Amorphous Computing[1]

Harold Abelson, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Thomas F. Knight, Jr., Radhika Nagpal, Erik Rauch, Gerald Jay Sussman, Ron Weiss

MIT Artificial Intelligence Laboratory

Abstract

Amorphous computing is the development of organizational principles and programming languages for obtaining coherent behavior from the cooperation of myriads of unreliable parts that are interconnected in unknown, irregular, and time-varying ways. The impetus for amorphous computing comes from developments in microfabrication and fundamental biology, each of which is the basis of a kernel technology that makes it possible to build or grow huge numbers of almost-identical information-processing units at almost no cost. This paper sets out a research agenda for realizing the potential of amorphous computing and surveys some initial progress, both in programming and in fabrication. We describe some approaches to programming amorphous systems, which are inspired by metaphors from biology and physics. We also present the basic ideas of *cellular computing*, an approach to constructing digital-logic circuits within living cells by representing logic levels by concentrations DNA-binding proteins.

Over the next few decades, two emerging technologies—microfabrication and cellular engineering—will make it possible to assemble systems that incorporate myriads of information-processing units at almost no cost, provided: 1) that all the units need not work correctly; and 2) that there is no need to manufacture precise geometrical arrangements of the units or precise interconnections among them. This technology shift will precipitate fundamental changes in methods for constructing and programming computers, and in the view of computation itself.

Microelectronic mechanical components are becoming so inexpensive to manufacture that we can anticipate combining logic circuits, microsensors, actuators, and communications devices integrated on the same chip to produce particles that could be mixed with bulk materials, such as paints, gels, and concrete. Imagine coating bridges or buildings with smart paint that can sense and report on traffic and wind loads and monitor structural integrity of the bridge. A smart-paint coating on a wall could sense vibrations, monitor the premises for intruders, or cancel noise.

Even more striking, there has been such astounding progress in understanding the biochemical mechanisms in individual cells, that it appears we'll be able to harness these mechanisms to construct digital-logic circuits. Imagine a discipline of cellular engineering

[1]This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for this research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-96-1-1228.

that could tailor-make biological cells to function as sensors and actuators, as programmable delivery vehicles for pharmaceuticals, or as chemical factories for the assembly of nanoscale structures. The ability to fabricate such systems seems to be within our reach, even if it is not yet within our grasp.

Yet fabrication is only part of the story. Digital computers have always been constructed to behave as precise arrangements of reliable parts, and almost all techniques for organizing computations depend upon this precision and reliability. So while we can envision producing vast quantities of individual computing elements—whether microfabricated particles or engineered cells—we have few ideas for programming them effectively. The opportunity to exploit these new technologies poses a broad conceptual challenge, which we call the challenge of *amorphous computing*:

> How does one engineer prespecified, coherent behavior from the cooperation of immense numbers of unreliable parts that are interconnected in unknown, irregular, and time-varying ways?

This paper sets out a research agenda for realizing the potential of amorphous computing and surveys some initial progress, both in programming and in fabrication.

One critical task is to identify appropriate organizing principles and programming methodologies for controlling amorphous systems. We discuss some preliminary ideas in section 1, paying most attention to hints from biology. The growth of form in organisms demonstrates that well-defined shapes and functional structures can develop through the interaction of cells under the control of a genetic program, even though the precise arrangements and numbers of the individual cells are variable. Accordingly, biology can be a rich source of metaphors to inspire new programming methodologies for amorphous systems.

As an illustration, we describe how amorphous media can be programmed to generate complex patterns, such as an arbitrary prespecified interconnection graph. The program is organized according to a botanical metaphor where "growing points" and "tropisms" control the differentiation of amorphous computing agents to form the various elements of the pattern. Other biologically-inspired programming metaphors considered below include marker propagation through diffusion and control of shape through cell mobility. We also explore techniques inspired by physics, where observing that the fundamental processes in the physical world are conservative suggests modeling conservative processes by local exchange methods.

Progress in amorphous computing may demand new approaches to fault-tolerance. Traditionally, one seeks to obtain correct results despite unreliable parts by introducing redundancy to detect errors and substitute for bad parts.[2] But in the amorphous regime, getting the right answer may be the wrong idea: it seems awkward to describe mechanisms such as embryonic development as producing a "right" organism by correcting bad parts and broken communications. The real question is how to abstractly structure systems so we get acceptable answers, with high probability, even in the face of unreliability.

[2] One compelling demonstration of this approach is the Hewlett-Packard Laboratories Teramac, a massively parallel computer constructed from defective chips, which can reconfigure itself and its communication paths to avoid the broken parts and compensate for irregular interconnections. [7] Although Teramac is built from conventional chips, Teramac's disigners view it as a prototype architecture for designing nanoscale computers that would be assembled by chemical processes, where a significant fraction of the parts might be defective.

In addition to new programming methodologies, exploiting the power of amorphous computing will require new physical substrates. It is easy to envision computational particles that combine communication and processing, and there appears to be no fundamental obstacle to building these.[3]

A more aggressive approach to fabricating amorphous systems looks to biology, not just as a metaphor, but as the actual implementation technology for a new activity of *cellular computing*, which is the subject of section 2 of this paper. Biological cells are self-reproducing chemical factories that are controlled by a program written in the genetic code. As engineers, we can take control of this process to make novel organisms with particular desired properties. It is in principle feasible to build a family of logic gates where the signals are represented by concentrations of naturally-occurring DNA-binding proteins, and where the nonlinear amplification is implemented by *in vivo* DNA-directed protein synthesis. Making progress here requires extensive experiments and measurements to characterize the static and dynamic properties of protein logic gates. It also requires the development of new tool suites to support the design, analysis, and the construction of biologic circuits, and we describe some of these.

Besides the obvious potential applications of cellular computing to medicine and to chemical sensing, programmed cells could enable us to manufacture novel materials and structures at molecular scales. The biological world already provides us with a variety of useful and effective mechanisms, such as flagellar motors. If we could co-opt cells to build organized arrays of such motors, with accessible interfaces for power and control, this could have considerable engineering significance. In addition, common biologically available conjugated polymers, such as carotene, can conduct electricity, and can be assembled into active components. If we, as engineers, can acquire mastery of mechanisms of biological differentiation, morphogenesis, and pattern formation, we can use biological entities of our own design as construction agents for building and maintaining complex ultramicroscopic electronic systems. Section 3 speculates on how this might be accomplished.

1 Programming Paradigms for Amorphous Systems

An amorphous computing medium is a system of irregularly placed, asynchronous, locally interacting computing elements. We can model this as a collection of "computational particles" sprinkled irregularly on a surface or mixed throughout a volume. The particles are possibly faulty, sensitive to the environment, and may effect actions. In general, the individual particles might be mobile, but the initial programming explorations described here do not address this possibility.

Each particle has modest computing power and a modest amount of memory. The particles are not synchronized, although we assume that they compute at similar speeds,

[3]In our research at MIT, we are currently implementing a first silicon prototype that will serve not only to explore the engineering issues that must be understood in order to achieve true computational particles, but will also provide a hardware substrate for investigating programming methodological and operating-system issues. This prototype consists of an aggregation of integrated circuits, each containing a 32-bit microprocessor, ROM, static RAM, a radio transmitter/receiver employing spread-spectrum techniques, and environmental sensors. Moving from this kind of prototype to real computational particles presents additional challenges (we are not addressing the issue of power distribution in this initial prototype), but they are nevertheless an evolution of current technologies.

since they are all fabricated by the same process. The particles are all programmed identically, although each particle has means for storing local state and for generating random numbers. In general, the particles do not have any a priori knowledge of their positions or orientations.

Each particle can communicate with a few nearby neighbors. In amorphous systems of microfabricated components, the particles might communicate via short-distance radio; bioengineered cells might communicate by chemical means. For our purposes here, we'll assume that there is some communication radius r, which is large compared with size of individual particles and small compared with the size of the entire area or volume, and that two particles can communicate if they are within distance r.

We assume that the number of particles is very large. Thus, the entire amorphous medium can be regarded as a massively parallel computing system, and previous investigations into massively parallel computing, such as research in cellular automata, is one source of ideas for dealing with amorphous systems. Amorphous computing presents a greater challenge that cellular automata, however, because its mechanisms must be independent of the detailed configuration and reliability of the particles. For example, smart paint should be able to determine geometric properties of the surface that it coats without initial knowledge of the positions of the paint's computational particles.

Another source of ideas may be research into self-organizing systems, which has exhibited how some coherent behaviors of large-scale systems can "emerge" from purely local interactions of individual particles. Amorphous computing might exploit similar phenomena, but it is not our goal to study the principles of self-organization per se. As engineers, we must learn to construct systems so that they end up organized to behave as we a priori intend, not merely as they happen to evolve.

From wave propagation to pattern formation

To get a sense of what it would be like to program an amorphous system, consider a simple process of wave propagation. An initial "anchor" particle, chosen by a cue from the environment or by generating a random value, broadcasts a message to each of its neighbors. These propagate the message to their neighbors, and so on, to create a diffusion wave that spreads throughout the system. The message can contain a hop count, which each particle can store and increment it before rebroadcasting, ignoring any subsequent higher values to prevent the wave from propagating backwards. The hop counts provide estimates of distance from the anchor: a point reached in n steps will be roughly distance nr away. The quality of this estimate depends on the distribution of the particles. Such relations have been extensively studied in investigations of packet-radio networks. (See, for example, Kleinrock and Silvester [10].)

For particles on a surface, one can produce two-dimensional coordinate systems by propagating waves from two anchors. Using three anchors establishes a triangular coordinate system, which can provide better accuracy, especially when augmented by smoothing techniques as discussed by Coore [3] and Nagpal [14]. In related work, Katzenelson [9] describes how to establish global coordinates over a region by propagating known coordinates from particles at the boundary.

Wave propagation with hop counts, as Nagpal [14] remarks, is evocative of the gradients formed by chemical diffusion that are believed to play a role in biological pattern formation. Consequently, we can attempt to organize amorphous processes by mimicking gradient phenomena observed in biology.

As an example, we can use diffusion waves to produce regions of controlled size, simply by having the processors relay the message only if the hop count is below a designated bound. Once a region is generated in this way, we can use it to control the growth of other regions. For instance, two particles A and B might each produce a diffusion wave, but the wave from B could be relayed only by particles that have not seen the wave from A. Drawing upon a biological metaphor, we might interpret this as saying that A generates a wave that "inhibits the growth" that has started from B. In a slightly more elaborate program, the B-wave might be relayed only by the particle located in each neighborhood that is closest to A (as measured by the A-wave). Our biological metaphor might interpret this by explaining that the region growing from B has a "tropism" that attracts it towards A.

These diffusion wave mechanisms are well matched to amorphous computing because the gross phenomena of growth, inhibition, and tropism are insensitive to the precise arrangement of the individual particles, as long as the distribution is reasonably dense. In addition, if individual particles do not function, or stop broadcasting, the result will not change very much, so long as there are sufficiently many particles.

Based on this kind of cartoon caricature of biological development, Coore [4] has developed a programming language called the growing-point language, (GPL), which enables programmers to specify complex patterns, such as those specifying the interconnect of an electronic circuit. The specification is compiled into a uniform state machine for the computational particles in an amorphous medium. All of the particles have the same program. As a result of the program, the particles "differentiate" into components of the pattern.

Coore's language represents processes in terms of a botanical metaphor of "growing points". A growing point is an activity of a group of neighboring computational particles that can be propagated to an overlapping neighborhood. Growing points can split, die off, or merge with other growing points. As a growing point passes through a neighborhood, it may permanently set some portion of the states of the particles it visits. We can interpret this as the growing point laying down a particular material as it passes. The growing point may be sensitive to particular diffused messages, and in propagating itself, it may exhibit a tropism toward a source, away from a source, or move in a way that attempts to keep the "concentration" of some diffused message constant. Particles that represent particular materials may "secrete" appropriate diffusible messages that attract or repel specific growing points.

Here is a fragment of a program written in the growing point language:

```
(define-growing-point (make-red-branch length)
  (material red-stuff)
  (size 5)
  (tropism (and (away-from red-pheromone)
                (and (keep-constant source-1-pheromone)
                     (keep-constant source-2-pheromone))))
  (avoids green-pheromone)
  (actions
   (secrete 2 red-pheromone)
```

```
(when ((< length 1)
       (terminate))
      (default
        (propagate (- length 1))))))
```

The program defines a growing point process called `make-red-branch`, which takes one parameter called `length`. This growing point "grows" material called `red-stuff` in a band of size 5. It causes each particle it moves through to set a state bit that will identify the particle as `red-stuff`, and also causes the particle to propagates a wave of extent 5 hops that similarly converts nearby particle to be `red-stuff`. The growing point moves according to a tropism that directs it away from any source of `red-pheromone`, in such a way that the concentration of pheromones secreted by `source-1` and `source-2` are kept constant, and so as to avoid any source of `green-pheremone`. All particles that are `red-stuff` secrete `red-pheromone`; consequently, the growing point will tend to move away from the material it has already laid down. The value of the `length` parameter determines how many steps the growing point moves: If `length` is less than 1 the growing point terminates. Otherwise, it propagates growth of `length` minus 1 steps.

Notice how this language encourages the programmer to think in terms of abstract entities like growing points and pheromones. The GPL compiler translates these high-level programs into an identical set of directives for each of the individual computational particles. The directives are supported by the GPL runtime system running on each particle. In effect, the growing point abstraction provides a serial conceptualization of the underlying parallel computation.

Figure 1(a) shows the first stages of a pattern being generated by a program in the growing-point language. For simplicity, we assume that the horizontal bands at the top and bottom have been previously generated, and that an initial growing point is located at the left. Growth proceeds following a tropism that tries to keep equidistant from the top and bottom bands. After a short while the initial growing point splits into two: one branch of growth is attracted towards the top and one is attracted towards the bottom. Figure 1(b) shows the process somewhat further along: the two branches are repelled by short-range pheromones secreted by the top and bottom bands, and start moving horizontally. They also change the kind of material they lay down.

Figure 1(c) shows the process evolved even further, to produce an elaborate shape. The shape is the layout of a chain of CMOS inverters, where the different colored regions represent structures in the different layers of standard CMOS technology: metal, polysilicon and diffusion. The program that specifies the shape is only a few paragraphs long, and the resulting state machine for the individual particles requires only about twenty states. Coore has demonstrated that any prespecified planar graph can be generated, up to connection topology, by an amorphous computer under the control of a growing-point program, provided that the distribution of particles is sufficiently dense.

Programming the particles

The growing point language is formulated in terms of abstractions that must ultimately be implemented by processes in the individual computational particles, which we assume are all programmed identically. Weiss [24] has developed a remarkably convenient and simple

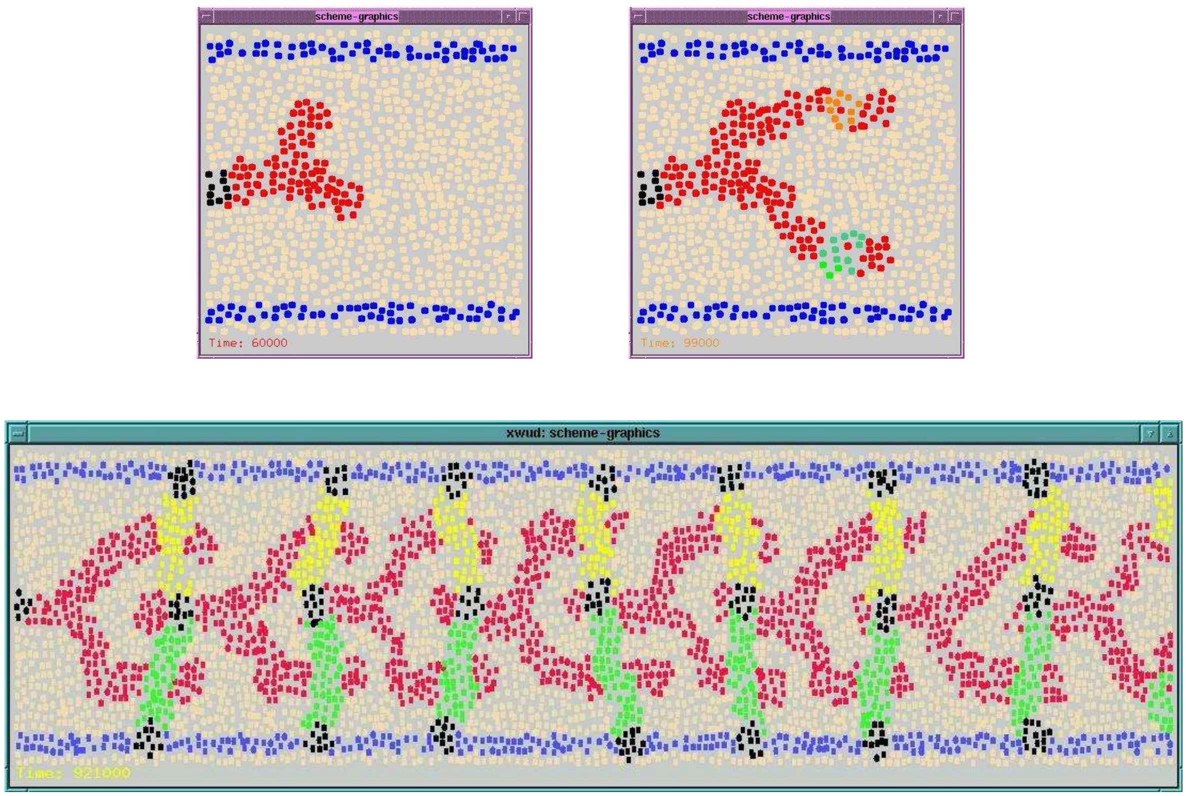

Figure 1: Evolution of a complex design—the connection graph of a chain of CMOS inverters—being generated by a program in Coore's growing-point language. (a) An initial "poly" growth divides to form two branches growing towards "V_{dd}" and "ground". (b) The branches start moving horizontally, and sprout pieces of "diffusion". (c) The completed chain of inverters.

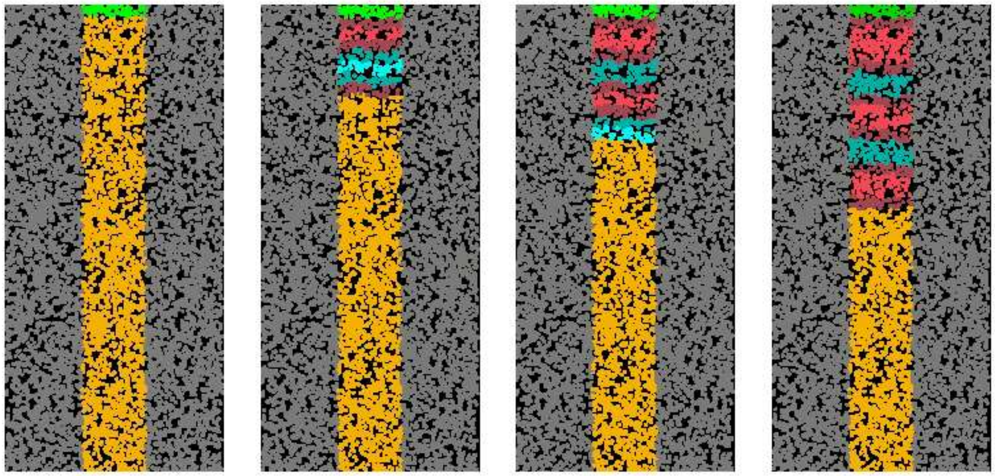

Figure 2: A pattern of alternating bands produced by marker propagation with the aid of Weiss's programming model.

language for programming the particles. In this model, the program to be executed by each particle is constructed as a set of independent rules. The state of each particle includes a set of binary markers, and rules are enabled by boolean combinations of the markers. The rules that are enabled are triggered by the receipt of labelled messages from neighboring particles. A rule may set or clear various markers, and it may send further messages. A message is count that determines how far it will diffuse, and a marker has a lifetime that determines how long its value lasts. Underlying this model is a runtime system that automatically propagates messages and manages the lifetimes of markers, so that the programmer need not deal with these operations explicitly.

Figure 2 shows Weiss's system generating a pattern of alternate bands of red and blue in a "Tube" of particles that are initially distinguished by having a `tube` marker set in them. Here is a fragment of a program to generate this, showing four rules:

```
((make-seg seg-type)
 (and Tube (not red) (not blue))
 ((set seg-type)
  (send created 3)))

(created
 (or red blue)
 ((set Waiting 10)))

(((make-seg *) 0)
 Tube
 ((set Bottom)))

((Waiting 0)
 (and Bottom red)
```

```
 ((send (make-seg blue) 3)))

((Waiting 0)
 (and Bottom blue)
 ((send (make-seg red) 3)))
```

The first rule describes the reaction of a particle to receiving a message labelled `make-seg`, specifying a `seg-type` (which will be `red` or `blue`). If the particle has its `tube` marker set, and does not have its `red` marker or `blue` marker set, it sets the bit for the specified `seg-type` and sends a `created` message that will propagate for 3 hops. The second rule says that when a particle receives a `created` message, and it has the `red` marker or the `blue` marker set, it turns on its `waiting` marker with a lifetime of 10. The third rule says that any particle whose `tube` marker is set, that receives any `make-seg` message with a hop count of zero, should set its `bottom` marker. The fourth rule says that when the lifetime of the `waiting` marker runs out, and the particle has both the `red` and `bottom` markers set, the particle sends a `(make-seg blue)` message, which will propagate for 3 hops. The result will be alternating red and blue bands, along the length of the tube.

Weiss's system is almost powerful enough to represent the processes described by Coore's growing points, yet it is simple enough that it can be implemented in an elementary way. It does not depend on any arithmetic or data structures, and it would be an obvious candidate for implementation by real biological cells and the cellular computing technology discussed in section 2.

Further metaphors from biology

The sketches above merely hint at the new primitives and organizational principles required for effective control of amorphous computing systems, and the use of metaphors from biology has hardly begun to be tapped. Coore, Nagpal, and Weiss [5], for instance, have developed a model for spontaneously organizing amorphous particles into hierarchies of groups that can act as single entities and can collaborate to form higher-level groups; within a group, members can specialize to perform particular functions. One can compare this to the organization of cells into tissues, tissues into organs, and organs into systems.

A particularly fruitful source of inspiration from biology should emerge from the observation that even the most basic morphogenetic processes, such as gastrulation, involve cell migration and deliberate changes of cell shapes. Figure 3 shows some evocative simulations based on mechanical models of cells by Odell et. al. [15]. In this model each individual cells preserve its volume, but has actuators (in this case, fibers) that it can stretch or relax to change cell shape, and it can react to the stresses in its neighbors. In addition, the entire collection of cells bounds a fluid-filled cavity that is constrained to preserve volume as the cell shapes change. The figure shows how different shapes and behaviors (elongation, invagination, and so on) appear as the result of changes by individual cells.

One can envision extending the message propagation models described above to incorporate local sensing and activation. Can one create a language of shapes—analogous to the growing-point language—that would permit programmers to generate prespecified macroscopic shapes in amorphous media, by prescribing local shape changes by individual particles? In general, it is plausible to expect that the most powerful techniques for amorphous computing will be ones that will tie computation intimately to particle activation and mobility, and to physical constraints from the environment.

Physics and conservative systems

Physics, as well as biology, can be a source of new metaphors for amorphous computing. The mechanisms discussed above were based on a "chemical diffusion" model. Chemical diffusion, and other dissipative processes such as heat diffusion, are natural candidates for simulation in amorphous media because dissipation loses information, erasing any microscale errors that occur in computation. The fundamental processes in the physical world, in contrast, are conservative. Simulating conservative processes, such as those characterized by the wave equation, is much more difficult because conservative (and especially reversible) processes never forget the error accrued. It is an especially challenging task to formulate processes that manifest exact conservation laws in such a way that imperfection in the implementation does not impair the exact conservation.

One approach is to simulate processes in terms of explicit discrete computational tokens of the conserved quantities. With such a scheme we can guarantee global conservation by formulating the process in terms of local exchanges of the tokens. Conservation laws then emerge globally as consequences of the local exchanges.

Consider a two-dimensional flexible sheet, constrained to move in one dimension (perpendicular to the sheet). We can simulate motions of the sheet by integrating the scalar wave equation $\partial^2 q/\partial t^2 = c^2 \nabla^2 q$, where $q(x, y)$ is the displacement of the sheet at (x, y). Imagine the sheet to be an amorphous medium densely populated by computational particles of unit mass, and formulate the integration as a continual process of momentum exchange, where each pair of neighboring particles exchanges an appropriate "token" of momentum. Letting q_i be the displacement of particle i, we can model the force on particle i from a neighboring particle j as resulting from Hooke's law: $F_{ij} = k(q_i - q_j)$, and so the amount of momentum transferred from particle i to particle j in time Δt is $\Delta p_{ij} = k(q_i - q_j)\Delta t$. In the resulting amorphous computing program, each particle repeatedly chooses a neighbor at random, (atomically) effects the momentum exchange, and evolves the position and velocity. It seems preferable to evolve the position and velocity using a scheme such as leapfrog integration, since this respects conservation of energy. (This is essentially the program advocated by Greenspan [6] for making "particle models" of physical systems. The higher-order symplectic integrators that are used in modeling the solar system can also be viewed in this light, and may form the basis for more accurate integrations [26].)

The simulation shown in figure 4 is from preliminary results by Rauch [18], who has been investigating such discrete, amorphous models of physical systems. In a generalization of this work, Katzenelson [9] has demonstrated that amorphous media can be programmed to integrate any differential equation for a conservation law, provided that the processors are sufficiently dense. ("Sufficiently dense" here means that when one draws lines connecting every particle to all particles within the communication radius r, the resulting triangles are all acute.)

The use of discrete tokens to represent conserved quantities works only if we can guarantee that the tokens are not lost or duplicated if communications network is imperfect or if the computational particles fail. One idea is to represent our tokens in a redundant

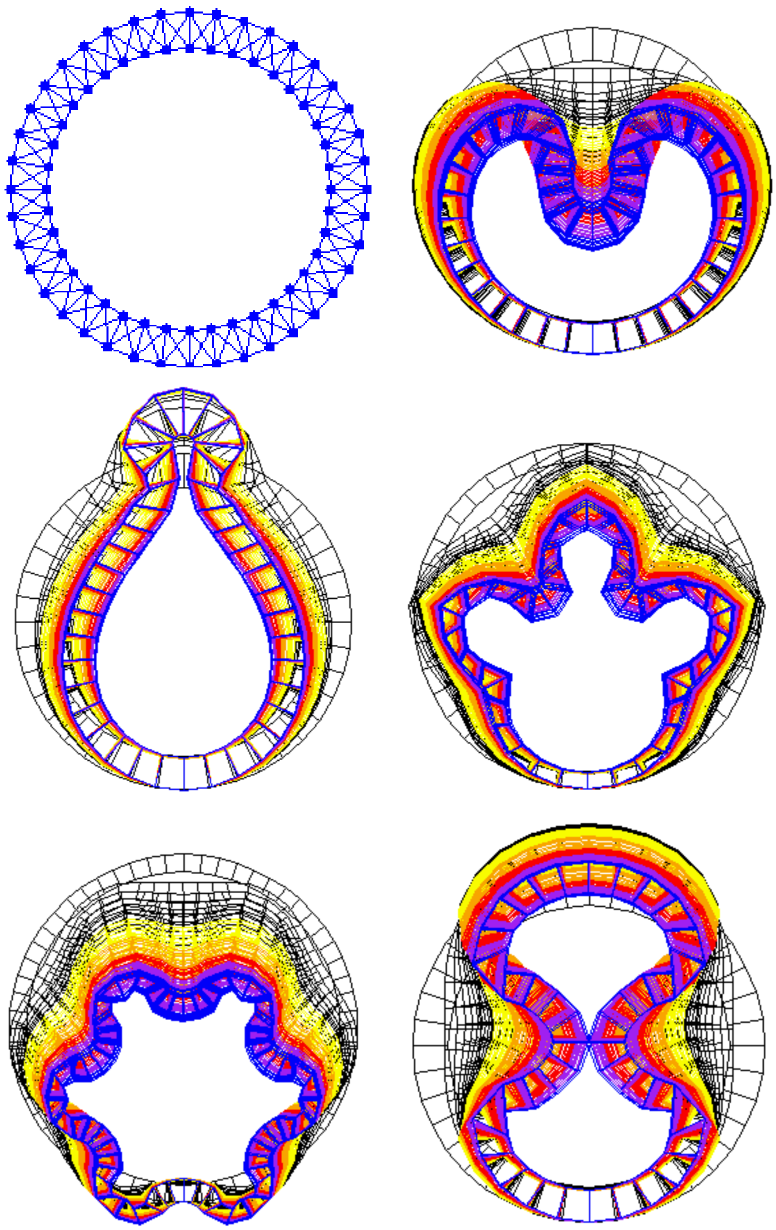

Figure 3: Control of shape changes in a ring of cells, based on the mechanical cell models of [15]. Each cell has a simple programmed behavior, and reacts to stresses in its neighbors.

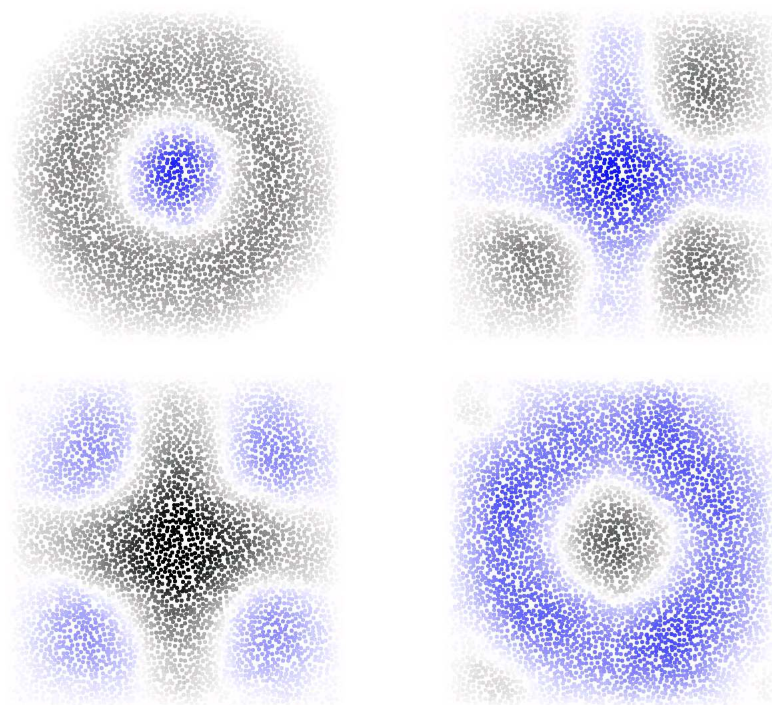

Figure 4: Simulations of the wave equation in a two-dimensional amorphous medium.

distributed form. In the spirit of amorphous computing, we should be prepared to use profligate amounts of local computation to compensate for the unreliability of the individual elements, but the details of how to accomplish this robustly remain an important challenge.

2 Cellular Computing

Living cells serve as isolated, controlled environments that house complex chemical reactions. In addition, cells reproduce themselves, allowing the creation of many copies with little manufacturing effort. The vision of cellular computing is to use intracellular chemical mechanisms to organize and control biological processes, just as we use electrical mechanisms to control electrical processes. The ability to control cellular function will provide important capabilities in computation, materials manufacturing, sensing, effecting, and fabrication at the molecular scale.

Sussman and Knight [12] have proposed a biochemically plausible approach to constructing digital-logic signals and gates of significant complexity within living cells. This approach relies on co-opting existing biochemical machinery found naturally within the cell, as a basis for implementing digital logic. The "signals" in this logic system are represented by concentrations of certain DNA-binding proteins.[4]

[4]Bacterial cells usually contain only a small number of molecules of any particular DNA-binding protein. This small number results in a degree of stochastic behavior in bacteria. In natural environments, this

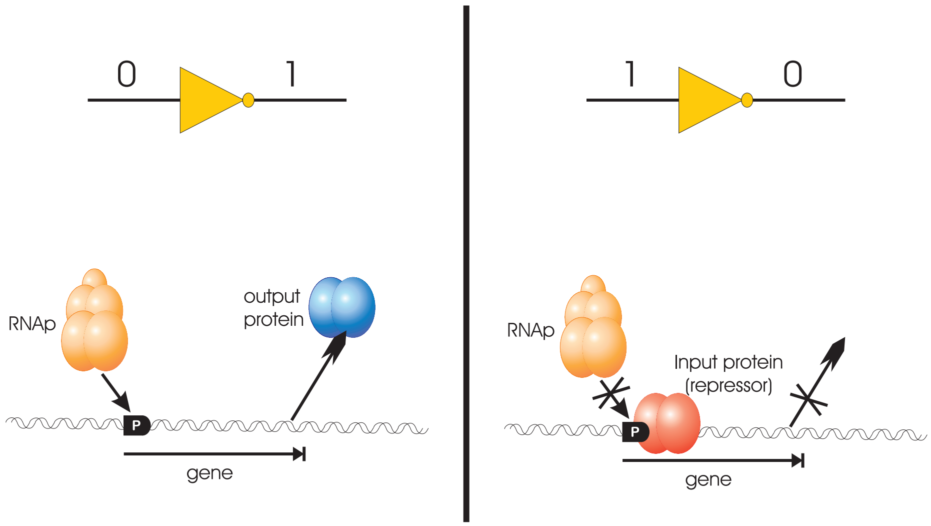

Figure 5: The two idealized cases for a biological inverter. If input repressor is absent, RNAp (RNA polymerase) transcribes the gene for the output protein and enables its synthesis. If input repressor is present, no output protein is synthesized.

The essential idea of cellular computing is to adopt the same strategy as in electrical engineering, where engineers create *digital abstractions* that permit the design of systems that are insensitive to variations in signal levels. The key to obtaining a digital abstraction is the existence of a inverting amplifier. The inverter must produce adequate noise margins, i.e., ranges where signal variations in the inputs are not significant to the next stage in computation. This requires an amplifier that is nonlinear and whose average gain is greater than unity.

To see how to construct such an amplifier in the cellular context, consider an "output" protein Z and an "input" protein A that serves as a repressor for Z. A cellular computing "inverter" can be implemented in DNA as a genetic unit consisting of an *operator* (a binding site for A), a *promoter* (a site on the DNA at which RNA polymerase binds to start transcription), and a *structural gene* that codes for the production of Z.

In order for Z to be produced, an RNA polymerase has to bind to the promoter site and transcribe the structural gene into messenger RNA. (The messenger RNA is then translated into the protein by another molecular machine, called a ribosome.) If a molecule of A binds to the operator site, it prevents the docking of the RNA polymerase to the promoter site, thus preventing the transcription (and later translation) of the gene. Thus, assuming that proteins are scavenged and have a finite lifetime, the concentration of Z will vary inversely with the concentration of A. Figure 5 depicts this process.

The gain of this "inverter" can be increased by arranging for multiple copies of the structural gene to be controlled by a single operator. The required nonlinearity can be obtained by using multimer binding proteins (i.e., proteins constructed from several subunits

stochastic behavior provides a survival advantage by increasing the apparent diversity of a population, as demonstrated by McAdams and Arkin [13]. For engineered systems, however, we would like behavior to be as predictable as possible, and this requires increasing the concentrations of the signalling proteins.

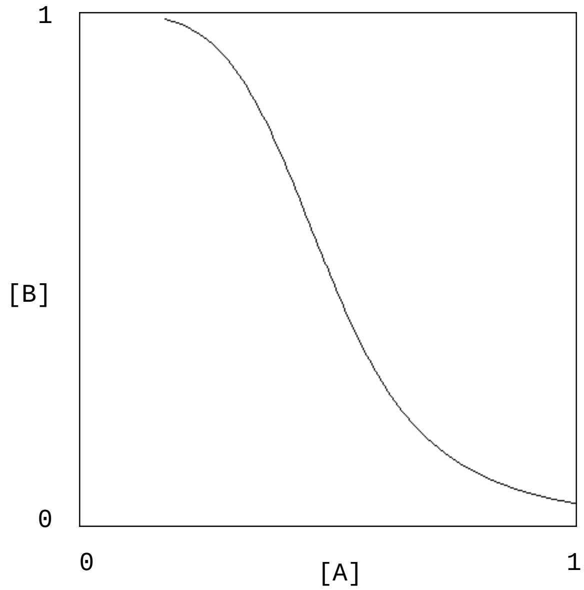

Figure 6: The DC transfer curve for the "inverter" that operates on DNA-binding proteins is similar to the input-output transfer characteristic of an digital inverter. The model of the chemical kinetics employed here postulates that four molecules of the repressor protein are required to inhibit production of the output.

that must come together to bind to the DNA).

Figure 6 (taken from [12]) shows the output protein concentration as a function of the input protein concentration, computed by modeling the chemical kinetics of the DNA-binding protein reactions. Observe that the relation between input and output here is an almost ideal transfer characteristic for a digital inverter: there is low gain for input concentrations that are very high and very low, separated by a relatively high-gain transition region. This nonlinearity is the essence of digital gates; it forms the basis for effectively rejecting small variations in the input signals—that is, for attenuating the input noise. Figure 7 shows the inverter's dynamic behavior derived from numerical simulation of the actual chemical kinetics for suitable proteins (see [25]).

Given the ability to implement inverters in cells, arbitrary logic gates can then be realized as combinations of inverters. For example, the NAND function can be implemented as two inverters that have different input repressors but the same output protein: the output will be produced unless inhibited by both of the inputs. More complex components, such as registers that store state, can be similarly constructed, just as in standard electrical logic design. In many respects, this is similar to the I^2L logic family of digital circuits. One difference is that, rather than using clocked circuits, cellular logic circuits are likely to be asynchronous and level-based rather than edge-based, because the signal propagation, based on diffusion of proteins, makes it difficult to achieve synchronization.

In addition to realizations of digital logic, cellular gates could also code for enzymes that effect some other action within the cell, such as motion, illumination, enzymatic catalysis or cell death. Similarly, an input to a cellular logic gate could consist, not of the output of another logic gate, but of a sensor that creates or modifies a DNA binding protein in response to illumination, a chemical in the environment, or the concentration of specific

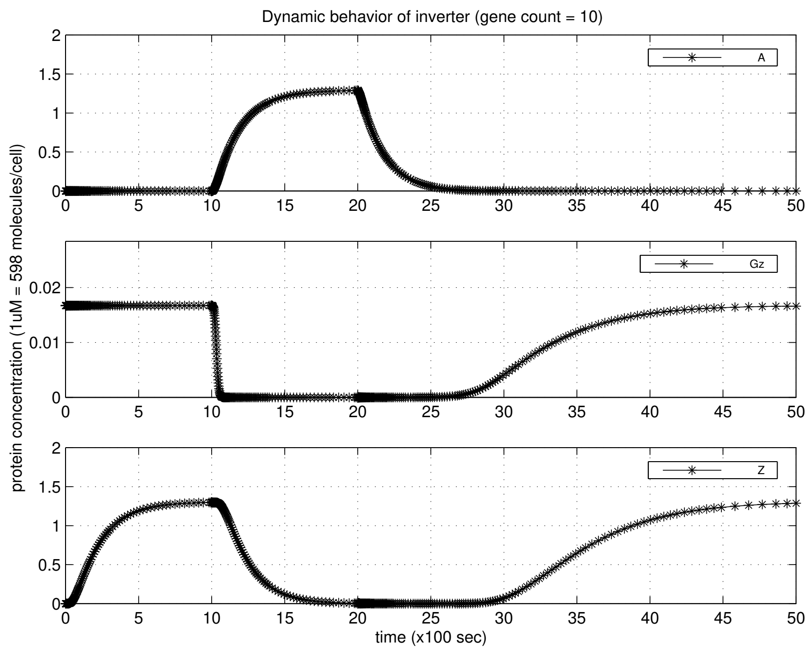

Figure 7: A simulation of the dynamic behavior of a cellular-computing inverter. The three graphs show the concentration of the input protein A, the concentration of the output protein Z, and the concentration G_Z of the gene coding for Z that is not repressed. The concentrations include both the monomeric and dimeric forms. The simulations include both the transcription and translation stages of protein synthesis. The particular proteins being simulated here are phage Lambda cI, with operator regions $\mathrm{O_R}1$ and $\mathrm{O_R}2$ and promoter region $\mathrm{P_R}$ using the kinetic models found in [8, 17].

intracellular chemicals.

A research agenda for cellular computing

In principle, these foundations should be sufficient to implement digital logic in cells. In practice, however, realizing cellular logic will require an ambitious research program. We do not have a library of the available DNA-binding proteins and their matching repressor patterns. We do not have good data about their kinetic constants. We do not know about potential interactions among these proteins outside of the genetic regulatory mechanisms. Most importantly, we do not have a sufficiently clear understanding of how cells reproduce and metabolize to enable us to insert new mechanisms in such a way that they interact with those functions in predictable and reliable ways.

Beyond just our lack of knowledge of the biochemistry, the design of cellular logic circuits raises difficulties not present with electrical circuits. To prevent interference between the gates, a different protein must used for each unique signal. Therefore, the number of proteins required to implement a circuit is proportional to the complexity of the circuit. Also, because the different gates use different proteins, their static and dynamic properties will vary. Moreover, unlike electrical circuits, where the threshold voltages are the same for all devices in a given logic family, the components (proteins) of cellular gates have different characteristics depending on their reaction kinetics. Therefore, the designer of biological digital circuits must take explicit steps to ensure that the signal ranges for coupled gates are appropriately matched.

One effort required for making progress in cellular computing is the creation of tool suites to support the design, analysis, and the construction of biologic circuits. One such tool (currently being explored by Weiss [24]) is a simulator and verifier for genetic digital circuits, called BioSpice. BioSpice takes as inputs the specification of a network of gene expression systems (including the relevant protein products) and a small layout of cells on some medium. The simulator computes the time-domain behavior of concentration of intracellular proteins and intercellular message-passing chemicals. A second tool would be a "Plasmid Compiler" that takes a logic diagram and constructs plasmids to implement the required logic in a way compatible with the metabolism of the target organism. Both the simulator and the compiler must incorporate a database of biochemical mechanisms, their reaction kinetics, diffusion rates, and their interactions with other biological mechanisms.

An even more aggressive approach to cellular computing would be to genetically engineer novel organisms whose detailed structure is completely understood and accessible from an engineering standpoint. One idea for accomplishing this is to gradually transfer functionality from a wild type bacterial chromosome to one or more increasingly complex plasmids. As functionality is transfered, the gene sequences transfered can be deleted from or inactivated in the wild type chromosome, leading to a cell dependent on the presence of the new construct. Eventually, when sufficient function has been transfered to one or more plasmids, the original wild type chromosome can be deleted, yielding a novel organism. Careful choices in what is transferred could lead to the design of "minimal organisms" that have clean modularity and well-understood structure. Such organisms could then serve as substrates for precision cellular engineering.

In essence, we are at a primitive stage in the development of this cellular computing analogous to the early stages of the development of electronics at the beginning of the 20th century. Progress in cellular computing would open a new frontier of engineering that could dominate the information technology of the next century.

3 Toward Nanoscale Computing?

Even though biological cells come in vast numbers, cellular computing will be slow: we cannot expect diffusion-limited chemical processes to support high-speed switching. Thus, we do not anticipate that cellular computing in itself be a good way to solve computationally hard problems. On the other hand, the ability to organize cells into precise patterns and to cause cells to secrete chemical components could be the foundation for the engineering construction of complex extracellular structures and precise control of fabrication at the sub-nanometer level. This kind of engineering will require applying the organizational principles of amorphous computing to the mechanisms of cellular computing. In the future, biological systems could be our machine shops, with proteins as the machine tools and with DNA as the control tapes.

We can envision applying this technology to the construction of molecular-scale electronic structures. Deliberately assembled molecular-scale electronic structures are likely to replace lithographically patterned electronics in the next century. While lithographic technologies struggle to surmount difficulties imposed by the small scale and statistical nature of doping profiles, deliberately assembled molecular-scale systems are atomically precise and uniform, with identical atoms in well-defined localized slots.

The delivery of molecular-scale electronics will require two major technical achievements. The first of these is the development and characterization of molecular-scale conductors, diodes, and transistors. Devices at this scale can incorporate electrical circuitry with picosecond cycle times [1, 2, 11, 16, 19, 20, 21, 23]. While many researchers are appropriately concentrating on the conductor and device behaviors, a solution to those problems will not be sufficient. Thus, the second major achievement required is a technology for assembling compound structures from molecular-scale components.

One plausible way to construct complex, information-rich electronic systems is to first fabricate a largely passive, but information-rich molecular-scale "scaffold," consisting of selectively self-assembling engineered molecules. This scaffolding can be used to support the fabrication of molecular conductive and amplifying devices that are interconnected in the way that the engineer requires. Proteins are a good candidate for scaffolding components because they are chemically and thermally stable, and they have exquisitely selective binding domains. For example, the selectivity of the antibody fold is the basis of the immune system.

In principle, to manufacture proteins requires merely inserting the appropriate DNA sequences into cells. This, of course, begs the enormously difficult question of how to engineer proteins that have the required properties. Rather than trying to create such proteins ab initio, we may be able to identify classes of biologically available proteins that are usable as scaffolding.

One intriguing idea is to build scaffolding components out of collagen. Collagen forms the basement membrane upon which animal cells self assemble, and it is probably the most common animal protein [22]. It consists of an offset triple helix, in which each strand has a stylized amino acid sequence of the form ... GLY-PRO-X-GLY-PRO-X-GLY-PRO-X .... In this sequence, every third amino acid is glycine, and many of the other amino acids are proline (or a modified form, hydroxy-proline). The amino acids other than the glycine and proline residues can can be arbitrary (here denoted by X). The entire structure forms a rigid, straight rod, where the side chains of the X amino acids point outwards. In our vision of collagen as scaffolding, the GLY-PRO- amino acids assure that the protein folds as a rigid rod, and, by choosing appropriate X amino acids, we control how these rods assemble into more complex two-dimensional and three-dimensional structures.

A Fantasy

With this perspective we can entertain a fantasy of nanoscale circuit fabrication in a future technology. Imagine that there is a family of primitive molecular-electronic components, such as conductors, diodes, and switches. We assume that we have bottles of these in the freezer. (Probably there are generic parts suppliers for these common components.)

Suppose that we have a (perhaps very large) circuit that we want to implement in this technology. The first stage of the construction begins with the circuit and builds a layout (perhaps 3 dimensional!) that incorporates the sizes of the components and the ways that they might interact.

Next, the layout is analyzed to determine how to construct a scaffold out of collagen. Each branch is compiled into a collagen strut that links only to its selected targets. The struts are labeled so that they bind only to the appropriate electrical component molecules. For each collagen strut, the DNA sequence to make that kind of strut is assembled, and a protocol is produced to insert the DNA into an appropriate cell. These various custom parts are then synthesized by such transformed cells.

Finally, we create an appropriate mixture of these custom scaffold parts and generic electrical parts. Specially-programmed worker cells are added to the mixture, to implement the circuit edifice that we desire. These worker cells have complex programs, developed with amorphous computing technology. The programs control how the workers perform their particular task of assembling the appropriate components in the appropriate patterns. With a bit of sugar (to pay for their labor) the workers construct (many copies) of our circuit, which we can then collect, test, and package for use.

References

[1] Robert Birge, "Protein-Based Computers," *Scientific American*, March 1995.

[2] C.P. Collier, E.W. Wong, M. Belohradsky, F.J. Raymo, J.F. Stoddart, P.J. Kuekes, R.S. Williams, and J.R. Heath, "Electronically Configurable Molecular-Based Logic Gates," *Science*, v. 285, no. 5426, p. 391, 16 July 1999.

[3] Daniel Coore, "Establishing a Coordinate System on an Amorphous Computer," in *1998 MIT Student Workshop on High Performance Computing in Science and Engineering*, MIT Laboratory for Computer Science, TR-737.

[4] Daniel Coore, "Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer," Ph.D. thesis, Massachusetts Institute of Technology Department of Electrical Engineering and Computer Science, December 1998.

[5] Daniel Coore, Radhika Nagpal and Ron Weiss, "Paradigms for Structure in an Amorphous Computer," MIT Artificial Intelligence Laboratory Memo no. 1614, 1997.

[6] Donald Greenspan, *Particle Modeling*, Boston: Birkhauser, 1997.

[7] James R. Heath, Philip J. Keukes, Gregory S. Snider, R. Stanley Williams, "A Defect-Tolerant Computer Architecture: Opportunities for Nanotechnology," *Science*, 280 (5370): 1716.

[8] Roger W. Hendrix, *Lambda II*, Cold Spring Harbor Press, Cold Spring Harbor, New York, 1983.

[9] Jacob Katzenelson, "Notes on Amorphous Computing," MIT Artificial Intelligence Laboratory Memo (to appear).

[10] Leonard Kleinrock and John A. Silvester, "Optimum Transmission Radii in Packet Radio Networks or Why Six is a Magic Number," *National Telecommunications Conference*, Birmingham, Alabama, pages 04.3.1-5, December 1978.

[11] H. Kless (ed.), *Conjugated Conducting Polymers*, Springer Verlag, New York, 1992.

[12] Thomas F. Knight and Gerald Jay Sussman, "Cellular Gate Technology," in *Unconventional Models of Computation*, Springer-Verlag, Berlin, 1998.

[13] Harley H. McAdams and Adam Arkin, "Simulation of Prokaryotic Genetic Circuits," *Annu. Rev. Biom. Struct.*, 1988, 27:199–224.

[14] Radhika Nagpal, "Self-Organizing a Global Coordinate System from Local Information," (in progress)

[15] G.M. Odell, G. Oster, P.Alberch, and B. Burnside, "The Mechanical Basis of Morphogenesis, 1. Epithelial Folding and Invagination," *Developmental Biology*, vol. 85, 1981, pp. 446–462.

[16] M.C. Petty, M.R. Bryce, and D. Bloor, *Introduction to Molecular Electronics*, Oxford University Press, New York, 1995.

[17] Mark Ptashne, *A Genetic Switch: Phage lambda and Higher Organisms*, second edition, Cell Press and Blackwell Scientific Publications, Cambridge, MA, 1986.

[18] Erik Rauch, Notes on a Discrete, Amorphous, Physical Models, (thesis in progress)

[19] Mark Reed, *Sub-nanoscale electronic systems and devices*, US Patent 5,475,341, December, 1995.

[20] Mark Reed and W.P. Kirk (eds.), *Nanostructure Physics and Fabrication*, Academic Press, San Diego, 1989.

[21] TA Skotheim, RL Elsenbaumer, and J.R. Reynolds, *Handbook of Conducting Polymers*, 2nd Ed. Marcel Dekker, Inc., New York, 1998.

[22] J. Vincent, *Structural Biomaterials*, Princeton University Press, Princeton, NJ, 1990.

[23] N Vsevolodov, *Biomolecular Electronics*, Birkhäuser, Boston, 1998.

[24] Ron Weiss, George Homsy, and Radhika Nagpal, "Programming Biological Cells," Abstract presented in *ASPLOS 98, Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1998.

[25] Ron Weiss, George Homsy, and Thomas F. Knight, "Toward in vivo Digital Circuits," *Dimacs Workshop on Evolution as Computation*, Princeton, NJ, January 1999.

[26] Jack Wisdom and Matthew Holman, "Symplectic maps for the N-body problem," *Astron J.*, vol. 102, 1528, 1991.