# AMORPHOUS

**For coherent behavior from vast numbers of unreliable microsensors, actuators, and communication devices interconnected in unknown ways, apply the lessons of cellular cooperation in biological organisms.**

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

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

Even more striking is the amazing progress in understanding the biochemical mechanisms in individual cells, promising that we'll be able to harness these mechanisms to construct digital logic circuits.

Harold Abelson, Don Allen, Daniel Coore, Chris Hanson, Erik Rauch, Gerald Jay Sussman, and Ron Weiss

# COMPUTING

Imagine a discipline of cellular engineering that tailor-makes 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 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 on this precision and reliability. We can envision producing vast quantities of individual computing elements—whether microfabricated particles or engineered cells—but we have few ideas for programming them effectively. The opportunity to exploit these new technologies poses a broad conceptual challenge—the challenge of *amorphous computing*. How can prespecified, coherent behavior be engineered from the cooperation of vast numbers of unreliable parts interconnected in unknown, irregular, and time-varying ways?

One critical task is to identify appropriate organizing principles and programming methodologies for controlling amorphous systems. The growth of form in biological 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, we discuss some ideas for controlling amorphous systems, especially the hints from biology. We turn to biology not just as a metaphor, but as an actual implementation technology for amorphous systems by means of "cellular computing," which constructs logic circuits within living cells.

## Programming Paradigms for Amorphous Systems

An amorphous computing medium is a system of irregularly placed, asynchronous, locally interacting computing elements. We can model this medium 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 produce actions. In general, the individual particles may be mobile, but the initial programming explorations described here do not address this possibility.

**Figure 1. A program 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 pheromone-1)
                     (keep-constant pheromone-2))))
  (avoids green-pheromone)
  (actions
   (secrete 2 red-pheromone)
   (when ((< length 1)
          (terminate))
         (default
           (propagate (- length 1))))))
```

Each particle has modest computing power and a modest amount of memory. The particles are not synchronized, although we assume they compute at similar speeds, since they are all fabricated by the same process. The particles are all programmed identically, although each one 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.

George Homsy, Thomas F. Knight, Jr., Radhika Nagpal,

Each particle can communicate with a few nearby neighbors. In amorphous systems of microfabricated components, the particles may communicate via short-distance radio, and bioengineered cells may communicate by chemical means. For our purposes here, we assume there is some communication radius *r* that is large compared to the size of individual particles and small compared to 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 a source of ideas for dealing with amorphous systems. Amorphous computing presents a greater challenge than cellular automata, however, because its mechanisms must be independent of the detailed configuration and reliability of the individual particles. For example, smart paint should be able to determine geometric properties of the surface 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 exhibiting how coherent behaviors of large-scale systems can "emerge" from the 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 have to learn to construct systems so they end up organized to behave as we a priori intend, not merely as they happen to evolve.

***Wave propagation.*** 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 neighbors 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 that each particle can store and increment before rebroadcasting, ignoring any subsequent higher values to prevent the wave from propagating backward.

The hop counts provide estimates of distance from the anchor; a point reached in *n* steps is roughly distance *nr* away. The quality of this estimate depends on the distribution of the particles. Such relations have been studied extensively in investigations of packet-radio networks [6]. For particles on a surface, one can produce 2D coordinate systems by propagating waves from two anchors. Using three anchors establishes a triangular coordinate system with better accuracy, especially when augmented by smoothing techniques [9].

Wave propagation with hop counts is evocative of the gradients formed by chemical diffusion believed to play a role in biological pattern formation. Correspondingly, we can attempt to organize amorphous processes by mimicking gradient phenomena observed in biology.

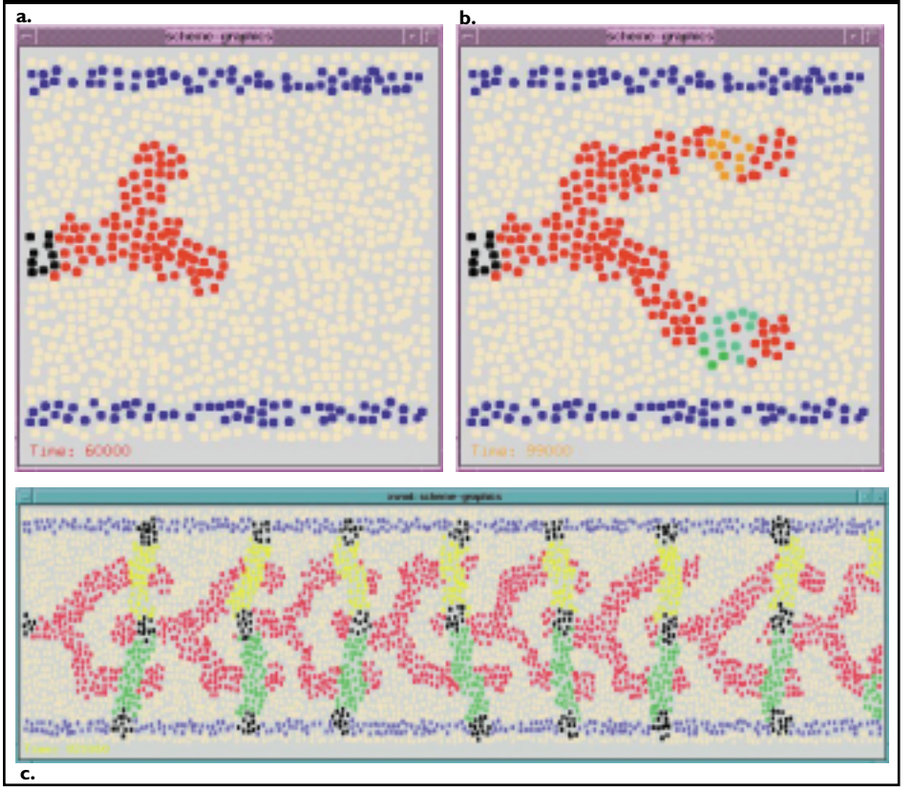
As an example, we can use diffusion waves to produce regions of controlled size, simply by having the particles relay the message if the hop count is below a designated bound. Once a region is generated in this way, we can use it to control the growth of other regions. For instance, two particles A and 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 on a biological metaphor, we might interpret this phenomena 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 closest to A (as measured by the A-wave). Our biological metaphor might interpret this situation by explaining that the region growing from B has a "tropism" attracting it toward A.

**> We turn to biology not just as a metaphor, but as an actual implementation technology for amorphous systems by means of "cellular computing."**

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 does not change much, so long as there are sufficiently many particles.

***From wave propagation to pattern formation.*** Based on this kind of cartoon caricature of biological development, the author Coore developed a programming language called the growing-point language (GPL) that enables programmers to specify complex patterns, such as the interconnect topology of an electronic circuit [2]. The specification is compiled into a state machine for the computational particles in an

**Figure 2. 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 toward "Vdd" and "ground." (b) The branches start moving horizontally and sprout pieces of "diffusion." (c) The completed chain of inverters.**




amorphous medium. All of the particles have the same program. As a result of the program, the particles "differentiate" into components of the pattern.

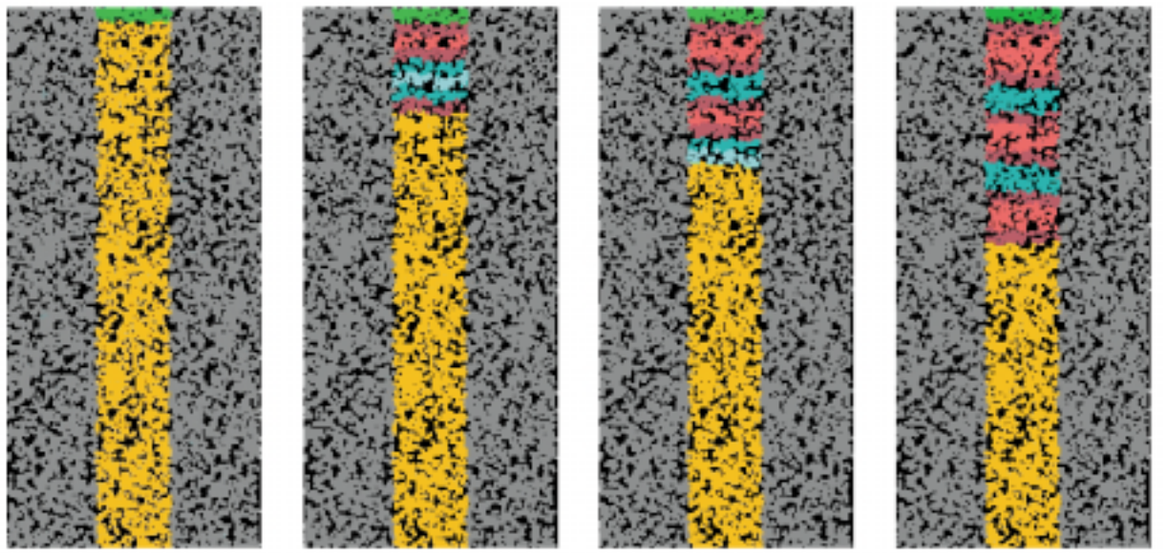
Coore's language represents processes in terms of the botanical metaphor of "growing points." A growing point is an activity of a group of neighboring computational particles that can be propagated to an overlapping neighborhood. Growing points can split, die off, or merge with other growing points. As a growing point passes through a neighborhood, it may modify the states of the particles it visits. We can interpret this state modification 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 or away from a source, or move in a way that attempts to keep constant the "concentration" of some diffused message. Particles representing particular materials may "secrete" appropriate diffusible messages that attract or repel specific growing points.

Figure 1 shows a fragment of a program written in GPL; the program defines a growing point process called `make-red-branch` that 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 identifies the particle as `red-stuff` and also causes the particle to propagate a wave of extent 5 hops that similarly converts nearby particles to be `red-stuff`. The growing point moves according to a tropism that directs it away from higher concentrations of `red-pheromone` in such a way that the concentrations of `pheromone-1` and `pheromone-2` are kept constant, so as to avoid any source of `green-pheromone`. All particles that are `red-stuff` secrete `red-pheromone`; consequently, the growing point tends 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.

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 2(a) shows the first stages of a pattern being generated by a program in GPL. For simplicity, we assume the horizontal bands at the top and bottom were generated earlier, and that an initial growing point is at the left. Growth proceeds, following a tropism that tries to stay equidistant from the top and bottom bands. After a short while, the initial growing point splits in two; one branch of growth is attracted toward the top, and one is attracted toward the bottom. Figure 2(b) shows the process somewhat further along; the two branches, which are repelled by short-range pheromones secreted by the top and bottom

**Figure 3. A pattern of alternating bands produced by marker propagation with the aid of Weiss's programming model.**



bands, start moving horizontally. They also change the kind of material they lay down.

Figure 2(c) shows the process evolved even further, producing an elaborate shape. That shape is the layout of a chain of complementary metal-oxide semiconductor (CMOS) inverters, where the different colored regions represent structures in the various layers of standard CMOS technology: metal, polysilicon, and diffusion. The program specifying the shape is only a few paragraphs long; the resulting state machine for the individual particles requires only about 20 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 the distribution of particles is sufficiently dense.

***Rules and markers.*** GPL is formulated in terms of abstractions that ultimately must be implemented by processes in the individual computational particles, which we assume are all programmed identically. The author Weiss has developed a remarkably convenient and simple language for programming the particles [12]. In Weiss's 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; rules are enabled by Boolean combinations of the markers. The enabled rules are triggered by receipt of labeled messages from neighboring particles. A rule may set or clear various markers; it may also propagate new messages. Messages contain counts that determine how far they diffuse, and markers have lifespans that determine how long their values persist. Underlying this model is a runtime system that automatically propagates messages and manages the lifespans of markers, so the programmer need not deal with these operations explicitly.

Figure 3 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 that generates this pattern, showing five 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)))
```

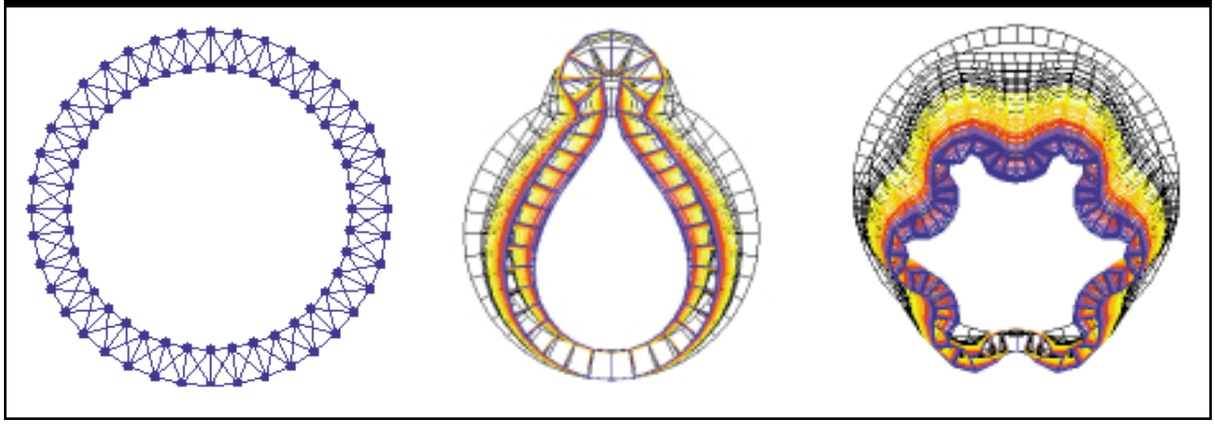
The first rule describes the reaction of a particle to receiving a message labeled `make-seg` specifying a `seg-type` (which is `red` or `blue`). If the particle has its `tube` marker set, and does not have its `red` marker

or `blue` marker set, it sets the bit for the specified `seg-type` and sends a `created` message that propagates for three 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 then 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 propagates for three hops. The fifth rule analogously causes `blue` particles to send `red messages`. The result is alternating red and blue bands along the length of the tube.

individual cells. A language of shapes, analagous to GPL, could allow programmers to generate prespecified macroscopic shapes in amorphous media by prescribing local shapes in individual particles.

Looking to biology also indicates that progress in amorphous computing may demand new approaches to fault tolerance. Traditionally, a system architect seeks to obtain correct results despite unreliable parts by introducing redundancy to detect errors and substitute for bad parts.[1] But in the amorphous regime, getting the right answer may be the wrong idea; it seems awkward to describe such mechanisms as embryonic development as producing a "right" organism by correcting bad parts and broken communications. Rather, the fundamental question is how to abstractly structure systems so we get acceptable answers, with high probability, even in the face of unreliability.

**Figure 4. Control of shape changes in a ring of cells, based on the mechanical cell models of [10]. Each cell has a simple programmed behavior and reacts to stresses in its neighbors.**



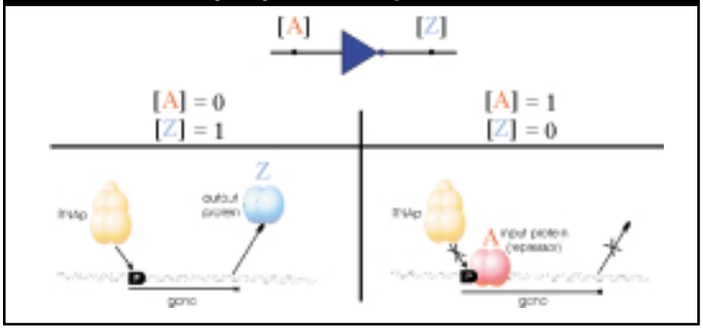
***More metaphors from biology.*** These sketches barely hint at the new primitives and organizational principles required for effective control of amorphous computing systems. Tapping the use of metaphors from biology has only barely begun.

A particularly fruitful source of inspiration from biology should emerge from the observation that even the most basic morphogenetic processes, such as gastrulation in embryo development, involve cell migration and deliberate changes in cell shapes. Figure 4 shows some evocative simulations by the author Nagpal, based on mechanical models of epithelial cells [10]. In Nagpal's model, each individual cell preserves its volume but has actuators (in this case, fibers) it can stretch or relax to change cell shape; it can also react to the stresses in its neighbors. The entire collection of cells bounds a fluid-filled cavity constrained to preserve volume as cell shapes change. The figure shows how different shapes and behaviors (such as elongation and invagination) appear as the result of changes by

***Physics and conservative systems.*** Physics, as well as biology, can be a source of new metaphors for amorphous computing. The mechanisms just discussed are 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 microscale errors in computation. In contrast, the fundamental processes in the physical world 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

[1] A compelling demonstration of this approach is the Hewlett-Packard Laboratories' Teramac, a massively parallel computer constructed from defective chips; it reconfigures itself and its communication paths to avoid the broken parts and compensate for irregular interconnections [5]. Although Teramac is built from conventional chips, its designers view it as a prototype architecture for designing nanoscale computers that would be assembled by chemical processes in which a significant fraction of the parts might be defective.

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




accrued. It is an especially challenging task to formulate processes manifesting exact conservation laws in such a way that imperfection in the implementation does not impair the exact conservation.

One approach, investigated by the author Rauch, is to simulate processes in terms of explicit discrete computational tokens of the conserved quantities [11]. 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. This scheme is essentially the program advocated by Donald Greenspan for making "particle models" of physical systems [4].

The use of discrete tokens to represent conserved quantities works only if we guarantee the tokens are not lost or duplicated if the communications network is imperfect or if the computational particles fail. One idea is to represent our tokens in a redundant 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 generate conservation robustly remain an important challenge.

## Cellular Computing

A major impetus for the study of amorphous computing is that the ability to program amorphous systems would greatly expand the set of physical substrates available to support information processing. One striking possibility is that we could create a programming technology based on living cells. Cells are isolated controlled environments housing complex chemical reactions. Cells also reproduce themselves, allowing creation of many copies with little manufacturing effort. The vision of cellular computing is to harness 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.

The authors Sussman and Knight have proposed a biochemically plausible approach to constructing digital-logic signals and gates of significant complexity within living cells [7]. This approach relies on co-opting existing biochemical machinery found naturally within cells as a basis for implementing digital logic. The "signals" in this logic system are represented by concentrations of certain DNA-binding proteins.[2]

The essential idea of cellular computing is to adopt the same strategy as is used in electrical engineering, where engineers create "digital abstractions" permitting the design of systems insensitive to variations in signal levels. The key to obtaining a digital abstraction is the existence of an inverting amplifier. This inverter must produce adequate noise margins, or ranges where signal variations in the inputs are not significant to the next stage in computation. Producing adequate noise margins 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 $A$ serving 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* coding for 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. If a molecule of $A$ binds to the operator site, the $A$ molecule prevents the docking of the RNA polymerase to the promoter site, thus preventing transcription of the gene. Assuming that proteins are scavenged and have a finite lifetime, the concentration of $Z$ varies inversely with the concentration of $A$ (see Figure 5).

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

[2]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 stochastic behavior provides a survival advantage by increasing the apparent diversity of a population [8]. For engineered systems, however, we would like the behavior to be as predictable as possible; this requires increasing the concentrations of the signaling proteins.

proteins, or proteins constructed from several subunits that must come together to bind to the DNA.

Given the ability to implement inverters in cells, arbitrary logic gates can then be realized as combinations of inverters. For example, the NAND logic function can be implemented as two inverters with different input repressors but the same output protein; the output is produced unless inhibited by both inputs. More complex components, such as registers that store state, can be constructed similarly, just as in standard electrical logic design [12]. 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 induce some other action within the cell, such as motion, illumination, enzymatic catalysis, even 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 intracellular chemicals.

***A research agenda for cellular computing.*** In principle, these foundations should be sufficient for implementing digital logic in cells. In practice, however, realizing cellular logic requires 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 the genetic regulatory mechanisms. Most important, 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 cellular functions in predictable and reliable ways.

Beyond our lack of knowledge of the biochemistry, the design of cellular logic circuits raises difficulties not present with electrical circuits. To prevent interference between gates, a different protein must be used for each unique signal. Therefore, the number of proteins required to implement a circuit is proportional to the complexity of the circuit. Moreover, because the different gates use different proteins, their static and dynamic properties vary. And unlike electrical circuits, in which 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 has to take explicit steps to ensure the signal ranges for coupled gates are matched appropriately.

One effort required for making progress in cellular computing is the creation of tool suites to support the design, analysis, and construction of biological circuits. One such tool—called BioSpice—is a simulator and verifier for genetic digital circuits [12]. BioSpice takes as inputs the specification of a network of gene expression systems (including 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 have to incorporate a database of biochemical mechanisms and their reaction kinetics, diffusion rates, and 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 such genetic engineering is to gradually transfer functionality from a wild type bacterial chromosome to one or more increasingly complex plasmids. As functionality is transferred, the gene sequences being transferred 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 is transferred 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" with clean modularity and well-understood structure. Such organisms could serve as substrates for precision cellular engineering.

## Toward Nanoscale Computing: A Fantasy

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 will be a good way to solve computationally difficult 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 subnanometer level. This kind of engineering requires 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 machine tools and DNA as control tapes.

We can envision applying this technology to the construction of molecular-scale electronic structures. 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 would be used to support fabrication of molecular conductive and amplification devices interconnected as the engineer requires. Proteins represent good candidates for scaffolding components; they are chemically and thermally stable and have exquisitely selective binding domains.

This perspective allows us to entertain the fantasy of nanoscale circuit fabrication in a future technology. Imagine a family of primitive molecular-electronic components, such as conductors, diodes, and switches, is available from generic parts suppliers. Perhaps we have bottles of these common components in the freezer.

Suppose we have a circuit to implement. The first stage of construction begins with the circuit and builds a layout incorporating the sizes of the components and the ways they might interact. Next, the layout is analyzed to determine how to construct a scaffold. The struts are labeled so they bind only to the appropriate electrical component molecules. For each 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 the 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 we want. The worker cells have complex programs, developed through amorphous computing technology. The programs control how the workers perform their particular tasks of assembling the appropriate components in the appropriate patterns. With a bit of sugar (to pay for their labor), the workers construct copies of our circuit we then collect, test, and package for use.

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

## References


1. Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Knight, Jr., T., Nagpal, R., Rauch, E., Sussman, G., and Weiss, R. *Amorphous Computing.* MIT Artificial Intelligence Laboratory memo no. 1665, Aug. 1999.
2. Coore, D. *Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer.* Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Dec. 1998.
3. Coore, D., Nagpal, R., and Weiss, R. *Paradigms for Structure in an Amorphous Computer.* MIT Artificial Intelligence Laboratory memo no. 1614, 1997.
4. Greenspan, D. *Particle Modeling.* Birkhauser, Boston, 1997.
5. Heath, J., Keukes, P., Snider, G., and Williams, R. A defect-tolerant computer architecture: Opportunities for nanotechnology. *Sci. 280,* 5370 (June 12, 1998), 1716–1721.
6. Kleinrock, L. and Silvester, J. Optimum transmission radii in packet radio networks, or why six is a magic number. In *Proceedings of the National Telecommunications Conference* (Birmingham, Ala., Dec. 1978), 4.3.1–4.3.5.
7. Knight, T. and Sussman, G. Cellular gate technology. In *Unconventional Models of Computation*, C. Calude, J. Casti, and M. Dinneen, Eds., Springer-Verlag, Berlin,1998.
8. McAdams, H. and Arkin, A. Simulation of prokaryotic genetic circuits. *Annu. Rev. Biophys., Biomol. Struct. 27,* (1988), 199–224.
9. Nagpal, R. *Organizing a Global Coordinate System from Local Information on an Amorphous Computer.* MIT Artificial Intelligence Laboratory memo no. 1666, Aug. 1999.
10. Odell, G., Oster, G., Alberch, P., and Burnside, B. The mechanical basis of morphogenesis: Epithelial folding and invagination. *Develop. Bio. 85* (1981), 446–462.
11. Rauch, E. *Discrete, Amorphous Physical Models.* Masters thesis, MIT Department of Electrical Engineering and Computer Science, May 1999.
12. Weiss, R., Homsy, G., and Knight, T. Towards in-vivo digital circuits. Presented at the DIMACS workshop on Evolution as Computation (Princeton, N.J., Jan. 1999).



**Harold Abelson** (hal@mit.edu) is a professor of computer science and engineering in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology in Cambridge, Mass.

**Don Allen** (dca@ai.mit.edu) is a research scientist in the Artificial Intelligence Laboratory at the Massachusetts Institute of Technology in Cambridge, Mass.

**Daniel Coore** (dcoore@uwimona.edu.jm) is a lecturer in the Department of Mathematics and Computer Science at the University of West Indies in Mona, Jamaica.

**Chris Hanson** (cph@ai.mit.edu) is a principal research scientist in the Artificial Intelligence Laboratory at the Massachusetts Institute of Technology in Cambridge, Mass.

**George Homsy** (ghomsy@ai.mit.edu) is a Ph.D. candidate in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology in Cambridge, Mass.

**Thomas F. Knight, Jr.** (tk@ai.mit.edu) is a senior research scientist in the Artificial Intelligence Laboratory at the Massachusetts Institute of Technology in Cambridge, Mass.

**Radhika Nagpal** (radhi@ai.mit.edu) is a Ph.D. candidate in the Electrical Engineering and Computer Science Department at the Massachusetts Institute of Technology in Cambridge, Mass.

**Erik Rauch** (rauch@ai.mit.edu) is a Ph.D. candidate in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology in Cambridge, Mass.

**Gerald Jay Sussman** (gjs@ai.mit.edu) is a professor of electrical engineering and computer science at the Massachusetts Institute of Technology in Cambridge, Mass.

**Ron Weiss** (weiss@ai.mit.edu) is a Ph.D. candidate in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology in Cambridge, Mass.



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.