of a custom VLSI chip for implementing a one-dimensional, fixed CA, which has a total throughput of more than 10^8 updates per second per chip [4]. We will then discuss the processor/memory bandwidth problems that arise in higher dimensional cases, which are important in fluid dynamics.

We will then describe certain one-dimensional automata that support persistent structures with soliton-like properties — a phenomenon qualitatively similar to those observed in certain nonlinear differential equations. This suggests ways to embed computation in homogeneous computing media, and leads us to speculate about how we can overcome the limits of lithography in large-scale integration.

2. Measuring Complexity

We begin by reviewing the usual way of measuring complexity for digital computer algorithms, and then discuss the extension of these ideas to the analog world.

A very effective and robust model for digital computation is the *Turing Machine*, which is characterized by the following components:

a) A tape with cells that can hold symbols from a *finite* alphabet. Without changing the essential features of our model, we take this to be the *binary* alphabet, the symbols {0, 1}. An unlimited supply of tape is available, but only a finite number of cells is ever used.

b) A head that moves on the tape. The head sees the symbol at its current position.

c) The machine is at any time in one of a *finite* number of states.

d) A *finite* set of rules, which play the role of the computer program. Given the symbol under the head and the current state of the machine, these rules determine at each time step what the new tape symbol shall be, whether the head then moves left, right, or remains stationary, and what the new state shall be. Of course time is discrete, and each application of the rules is counted as one time step.

A critical feature of this model is the fact that the sets of symbols, of states, and of rules, are all *finite*. This is in sharp contrast with the usual models for analog systems, which are usually differential equations, and which have a continuous time parameter, as well as continuous state variables.

A finite number L of cells at the beginning of the tape are initially set, and these correspond to the program input. The number L is then taken to be the *size* of the input, measured in *bits*. We will measure the input size to an analog

device in the same way, as a sequence of bits of length L. It is important to notice at this point that an input of length L can represent a number as large as 2^L.

In the case of a Turing Machine, the only resources used by a computation are time and space, the former being the number of discrete time steps, and the latter being the number of tape cells required by a given computation. In the analog case, the situation is not so clear. There is certainly the time taken for the computation, and the volume occupied by the device. But there is also the energy used, the maximum force employed, the largest electric field, and so on; in other words: any physical quantity that might have a real cost associated with it, and which might be bounded by physical constraints. We call these quantities, collectively, the *resources* used by the analog computer. We can then speak of a computation by an analog device as taking *polynomial* resources if the resources are bounded by a polynomial function of the input description length L, and similarly for computations that take *exponential* resources. (We usually use the term *exponential* to mean *not polynomial.*)

3. An Analog Computer that uses Polynomial Resources

We now give an example of a simple computation that can be performed by an analog computer using polynomial resources.

The Maze Problem: A graph G is defined on an $n \times n$ array of nodes, and each node can be connected only to the nodes adjacent to it in a column or row. The problem is to determine if there is a path from the upper-left corner node s to the lower-right corner node t. (See Fig. 1a.)

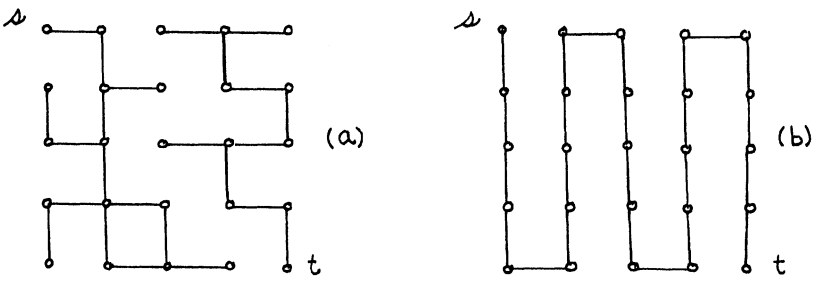

Fig. 1 a) The maze problem; b) A longest s–t path.

This is not a hard problem for a digital computer; in fact, it can be solved in a number of steps proportional to the number of edges. A natural way to approach this problem with an analog computer is to build an electrical network that has a terminal for each graph node, and a wire wherever an edge appears in the graph. We can then apply a voltage source across the terminals s, t, and measure the

resulting current flow. The terminals s and t are connected if and only if there is a "positive" current flow.

We need to more precise about what constitutes a positive current, about how long we need to wait to make our decision, and about how expensive the network is. Assume for this purpose that each edge is represented by a wire of rectangular cross-section with width w, length d, and height h, all of which can be a function of n, provided that w doesn't grow faster than d.

Assume first that there is a path from s to t. The longest such path can have $O(n^2)$ edges, as shown in Fig. 1b. The resistance of each wire is proportional to its length, and inversely proportional to its cross-sectional area, so that the total resistance between s and t is

$$R_{cl} \leq k_1 \frac{n^2 \cdot d}{wh} \tag{1}$$

where k_1 is some constant. Notice that if d, w, and h are held constant, the closed-circuit resistance grows as the square of n, or linearly with the number of nodes.

Next, consider what happens when there is no s–t path. The open-circuit (leakage) resistance R_{op} is proportional to d, inversely proportional to the area hd, and in the case leading to the lowest resistance, our worst case, inversely proportional to n^2. This is because there can be in effect no more than n^2 wires in parallel across an s–t cut. We thus have the asymptotic lower bound

$$R_{op} \geq k_2 \frac{1}{n^2 h} \tag{2}$$

where k_2 is another constant, but much larger than k_1.

Combining these two inequalities, we can find an asymptotic lower bound on the ratio between the closed- and open-circuit current:

$$I_{cl}/I_{op} = R_{op}/R_{cl} \geq (k_2/k_1)\frac{w/d}{n^4} \tag{3}$$

If we plan to distinguish successfully between the closed- and open-circuit cases, we need to ensure that this ratio is large enough to make possible reliable measurement. We see from this analysis that the scaling ratio $w/d < 1$ cannot prevent ultimate disaster as n grows indefinitely. What makes this circuit work in a practical situation is the fact that

$$k_2/k_1 \gg 1 \tag{4}$$

Thus, the device will work effectively for a large range of n, as we might expect.

However, we also see from this analysis that there is a limit on the size of problem, determined by the technology, beyond which this device simply will not function. It appears that this is an unavoidable limitation in all physical devices that we construct to solve arbitrarily large instances of problems. Suppose, for example, that we try to use water, and model the edges of the maze by pipes. We then need to worry about making the pipes heavy enough to support water pressure that grows indefinitely. If we use microwaves, we need to worry about loss, and so on. But we can accept the fact that there is a large regime in which a device will operate well; the same is true of any real digital computer, even for polynomial-time problems.

Next, consider how long we need to wait (asymptotically) before we can make a reliable decision. This is determined by the RC time constant in the closed-circuit case. The greatest possible capacitance across the terminals of the voltage source can be no worse than proportional to the total surface area between wires, or $O(n^2dh)$, and inversely proportional to the inter-wire distance d, assuming w grows slower than d, for a total capacitance that is $O(n^2h)$. The largest total resistance is $O(n^2d/(wh))$, by (1). The time constant is therefore

$$RC = O(n^4d/w) \tag{5}$$

Thus, letting w grow as d, we find this "analog computer" takes time $O(n^4)$, proportional to the square of the number of nodes. We can also check that the total mass of the machine, and the power consumption, are also polynomial, provided we are in the regime of operation provided by the technology constants k_1 and k_2.

4. Analog Machines that use Exponential Resources

It is all too easy to construct analog computations that require exponential resources, simply because of our previous observation that L bits in the input can encode a number as large as 2^L. An example is provided by A. K. Dewdney's "Spaghetti Analog Gadget" (SAG) [22], for sorting integers, a polynomial-time problem for a digital computer. Instructions for this analog computation follow:

Spaghetti Analog Gadget

1) Given a set of integers a_i to be sorted, cut a piece of (uncooked) spaghetti to the length of each of the numbers.

2) Assemble the pieces in a bundle, and slam them against a flat surface, so that all the pieces are flush at one end.

3) Remove the pieces that extend farthest at the other end, one at a time. The order in which the pieces are removed from the bundle sorts the integers from largest to smallest.

The difficulty here is that the pieces are cut to lengths proportional to the integers in the input data, but these are encoded in binary. The pieces of spaghetti code the numbers in unary, instead of binary. Thus, the machine uses an exponential amount of spaghetti.

We might try to circumvent this problem by using a logarithmic scale to size the spaghetti. This will keep the mass of the machine polynomial, but raises a new problem. We will then need to distinguish between ends of pieces that are exponentially close in size. In the presence of fixed measurement noise, this will require an exponential scaling to keep the measurements reliable, again leading to exponential mass. As proved in [1], there is no way around this difficulty if we insist on representing input numbers by single analog quantities. The digital computer can represent a single number by a (theoretically) unlimited number of registers, and that seems to be an essential difference between digital and analog computation, and the source of the term "analog(ue)".

We now take up the question of whether an analog computer can solve a problem that is intractable for digital computers. To this end we will consider NP-complete problems, problems as difficult as any in the wide class of NP problems, and widely considered to require exponential time on digital computers. (This belief is usually expressed in the conjecture that $P \neq NP$.)

5. Strong Church's Thesis

We now need to emphasize an important distinction between statements about Turing Machines, or digital computers in general, and analog computers. In the former case, our statements can be made and proven with mathematical precision — they are, in fact, mathematical statements. When discussing analog devices, however, we are making statements about what will happen in the physical world, and the correctness of our conclusions depends on the models we choose. For example, we choose to describe the electrical gadget for the Maze Problem by the usual electrical models, including Ohm's law. Our prediction of its speed of operation depends on a differential equation model.

Thus, there is always room to refine our model of analog computers, and it is never guaranteed that predictions of behavior will be correct. For example, we ignored the inductance in the circuit and its effect on its time-domain behavior; we assumed that the noise environment does not interfere with the measurement of current as the size of the device grows to infinity; and so on.

To carry over the ideas of complexity theory from the mathematical realm to the physical, we need something that will relate the two worlds in some way. Such a connection between mathematics and the informal world is provided by Church's Thesis, also called the Church-Turing Thesis [12,13]. The usual way of stating it is that the Turing Machine completely captures the notion of computation; that is, that anything that can be computed at all can be computed by the Turing Machine. In our context, we can view this as stating that the Turing Machine can compute anything that an analog device can.

We now go one step further, and formulate a stronger version of Church's Thesis that relates complexity in the two domains. (A similar idea has been proposed by R. Feynman [14].)

Strong Church's Thesis (SCT) An analog device can be simulated by a Turing Machine in time that is a polynomial function of the resources it uses.

What this says, essentially, is that any piece of the physical world can be simulated by a digital computer without requiring time exponential in any measure of the size of the piece. SCT is not susceptible to mathematical proof; it is in fact a statement about physics that may or may not be true, as is the usual Church's Thesis. Only by postulating a particular model for a piece of the physical world could one hope to prove it. For example, it is proved for systems described by a certain class of differential equations in [1].

We are going to consider the possibility of using analog computers to solve NP-complete problems, which are generally considered to be intractable in the sense of requiring exponential time. Furthermore, all the members of the class are equivalent to each other in the sense that if one can be solved in polynomial time, they all can. Thus, either they are all in P (P = NP), or none is (P $\neq$ NP). The reader can find more on this subject in [15].

SCT now allows us to make the following kind of argument. Suppose an analog device solves an NP-complete problem using a polynomial amount of resources. Then the fact that we can simulate it in polynomial time with a Turing Machine means that there is also a Turing Machine that solves the problem in polynomial time, and so P = NP. If we then take as postulates P $\neq$ NP and SCT, we have a metamathematical argument that the given device cannot operate using polynomial resources. We will study such a device in the next section.

6. An Analog Machine for an NP-complete Problem

The NP-complete problem we will solve is called PARTITION [15]:

PARTITION Given a set of positive integers $S = \{\omega_1, \omega_2, \ldots, \omega_n\}$, is there a nonempty subset S' of S with the property that the elements in S' sum to exactly $(1/2)\sum_{i=1}^{n} \omega_i$? That is, does there exist an S' that partitions S into two equal-weight subsets, those in S', and those not?

The analog device we will construct will represent the integer ω_i by a cosine signal with frequency ω_i . We observe first that by using the fundamental operations of addition, subtraction, and squaring of signals, we can form the product of two cosine waves, which also consists of the sum of cosine waves at the sum and difference frequencies. This follows from the following identities:

$$\begin{aligned} 4\cos x \cos y &= 2[\cos(x+y) + \cos(x-y)] \\ &= (\cos x + \cos y)^2 - (\cos x - \cos y)^2 \end{aligned} \tag{6}$$

From this we see that we can synthesize a signal at the frequency ω with $O(\log\omega)$ such operations, and so the synthesis of these signals requires only a polynomial amount of equipment.

In the same way we can construct the signal

$$\tfrac{1}{2}\prod_{i=1}^{n}\cos\omega_i \;=\; (2^{-n})\sum\cos(\omega_1+\sum_{i=2}^{n}\pm\omega_i) \tag{7}$$

where the outer sum is over all 2^{n-1} combinations of + and − in the inner sum. We can then determine if one of these sums is zero by integrating over one period, from $t = 0$ to 2π. In fact, that integral will be zero if and only if one of the inner sums in (7) is zero, and this is equivalent to there being a solution to the partition problem on the numbers $\{\omega_i\}$. Thus we have shown that the following problem is NP-complete, a result due to Plaisted [15, 18]:

COSINE PRODUCT INTEGRATION Given a set of positive integers $S = \{\omega_1, \omega_2, \ldots, \omega_n\}$, does

$$\frac{1}{2\pi}\int_0^{2\pi}\tfrac{1}{2}\Big(\prod_{i=1}^{n}\cos(\omega_i t)\Big)dt \;=\; 0\;? \tag{8}$$

Figure 2 shows a sketch of a device that will make this decision, using adders, subtracters, squarers, an integrator, and a threshold detector. While we have tried to make the operation of this PARTITION machine as practical-sounding as possible, the main point of the preceding discussion is that it will not work in practice, in the sense of requiring resources exponential in the length of the input data. Where does the machine founder? Perhaps providing enough bandwidth for the integrator is the problem. Perhaps noise will obscure the distinction between zero and not zero. Perhaps the accuracy requirements on the square-law device are prohibitive. As satisfying as it might be to find the flaw in its operation, it is not necessary to analyze the device; if we accept that P $\neq$ NP and Strong Church's Thesis, it cannot work efficiently in practice.

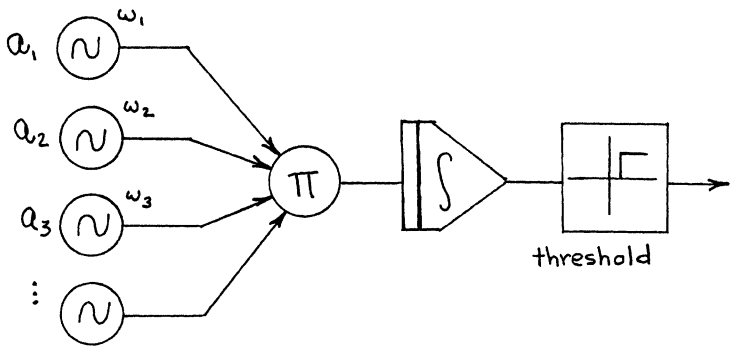

Fig. 2 The PARTITION machine.

In fact, however, it is not hard to locate one flaw that would become evident if we tried to solve large instances of PARTITION with such a machine. Each distinct partition of the ω_i contributes a zero-frequency term to the integrand in (8), and that results in an integral with value 2^{-n}, from (7). Therefore, if there is only one such term we must distinguish between the values 0 and 2^{-n}. If there is a fixed level of noise, we would then need to scale by an exponential factor to accomplish this discrimination. In other words, the machine proposed operates with an exponentially small signal-to-noise ratio. Somehow, it seems that P $\neq$ NP and SCT together imply that there is an irreducible level of noise in the world.

In [1] another analog machine for an NP-complete problem is described. A device constructed with gears, levers, and cam-followers is constructed that ostensibly solves the NP-complete Boolean Satisfiability problem. (Actually, the special case called 3-SAT.) This problem does not have integers in its input description, and the flaw in the operation of the machine is perhaps less obvious than for our PARTITION machine. The link that Strong Church's Thesis provides between the mathematical and physical worlds is strong enough to allow us to make nontrivial statements about certain physical systems.

7. Cellular Automata

We next turn our attention to another way of thinking about computation, *cellular automata.* In one sense this subject does not present the kind of fundamental difficulties we encountered when we studied analog computation. There is nothing that can happen in a cellular automaton that, in principle, cannot be simulated by Turing Machine. Rather, the interest lies in the organization of the computation. A cellular automaton can reflect the way that computations are performed in nature, and so can be a more natural framework than a conventional serial computer for studying certain phenomena — the kinds of phenomena that occur in systems with a large number of identical, locally interacting components. The important features of a cellular automaton are the *uniformity* and *locality* of its computations.

A cellular automaton has three components:

a) A discrete, finite *state space* Σ containing the value 0, usually the additive group mod k on the integers $\{0, 1, 2, \cdots, k\text{-}1\}$. Often $k = 2$, in which case we say the automaton is *binary.*

b) A set of *sites* or *cells* called the *domain* Δ, usually a discrete regular lattice of points in n dimensions, often simply equally spaced points on a line or circle, or a rectangular or hexagonal grid in the plane. Each site in the domain has a value in Σ associated with it, and we refer collectively to the domain and the state values associated with each site as the *state* of the automaton. We also refer to n as the dimension of the automaton.

c) A *rule* Φ which, given the state at discrete time t, yields the state at time $t+1$. The rule uses as arguments the values in some fixed neighborhood of each site.

Thus, the rule Φ can be thought of as "re-writing" the values at each of the sites, for times $t = 1, 2, 3, \ldots$, given an initial state at $t = 0$. We will always assume that the initial state has only a finite number of sites with non-zero values, and so can be finitely described.

The cellular automaton was invented by von Neumann to study self-replication [19]. Today, there are two main reasons for studying the model: first, as a model for physical phenomena; and second, as a model for computation in regular networks, such as neural networks. We will naturally concentrate here on the latter category, but we should mention some recent, exciting work in the first category, the application of cellular automata to fluid dynamics, as an alternative to the numerical solution of the Navier-Stokes equation [23,24].

The idea behind the fluid dynamic machines is to model the behavior of a fluid by a collection of particles which can move around a lattice, with fixed rules for what happens at collisions. Under certain circumstances, and with the right rules, it can be shown that the average velocity fields obtained do in fact coincide with solutions to the Navier-Stokes equation [25]. So far, the most successful computations use a hexagonal grid in two dimensions, and each site is assumed to have up to 6 particles, possibly one traveling towards each of the 6 neighbors of the site, and possibly a particle at rest at the site. Thus, 7 bits suffice to describe the state at a site. The rules need to be carefully designed so that momentum and mass are conserved. When the automaton is started from an initial random state with a certain population density, numerical velocity fields are obtained by averaging over blocks of sites, typically 48×48 sites square. Results so far illustrate the development of vortex streets, and other qualitative phenomena associated with turbulent flow, for low effective Reynolds Numbers.

Whether this approach is ultimately of practical importance in fluid dynamics depends on whether the calculations required can be made highly parallel. This is one way in which the cellular automaton model shines; its great regularity and simplicity invite deep pipelining and custom VLSI implementations, and the parallelism obtained this way may more than compensate for the primitive nature of the individual computations. This is especially true if one-dimensional pipelining can be used. To illustrate the way in which one-dimensional computation can be pipelined to an arbitrary depth, we will describe briefly a custom VLSI chip that was designed and tested at Princeton.

8. Cellular Automata: Notation and an Example

We first set down some notation to describe cellular automata. Let a_i^t be the state value at time t and position i, $-\infty \leq i \leq +\infty$, and $0 \leq t \leq +\infty$. The next-state rule is of the form

$$a_i^{t+1} = \Phi(a_{i-r}^t, a_{i-r+1}^t, \ldots, a_i^t, \ldots, a_{i+r}^t) \tag{9}$$

We also usually assume that Φ preserves zero,

$$\Phi(0,0, \ldots, 0) = 0$$

The next value of site i is a function of the previous values in a neighborhood of size $2r + 1$ that extends from $i - r$ to $i + r$. Given initial states at all the sites, repeated application of the rule Φ determines the time evolution of the automaton.

As an example, we will describe a particular binary, one-dimensional automaton with $r = 2$, denoted by Wolfram [2] the rule-20 totalistic cellular automaton. This is perhaps the simplest automaton that exhibits very complicated behavior, and may in fact be universal in the sense of having the power of a Turing machine. The rule is called *totalistic* by Wolfram [2], because the next-state function Φ depends only on the sum of its arguments. Let that sum at time t and position i be denoted by S_i^t,

$$S_i^t = \sum_{i-r}^{i+r} a_i^t \tag{10}$$

The rule is the following with $r = 2$,

$$a_i^{t+1} = \begin{cases} 1 & S_i^t \text{ even but not } 0 \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

We will call any rule of this form a *parity rule*. It is called *rule-20* because the binary expansion of 20 is 10100, and this string determines which values of S_i^t yield an output value of 1, by the 1's at bits 4 and 2, counting from the right and starting with 0.

Figure 3 shows the evolution of this automaton, and illustrates an interesting feature, namely the presence of persistent structures that we will call *particles*. The figure shows two examples of particles of different speeds colliding destructively. Particles are quite rare in this automaton, and almost all collisions are destructive. Later on we will describe other automata in which particles are much more common, and collide non-destructively.

Based on extensive experimental evidence, Wolfram [2] has classified the behavior of cellular automata into four types:

1) evolution to a homogeneous state (analogous to a *limit point* in nonlinear differential equations),
2) evolution to separated stable or periodic structures (analogous to *limit cycles*),
3) evolution to chaos, and
4) evolution to complex, often long-lived structures.

The last type of behavior, exhibited by the rule-20 cellular automaton, will be most interesting to us, because it will suggest ways in which computation can be embedded into the operation of the automaton.

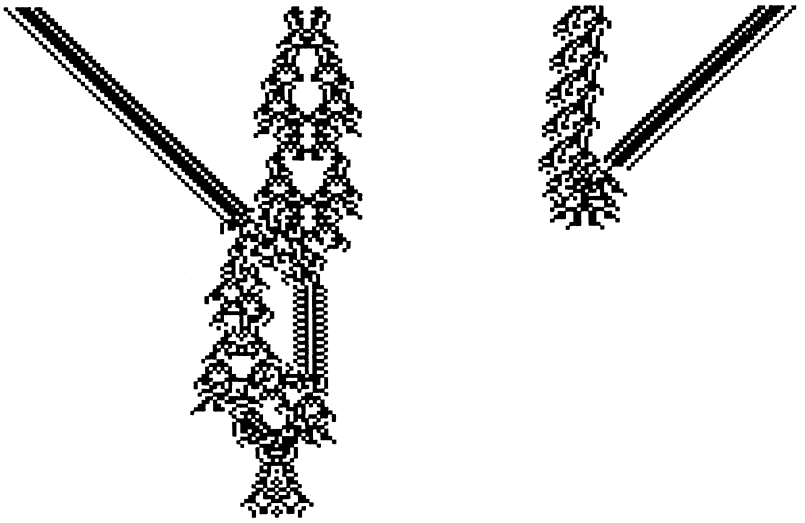

Fig. 3 Destructive collisions in the code-20 cellular automaton.

9. A Custom Systolic Chip for a Cellular Automaton

A chip was designed to implement the rule-20 cellular automata described above. The implementation is interesting from the point of view of the kinds of systolic arrays used to achieve highly parallel computation for signal processing, and we will describe it here briefly.

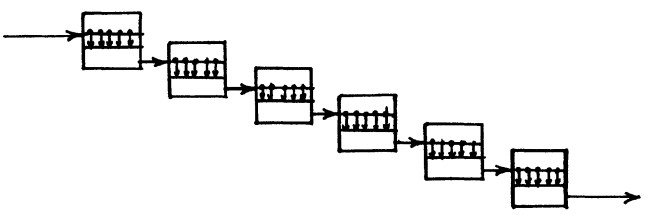

Fig. 4 Bit-serial systolic array for a cellular automaton.

Figure 4 shows how a simple concatenation of bit-serial processors can be used to perform the updating operation. Each processor consists of a $(2r+1)$-bit shift register which holds the argument bits, and fixed combinatorial logic which performs the calculation of the next-state function Φ. The operation is synchronous, and each major clock cycle a new output bit is produced by each processor, and the contents of the shift registers advance one cell.

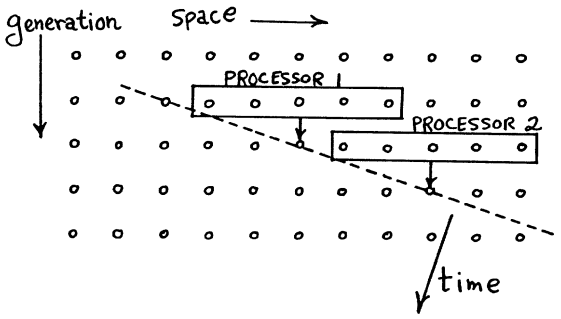

Fig. 5. The computational wavefront.

At any given moment, each processor is fully occupied, and so the array achieves maximum parallelism. Note, however, that it is also true that at any given moment, each processor is working on the generation after the one of the preceding processor, as shown in Fig. 5. Thus, the calculation proceeds along a northwest-to-southeast wavefront. In an actual implementation, the data is circulated through some number N of processors, and each complete circulation results in an update of the one-dimensional array by N generations.

The fixed processor for rule-20 was implemented in 4 μ nMOS, using a 5-bit static shift register and a PLA for the update function Φ. The small chip of 3.1×3.8 μ holds 18 such processors, in 3 columns of 6 processors each, plus the inter-processor wiring and pads. Of the 16 chips that were returned from the MOSIS fabrication facility, 9 were fully functional at speeds of at least 6 Mhz. This implies an effective computational rate of greater than 10^8 site-updates per second per chip, and illustrates the power of the bit-serial systolic approach to one-dimensional computation in such automata.

If this idea is extended to two-dimensional automata, it requires local storage of two full rows of state values at each processor, as opposed to $(2r+1)$ site values in the one-dimensional case. This fact means that many fewer processors can be fit on a single chip, and thus reduces the amount of effective parallelism that can be achieved on a chip. The approach is still very useful in two-dimensional applications such as image processing [26], and is now being studied at Princeton for fluid dynamics applications [27].

10. Parity-Rule Filter Automata

We will now discuss a slightly different kind of automaton, called *Filter Automata* [5]. The original motivation for studying them came from digital signal processing [7]. Noticing that the usual kind of cellular automaton corresponds in its data access pattern to a finite-impulse-response (FIR) digital filter, it is natural to see what happens when the corresponding infinite-impulse-response (IIR) pattern is used. This corresponds to replacing the update rule (9) by one that scans left-to-right, and uses new state values as they become available,

$$a_i^{t+1} = \Phi(a_{i-r}^{t+1}, a_{i-r+1}^{t+1}, \ldots, a_i^t, \ldots, a_{i+r}^t) \tag{12}$$

Some experimentation then shows that these filter automata with the parity rule (11) are especially interesting for a number of reasons. Naturally, we call these the *parity-rule filter automata*, parameterized by the neighborhood width r.

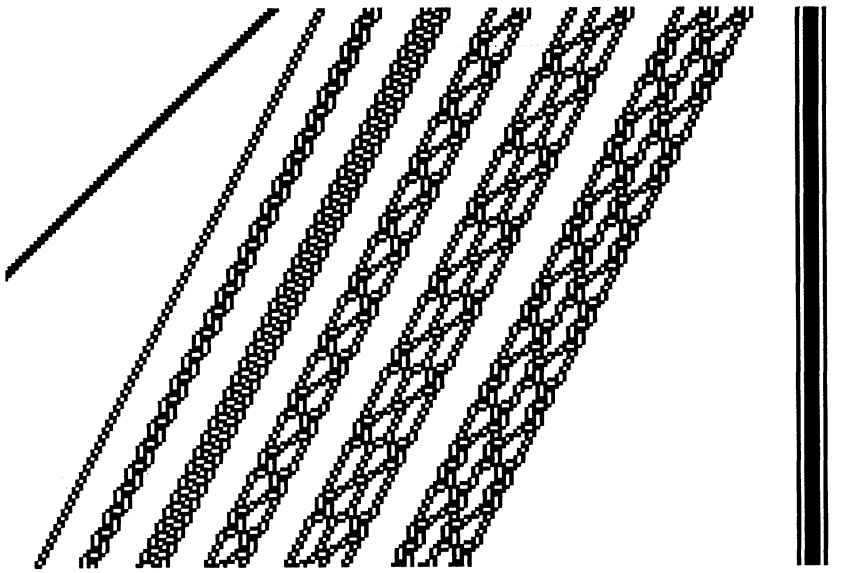

Fig. 6 Some particles in the $r = 2$ parity-rule filter automaton.

The first interesting thing to observe is that these automata support a very wide variety of particles. Figure 6 shows just a few of these for the $r = 2$ parity-rule filter automaton. For each r, there is a unique zero-speed particle (shown rightmost), and all the other particles move to the left. There is also a unique fastest particle, called the *photon*, which consists of $(r+1)$ consecutive 1's, moving

to the left at a speed of $(r-1)$.

To classify a particle, we interpret the bit strings in its (periodic) evolution as binary numbers, and define the smallest such number as the *canonical code* of the particle. Obviously, this is a unique way of identifying a particle. We also refer to the number of generations before repetition as the particle's *period*, the number of sites moved left in a period as its *displacement*, and the ratio of displacement to period as its *speed*.

The number of distinct particles grows rapidly with r. An exhaustive search program yields the following numbers of particles with canonical code width up to and including 16:

radius	number of particles
2	8
3	198
4	682
5	6534
6	13109

These particles come with a variety of displacement/period pairs.

The next interesting thing about the parity-rule filter automata is that particles pass through each other, sometimes with their identities preserved, and sometimes not. Figure 7a illustrates some identity-preserving collisions for the $r = 3$ automaton, while Fig. 7b shows collisions that don't preserve identity, for the same automaton.

Close inspection of Fig. 7b shows a collision in which two particles come together and interact in such a way as to produce two different particles traveling in parallel (the second and third particles from the left at the bottom). This shows that the parity-rule filter automata are in general not reversible in time.

At this point the similarity of these particles with the solitons supported by differential equations [10,11] becomes apparent. These "solitary waves" occur in nonlinear systems, and are characterized chiefly by the non-destructive nature of their collisions. The close connection with the solitons supported by the nonlinear lattices of Hirota and Suzuki [11] is being studied at Princeton with Nayeem Islam [21].

The particles illustrated in Fig. 6 that appear to be composed of several resonating particles traveling in parallel also have their counterpart in physical solitons, where they are called "puffers".

Many further properties of the parity-rule filter automata have been proved by C. H. Goldberg, in a forthcoming paper [28]. For example, he shows that they are inherently stable, in the sense that an initial state with a finite number of 1's always leads to states that also have a finite number of 1's.

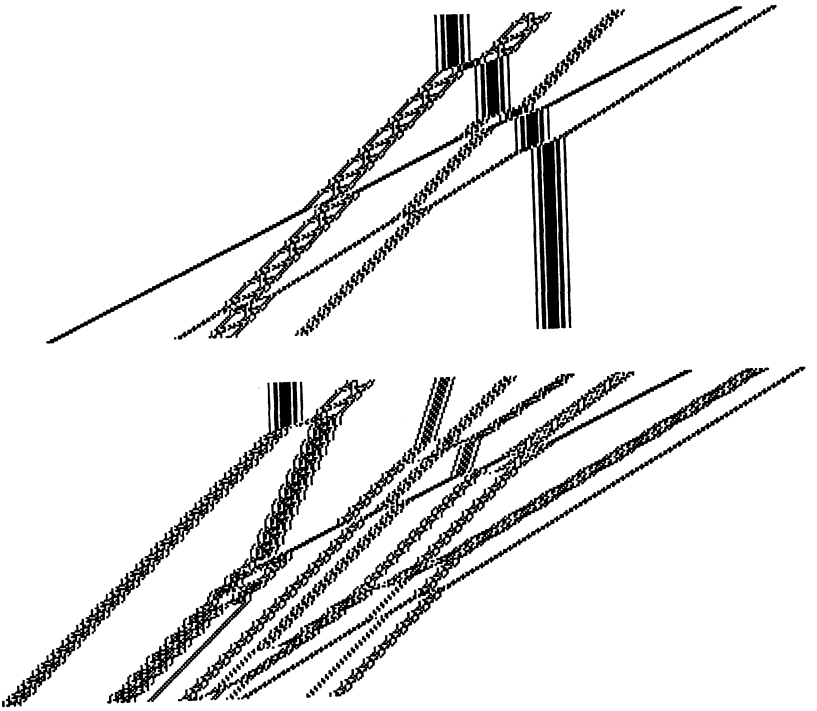

Fig. 7 a) Identity-preserving collisions in the $r = 3$ parity-rule filter automaton; b) Identity-destroying collisions in the same automaton.

11. Phase Shifts and Embedded Computation

We now discuss the possibility of encoding information in the particles supported by these filter automata, and of processing this information by way of the collisions.

As in physical solitons, a collision produces a shift in the trajectory of each particle, which we can think of as a translational phase shift. It always happens that when a fast particle catches up with and passes through a slow particle, the slow particle is thrust backward from where it would otherwise be, and the fast particle propelled forward. This can be verified for the examples shown in Fig. 7. (Incidentally, physical solitons behave in exactly the same way.) We can thus think of the position of a particle relative to some constant-speed reference as representing information that is changed on collision, the *translational* phase of the particle. As a simple example, we might encode a 0/1 bit in the translational phase according to whether its position relative to a reference is even/odd.

There is another kind of phase inherent in the motion of a particle — the relative state of the particle in its periodic path through different configurations. This, too, can be changed by collision, and we call these *orbital* phase changes, as opposed to the translational phase changes. If we liken the motion of the particle

to that of a wave packet, we can think of the translational and orbital phases as those of the envelope and carrier, respectively. In what follows, we will refer to the composite translational/orbital phase as simply the *phase* of a particle.

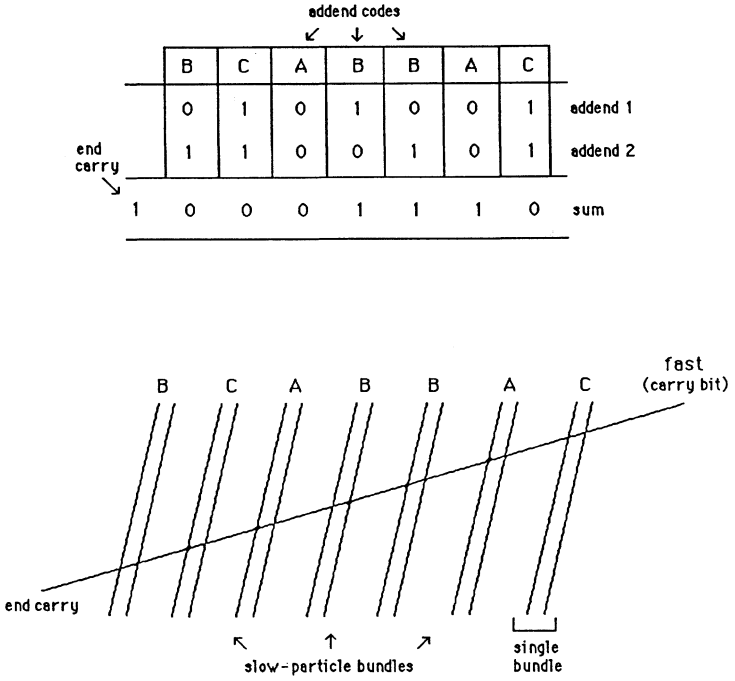

Fig 8. A carry-ripple adder embedded in an $r = 5$ parity-rule filter automaton.

A simple example of a non-trivial computation that can be performed by colliding solitons is described in [20], and we summarize the idea here. The computation is that of a carry-ripple adder, and is illustrated in Fig. 8. The carry bit is represented by the phase of a fast particle entering the scene from the right; the addends are represented by pairs of slow particles. Solutions of this form were found for $r = 5$. Another solution was found for $r = 3$ with 3 particles per addend pair.

Some work is involved in making such a scheme work. We need to make tables of the phase changes that result from collisions, and then we need to search for combinations of particles that behave in the way we want. This can be automated as a search for paths in a constructed graph, and the details are in [20]. We can think of this part of the problem as "programming" the embedded computation.

The idea of using persistent moving structures to do computation is not new; for example, Conway's game of Life [8] is able to mimic the operation of a general purpose computer, and so is capable of universal computation. Similar embeddings are described in [9]. There are two important differences here: first, we are dealing with a one-dimensional "computer", and second, as we have seen, we do not need to initialize the states of the automaton to reflect a logical organization, such as gates, registers, etc. The computation can be encoded very naturally in the serial input stream.

On the other hand, our present knowledge of the power of such computation is very limited, and it is not known if the parity-rule filter automata are universal. We do not yet know much about how to program such machines.

An important aspect of this notion of computation is that a medium can be used to compute, without tailoring the medium to the particular computation, either by designing its "wiring", or by initializing its states. If it were possible to synthesize such a medium at the molecular level [6], we would avoid the problems posed by the limits of lithography, and could have a computer that functioned at that level. The approach also suggests analogous computation with physical solitons [21], and, in a more speculative vein, possible explanations of some kinds of brain function.

Acknowledgements

This work was supported directly by NSF Grant ECS-8414674 and U. S. Army Research Office-Durham Grant DAAG29-85-K-0191; and by computing equipment made possible by DARPA Contract N00014-82-K-0549 and ONR Grant N00014-83-K-0275.

Much of this work was done in collaboration with others, and reported in referenced papers. I am indebted to my co-authors, Brad Dickinson, Nayeem Islam, Irfan Kamal, James Park, Bill Thurston, Tassos Vergis, and Arthur Watson. I also thank the following for helpful discussions: Kee Dewdney, Jack Gelfand, Charles Goldberg, Andrea LaPaugh, Martin Kruskal, Steven Kugelmass, James Milch, Bob Tarjan, Doug West, and Stephen Wolfram.

References

1. A. Vergis, K. Steiglitz, B. Dickinson, "The Complexity of Analog Computation," *Mathematics and Computers in Simulation*, in press.
2. D. Farmer, T. Toffoli, S. Wolfram (eds.), *Cellular Automata*, North-Holland Physics Publishing, Amsterdam, 1984.
3. K. Preston, Jr., M. J. B. Duff, *Modern Cellular Automata, Theory and Applications*, Plenum Press, 1984.
4. K. Steiglitz, R. Morita, "A Multi-Processor Cellular Automaton Chip," Proc. 1985 IEEE International Conference on Acoustics, Speech, and Signal Processing, Tampa, Florida, March 26-29, 1985.

5. J. K. Park, K. Steiglitz, W. P. Thurston, "Soliton-Like Behavior in Automata," *Physica D*, in press.

6. F. L. Carter, "The Molecular Device Computer: Point of Departure for Large Scale Cellular Automata," pp. 175-194 in [2].

7. A. V. Oppenheim, R. W. Schafer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, N. J., 1975.

8. E. R. Berlekamp, J. H. Conway, R. K. Guy, *Winning Ways for your Mathematical Plays, Vol. 2: Games in Particular*, (Chapter 25, "What is Life?"), Academic Press, New York, N. Y., 1982.

9. F. Nourai, R. S. Kashef, "A Universal Four-State Cellular Computer," *IEEE Trans. on Computers*, vol. C-24, no. 8, pp. 766-776, August 1975.

10. A. C. Scott, F. Y. F. Chu, D. W. McLaughlin, "The Soliton: A New Concept in Applied Science," *Proc. IEEE*, vol. 61, no. 10, pp. 1443-1483, October 1973.

11. R. Hirota, K. Suzuki, "Theoretical and Experimental Studies of Lattice Solitons on Nonlinear Lumped Networks," *Proc. IEEE*, vol. 61, no. 10, pp. 1483-1491, October 1973.

12. A. Church, "An Unsolvable Problem of Elementary Number Theory," *Amer. J. Math.*, vol. 58, pp. 345-363, 1936. (Reprinted in [13].)

13. M. Davis, *The Undecidable*, Raven Press, Hewlett, NY, 1965.

14. R.P. Feynman, "Simulating physics with computers," *Internat. J. Theoret. Phys.*, vol. 21, pp. 467-488, 1982.

15. M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., San Francisco, CA, 1979. See also: D.S. Johnson, "The NP-completeness Column: an Ongoing Guide," *J. Algorithms*, vol.4, pp. 87-100, 1983.

16. A. Jackson, *Analog Computation*, McGraw-Hill, New York, NY, 1960.

17. W. Karplus and W. Soroka, *Analog Methods*, 2nd. ed., McGraw-Hill, New York, NY, 1959.

18. D. Plaisted, "Some Polynomial and Integer Divisibility Problems are NP-Hard," *Proc. 17th Ann. Symp. on Foundations of Computer Science*, pp. 264-267, 1976.

19. A. M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proc. London Math. Soc., Series 2*, vol. 42, pp. 230-265, 1936-7; corrections *ibid*, vol. 43, pp. 544-546, 1937. (Reprinted in [13].)

20. K. Steiglitz, I. Kamal, A. Watson, "Embedding Computation in One-Dimensional Automata by Phase Coding Solitons," Tech. Rept. No. 15, Dept. of Computer Science, Princeton University, Princeton NJ 08544, Nov. 1985.

21. N. Islam, K. Steiglitz, "Phase Shifts in Lattice Solitons, and Applications to Embedded Computation," in preparation.

22. A. K. Dewdney, "On the Spaghetti Computer and other Analog Gadgets for Problem Solving," in the Computer Recreations Column, *Scientific American*, vol. 250, no. 6, pp. 19-26, June 1984. See also the columns for Sept. 1984, June 1985, and May 1985, the last also containing a discussion of one-dimensional computers.

23. J. B. Salem, S. Wolfram, "Thermodynamics and Hydrodynamics with Cellular Automata," unpublished manuscript, November 1985.

24. U. Frisch, B. Hasslacher, Y. Pomeau, "A Lattice Gas Automaton for the Navier Stokes Equation," Preprint LA-UR-85-3503, Los Alamos National Laboratory, Los Alamos, New Mexico 87545, 1985.

25. S. Wolfram, "Cellular Automaton Fluids 1: Basic Theory," preliminary manuscript, Institute for Advanced Study, Princeton, NJ 08540, 1986.

26. S. R. Sternberg, "Computer Architectures Specialized for Mathematical Morphology," pp. 169-176 in *Algorithmically Specialized Parallel Computers*, L. Snyder, L. H. Jamieson, D. B. Gannon, H. J. Siegel (eds.), Academic Press, 1985.

27. S. Kugelmass, K. Steiglitz, in progress.

28. C. H. Goldberg, "Parity Filter Automata," in preparation.