Is it Possible to Build Computers from Living Cells?

Vijay Sharma

Building a computer out of living cells may sound like science fiction, but the foundations of such a technology are already being laid down. In the last 25 years we have seen astounding developments in microelectronics and computation, but even the most modern computers still have their weaknesses. First of all, current computer technology is fast approaching the limits of its capabilities, so there is considerable interest in finding newer, better technologies. Also, with current computers, a single fault in one circuit or a single line error in a program can cause the entire computer system to fail because these models rely heavily on the perfection of their parts and programming to function.

In contrast, biological systems demonstrate remarkable robustness. For example, simple and potentially faulty cells cooperate to form incredibly complex structures during embryological development, which are subsequently well maintained. For computer engineers, there are clearly important lessons to be learned from the natural world. The last 25 years have also seen remarkable advances in our understanding of molecular biology. Recombinant DNA technology is a powerful tool that already has a wide range of applications from therapeutics to agriculture. Armed with this new tool, engineers are now trying to create new computer technology that consists of living cells.

The Theory of In Vivo Digital Circuits

A brief summary of the principles underlying conventional computing and gene regulation is given in the following articles. This section describes how gene regulation can be used to make logic gates.

A simple theoretical example described by Weiss et al.[1] is a biochemical inverter (a NOT gate), as shown in Figure 1. When a repressor protein binds to an operator of the gene's promoter, RNA polymerase is prevented from transcribing the gene. Therefore, when the concentration of the repressor is high, the concentration of the gene product is low, and vice

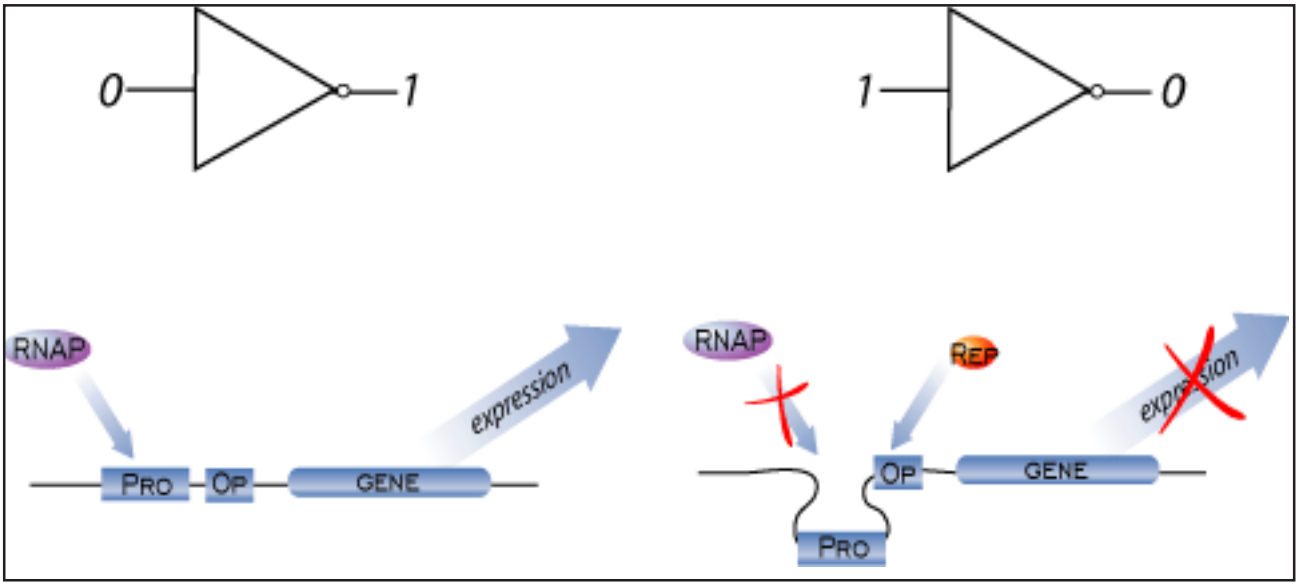

Figure 1. A repressor protein acting as a NOT gate.

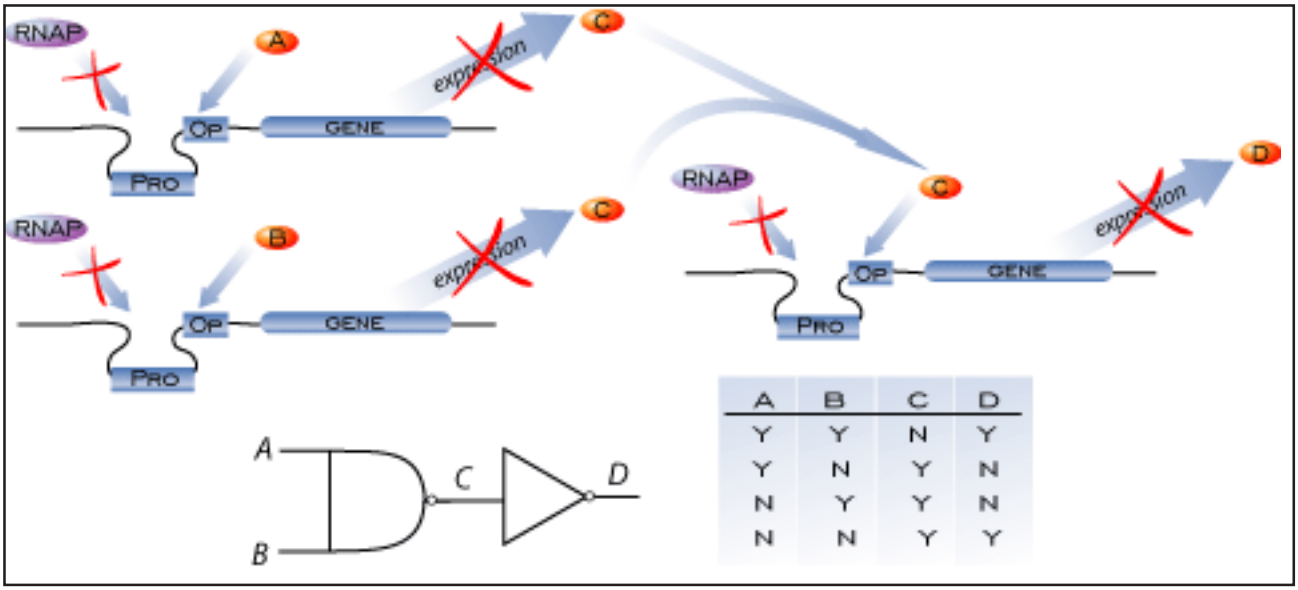

Figure 2. Repressor proteins acting as a NAND gate.

versa. Applying Boolean logic to the system, where high protein concentrations represent '1' and low concentrations represent '0', it can be seen that the above system is behaving like an inverter. Note that it is the concentration of protein that determines the signal.

Taking the theory a step further, these biochemical inverters can be linked together to make other gates[1]. If there are two biochemical inverters with different input repressor proteins A and B but the same output protein, C, the concentration of C will always be high unless A and B are both high. This is a NAND gate (Figure 2). Now suppose that C is itself the repressor of another gene that encodes protein D. When C is high, D will be low and vice versa. In other words, the output of the NAND gate is being fed into yet another inverter. This inverter inverts the outputs of the NAND gate, giving an AND gate. Notice that, in this example, the inverters are being interconnected to perform more advanced functions, and that proteins are responsible for these interconnections *In other words, the proteins are acting as wires*.

Figure 3 gives the example of an IMPLIES gate [2]. In this example, the input proteins are the inducer and repressor of a gene's promoter. Without the inducer, the system behaves like an inverter. However, when the inducer is present, it binds to the repressor and alters its conformation so that it can no longer bind to the operator, thereby allowing transcription to continue uninterrupted. The output is high unless the repressor is high and the inducer is low. The IMPLIES gate is believed to be used by cells as a sensor of environmental conditions and a receiver of messages from remote sources[2].

There are naturally occurring AND gates in cells which are used to detect messages from neighbouring cells [3]. In this system, the input proteins are the activator and inducer proteins of a gene. Only a complex of the two proteins can bind the operator and allow transcription of the gene to proceed. Thus the output is high only when both inputs are high, giving an AND gate.

The First In Vivo Circuits

A fundamental problem with biochemical processes is that there is considerable cell-cell variation in behavior. For an *in vivo* digital circuit to function, fluctuations in the concentrations of proteins that are involved must be controlled. In 1974, Savigau suggested that gene autoregulation by means of negative feedback loops is a natural mechanism by which the cell exerts such control [4]. Becskei and Serrano used this idea to construct a simple prototype circuit called an auto repressor[5].

In this study, a tetracycline repressor gene (*tetR*) was fused with the Enhanced Green Fluorescent Protein (*EGFP*) gene to give *tetR-EGFP* so that fluorescence could be used to reflect the degree of gene expression. In the natural system, the tetR protein

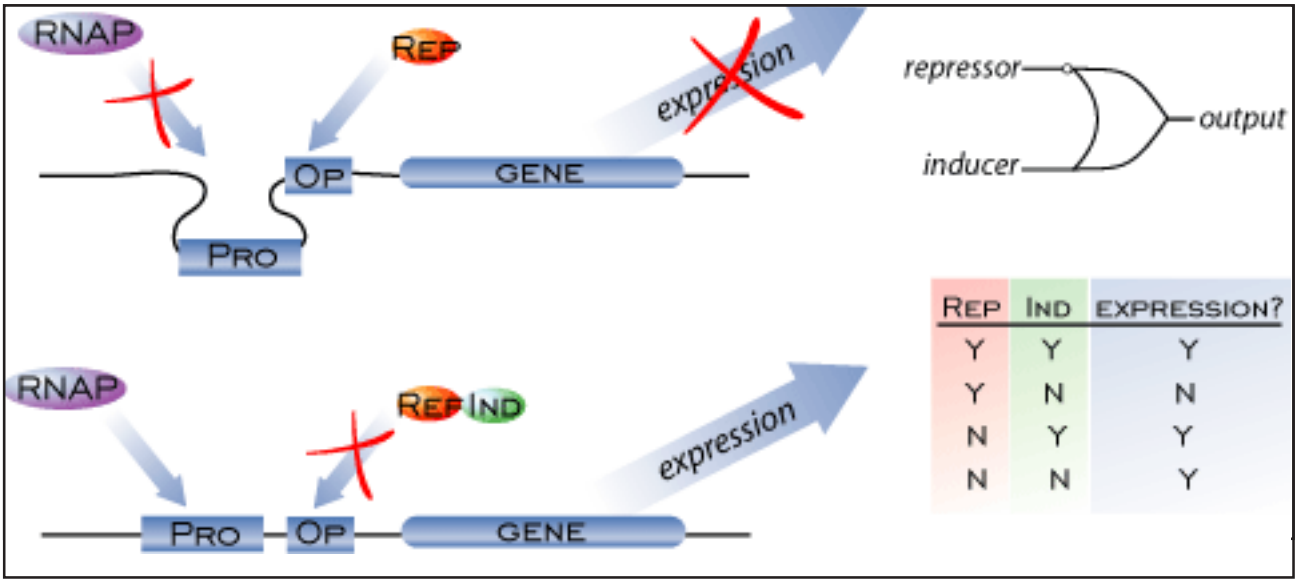

Figure 3. Repressors and inducers working together as an IMPLIES gate.

product binds to an operator of the *tetR* promoter to inhibit further transcription of the gene. Two groups of *Escherichia coli* cells were transformed with plasmids containing different DNA inserts. In the first group, the insert contained *tetRY24A-EGFP*, in which the *tetR* had been mutated to code for a non-functioning protein product that does not bind the operator. The second contained wild-type *tetR-EGFP*. In the first group, there was wide variation in fluorescence, reflecting wide variations in gene expression between cells. In the second group, where the negative feedback loop was in operation, the variation in fluorescence was three times smaller. The negative feedback effectively cleaned the output signal of this system, conferring stability and making it more suitable for use in a digital circuit.

Using similar methods, other very simple circuits have been built in *E. coli*. Gardner et al. constructed a toggle switch[6]. A toggle switch can exist in one of two states (1 or 0), and will only switch between these states in response to specific stimuli. In the absence of such stimuli, the state of the switch is maintained. The system used by Gardner et al consisted of two repressors, their promoters and inducers. In their system, they used a temperature sensitive repressor (*cIts*) and *lacI*, which is sensitive to IPTG (isopropyl bD-thiogalactoside). By varying the positions of these repressors and their promoters, four different versions of the network were made. The state of the system was determined by use of a fluorescent protein (1=fluorescence, 0=no fluorescence). In all four versions, the system maintained its state unless stimulated to change by a change in temperature or by the binding of IPTG to *lacI*; in other words, the system was behaving like a toggle switch.

Elowitz and Leibler made the first attempt at constructing a ring oscillator *in vivo*[7]. A ring oscillator consists of an odd number of inverters linked in series with the output of the last inverter fed into the input of the first one. Oscillators are very useful for any function that involves timing; they are used in digital clocks and watches, for example. Elowitz and Leibler's system consisted of three biochemical inverters linked in series. *CI*, *LacI* and *TetR* and their corresponding promoters were used. The state of the system was reported by using a fluorescent protein. Half of the *E.coli* cells transformed demonstrated oscillatory behavior, but the amplitude and frequency of the oscillations varied considerably between cells. For a ring oscillator to be of any practical use, it must oscillate at constant amplitude and frequency. Various studies are currently underway to find ways of achieving this *in vivo*.

Designing In-Vivo Circuits

Two general approaches have been employed in designing *in vivo* circuits. The first is 'rational design' which uses computer simulations to model the gene networks. The results of the simulations are used to identify modifications that can subsequently be made

to the *in vivo* system to produce the desired behavior[8]. This method was successfully used to improve the repressor and oscillator circuits discussed above.

The second approach that is employed is called 'directed evolution'. In this approach, a large number of random mutations are introduced into the DNA sequences of a gene network. Cells are transformed with plasmids containing the mutated gene networks. The cells are observed and those which exhibit the desired behavior are selected. DNA sequencing is then used to identify the mutations that made these cells exhibit the behavior[8]. As a proof of concept, Yokobayashi et al. used this approach to try and reproduce the toggle switch discussed above[9]. Starting with a non-functional system, random mutations were introduced into the *cI* gene and its repressor binding site. 50% of the colonies developed the desired behavior, and the various mutations that produced it were identified using DNA sequencing; while some of these would have been overlooked had they used the rational design methodology. The introduction of these mutations into the system resulted in a sharper response than was seen in the original experiment.

Directed evolution relies on the assumption that random mutations of limited regions will make some cells exhibit the desired behavior. As the size and complexity of the gene network grows, this will become less likely, and the number and combinations of mutations involved will become too big to handle. It is then necessary to use the rational design model to select the best regions to mutate. Directed evolution can then be used to obtain cells which exhibit the desired behavior. The combination of the two approaches is more likely to be successful at designing future circuits.

It is clear that, even in these small circuits, there are substantial problems in obtaining consistent results. As reviewed by Abelson et al, there are many practical difficulties to overcome[10]. We do not have a complete list of all repressors and the sites to which they bind. We do not have accurate or complete data on the kinetic constants involved. We still do not understand the metabolism or reproduction of cells well enough to accurately predict the effects of any interference with them. Simulator results employed in the rational design approach frequently disagree with what is actually observed because the simulator does not have complete and accurate information about the system being modelled[8]. Beyond this, the fact remains that this is a stochastic system and there will always be variation in behavior between cells.

There are other biochemical processes in the cell besides those that are used in the circuit. Not only does this produce background noise, making it difficult to discern the signal, but it can also directly interfere with the circuit. There are likely to be other gene regulatory processes besides those that are manipulated to create the circuits, and there are likely to be interactions between the proteins outside of the gene regulatory system[10]. For genetic logic to work, each signal needs to be represented by a different protein and each gate by a different gene or set of genes. As the number of genes and proteins involved grows, so does the potential for outside interactions.

Genetic logic is slow, having a maximum switching speed of 10^{-2}. By comparison, electronic logic has a minimum switching speed of 10^{6} . It is therefore unlikely that *in vivo* circuits using genetic logic will be able to solve computationally difficult problems by themselves[11].

Communication Between Cells

The key to building a computer based on cells may lie in controlling the behavior of groups of cells. The first step toward achieving this is to enable the cells to communicate with each other. It is clear that this happens in eukaryotic systems, but it is also known to occur in bacteria[12]. An example of this is quorum sensing.

In quorum sensing, a species-specific chemical signal is produced which diffuses across the population and enables each bacterium to sense the population density[13]. The marine prokaryote *Vibrio fischeri* has a quorum sensing system that produces bioluminescence[14]. Each bacterium secretes an autoinducer which diffuses into the surrounding media and permeates neighbouring cells. As the cells grow, the concentration of autoinducer inside and outside the cells increases. Once the concentration of autoinducer reaches a threshold level, it initiates a series of intracellular events which activate transcription of the luciferase and *luxI* genes. *LuxI* increases the

concentration of autoinducer, whilst luciferase activity produces the bioluminescence[15].

The quorum sensing genetic constructs from *Vibrio fischeri* were successfully transferred to *E.coli* to enable the bacteria to communicate[3]. The genetic circuits used were constructed so that some cells produced the autoinducer (sender cells) whilst others detected the presence of the autoinducer and expressed a fluorescent protein (GFP) in response (receiver cells). A droplet of sender cells was placed close to receiver colonies on an agar plate, and time-lapse green fluorescent photographs were taken. As the autoinducer diffused across the plate, the receiver colonies fluoresced. Weiss et al. have subsequently successfully built further complexities into the system to allow it to respond to two chemicals and to detect concentration ranges[3].

Cellular Computing

As discussed in the 'Some Basic Facts about Computer Science' sidebar, conventional computers are based on Von Neumann architecture, in which one complex processor, the CPU, carries out a task in a sequential manner (one thing at a time). However, there are advantages to having a parallel system. The idea of parallel computers is to have many processors instead of one; 'many hands make light work' as the proverb goes, so parallel computing has the potential to be faster and more efficient.

There is an emerging computational philosophy called cellular computing*, which takes parallel computing to the next level, and which could be applied to bacteria[16]. Sipper defines three principles of cellular computing:

1. *Simplicity:* The processor in cellular computing is defined as a cell. It can do very little on its own. The system performs complex tasks through the combined function and cooperation of many cells.

2. *Vast Parallelism:* Most parallel computers have at most 40-60 processors. 'Massively parallel' computers, as they are called, contain thousands or tens of thousands of processors. In this system, the parallelism is exponential, with 10^X processors.

3. *Locality:* Each cell communicates only with other cells that are close by, and each communication contains only a small amount of information. There is no single cell that has an overview of the computer system, and no single cell that controls the entire system i.e. there is no CPU or central processing unit.

Systems which use bacteria as substrates for engineering meet these three criteria for a cellular computing system. The major drawback once again is speed. The system developed by Weiss et al. relies on the diffusion of a chemical signal[8], so the computational speed of a cellular computer based on this system is limited by the rate of diffusion, which is slow. There are many other applications for Cellular Computing such as DNA computing, artificial automata and the search for a 'Quantum Computer', which are not discussed here.

Amorphous Computing

Assuming it is possible to build a cellular computer comprised of living cells, there is still the question of how to program it. Living cells are not silicon chips; they are unreliable parts, their interconnections are unknown, and both their behavior and interconnections vary with time. Moreover, in order to perform tasks, they must be made to behave coherently in prespecified ways. Researchers at the Massachussets Institute of Technology have called this 'Amorphous Computing'[17].

Abelson and Forbes propose that the system could first be programmed to perform a self-diagnostic operation in which it discovers which elements are operational and what the interconnections are[17]. The proof of concept for this approach is the Teramac machine. This is a massively parallel computer constructed from defective chips at Hewlett Packard

---

**The use of the term 'Cellular Computing' in the literature is confusing. Sipper (1999) uses it to describe a computing philosophy based on simplicity, vast parallelism and locality. Abelson et al (2000) use it to describe the efforts to design in vivo circuits. This article uses the term according to Sipper's definition.*

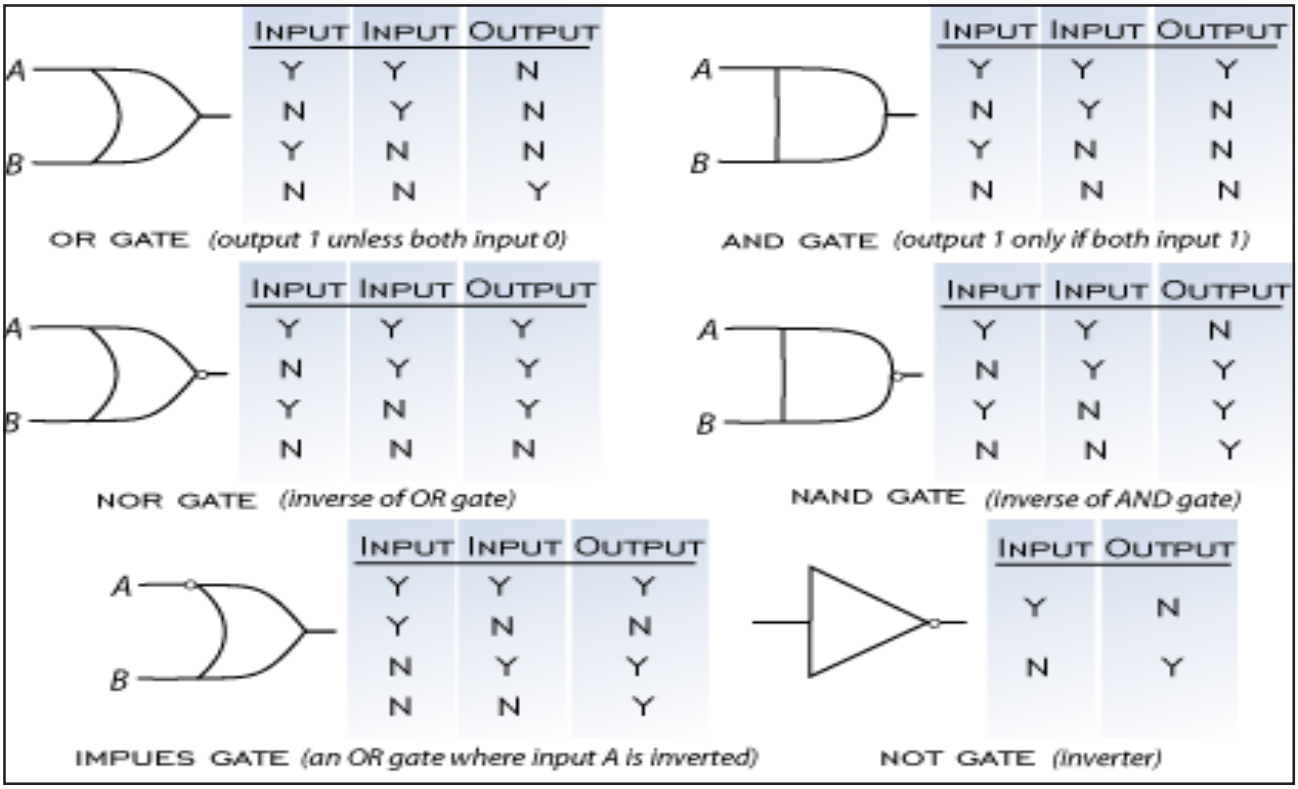

Figure 4. Logic gates and corresponding gene expressions.

(HP) laboratories. Teramac is able to perform a self-diagnostic to find the functional components and their interconnections, and achieves greater processing speeds than HP's best workstations despite containing more than 200,000 defects[18].

Another approach is to find fundamentally new ways of programming that do not require precise control over the computer hardware. In the natural world, there are many metaphors for this kind of programming. For example, a swarm of bees cooperate to form a collective hive. During morphogenetic processes, cells cooperate to form complex tissues and organs. Let us assume that the starting point is a cellular computer based on bacteria, which communicate by means of chemical signals. Abelson et al. describe a system by which this computer could be programmed[10]. A central sender bacteria sends a signal to each of its neighbours, which then send signals to their neighbours, and so on. This sets up a diffusion wave which spreads throughout the system. Complexities can be built into the system; for example if there are two waves from points A and B, bacteria could be programmed to relay the wave from A only if they have not seen the wave from B. Adding complexities like these will cause the waves to spread in particular directions, and to start generating patterns.

Coore developed the growing point language (GPL) based on this premise [10]. This language enables the formation of complex prespecified patterns by manipulating the diffusion waves. The language treats all operations as 'growing points', which is taken from the botanical metaphor of a branch growing from the stem of a plant, and then sprouting leaves and flowers. In this case, a program written in GPL is able to successfully direct the waves to grow, branch and sprout into a meaningful shape and form such as the interconnected topology of an electronic circuit. Another metaphor from biology that has been used in this way is embryological development. Nagpal developed a computer model based on the mechanics of epithelial folding[19]. The program generates cyst-like structures with different shapes but the same volume. A shape program could be written to instruct these 'cysts' to assume useful shapes. There are countless other biological metaphors that could be used to inspire programmers in this way.

Conclusion

The technology required to support a bacterial cellular computer is at a primitive stage of development. The stochastic nature of the system is problematic for circuit design in individual cells. However, at the programming level, this characteristic might be the key to designing a robust system. It is not clear what a bacterial cellular computer programmed using amorphous computing would be able to do. The processing speed of the hardware is likely to be slow, but amorphous computing may yield software that can overcome this drawback.

Bacterial cellular computers are unlikely to replace conventional computers in the foreseeable future. It is more likely that there will be specific niches in which this technology can excel, such as the design of programmable cells that can deliver drug doses directly to the site of action or act as vectors for gene therapy. On the other hand, conventional computing has already come much further than its founders predicted. Perhaps the search for a 'living computer' has some surprises in store for us too.

Some Basic Facts about Computer Science

During the 1800's, George Boole formulated a mathematical form of logic called Boolean Logic. In mathematical logic, the ambiguity of natural language is overcome by using an artificial language or set of symbols. Boolean Logic, as applied to computer science, is two state logic where 1 = true (signal) and 0 = false (no signal). Computer hardware can simply be regarded as a physical manifestation of Boolean Logic, with billions of 1's and 0's flowing through it to carry out the computer's functions.

Computer architecture describes what the components of a computer are and how they are connected. In the 1900's, John von Neumann divided computer architecture into three parts: the central processing unit (CPU), memory and a connecting device (data bus). The CPU is the controller of all other components. This layout is called Von Neumann architecture. Alan Turing then invented the Universal Turing Machine, which is a way of determining what can and cannot be computed using Von Neumann architecture. Between them, Turing and von Neumann laid down the foundations upon which all modern computers are based. Data stored by computers based on the above principles is most commonly binary data represented by electrical or electromagnetic signals (1=signal, 0=no signal). Such a system is digital, because the data is discrete. By contrast, systems in which the data is continuous are called analogue. Digital data is processed by means of logic gates. A logic gate receives fixed inputs (1 or 0) and gives predictable outputs. Tables showing the output given for particular inputs are called truth tables. Logic gates can be linked together to perform meaningful functions. They form the basis of all computer components. Examples of common gates and their truth tables are shown above.

Systems can be described as stochastic or deterministic. The behavior of a stochastic system is governed by probability and cannot be exactly predicted, whereas the behavior of a deterministic system can be predicted exactly. Conventional computers rely on the presence of a deterministic system.

If the computer hardware is to perform meaningful tasks, it needs to be given instructions by means of a computer program. Computer programs are sets of instructions written in programming languages which can be understood by the computer. After the instructions have been recognized, they are translated into machine code by compilers. The machine code then directs the hardware to carry out the instructions.

Gene Regulation in Prokaryotes

Gene expression refers to the production of a biologically active protein by a gene. Many biological functions require the coordination of several proteins, and the genes which encode these proteins are grouped together in clusters called operons. The initiation of gene transcription is controlled by two sequences located upstream of the gene. These are called promoters. RNA polymerase recognises these promoters as a signal to start transcription. This initiating step is the main site at which the rate of gene transcription is controlled.

The ability of RNA polymerase to recognize and bind promoters can be altered by accessory proteins.

Proteins that enhance or are required for this recognition of promoters are called activators, and proteins that inhibit it are called repressors. In many cases, these proteins exert their affects by regulating the accessibility of the promoter regions. They do this by binding to sequences adjacent to the promoters called operators. The activator and repressor can themselves be modulated by other proteins. An inducer is a protein that binds and inactivates the repressor. However, in some cases it binds and activates the activator. The precise mechanism of gene regulation varies between genes.

References

1. Weiss R., Homsy G., and Knight Jr T.F., Toward in-vivo Digital Circuits In: *Dimacs Workshop on Evolution as Computation*. Princeton, NJ, 1999.

2. Weiss R. Cellular Computation and Communications using Engineered Genetic Regulatory Networks. *PhD thesis*. Massachussetts Institute of Technology, (2001).

3. Weiss R., & Knight Jr T.F. Engineered Communications for Microbial Robotics, In: *DNA 6: Sixth International Workshop on DNA-Based Computers, DNA2000*. Leiden, The Netherlands, pp 1-16, Springer-Verlag, (2000).

4 Savigau M.A. Comparison of Classical and Autogenous Systems of Regulation in Inducible Operons. *Nature* **252**, 546-549 (1974).

5. Becksei A. & Serrano L. Engineering Stability in Gene Networks by Autoregulation. *Nature* **405**, 590-593(2000).

6. Gardner T., Cantor R., & Collins J. Construction of Genetic Toggle Switch in *Escherichia coli*. *Nature* **403**, 339-342 (2000).

7. Elowitz M. & Leibler S. A Synthetic Oscillatory Network of Transcriptional Regulators. *Nature* **403**, 335-338 (2000).

8. Weiss R., Subhayu B., Hooshangi S., Kalmbach A., Karig D., Mehreja R., & Netravali I., Genetic Circuit Building Blocks for Cellular Computation. *Natural Computing* **2**, 47-84 (2003).

9. Yokobayashi Y., Weiss R.,& Arnold F.H., Directed Evolution of a Genetic Circuit. *Proceedings of the National Academy of Science* **99**, 16587-16591 (2002).

10. Abelson H., Allen D., Coore D., Hanson C., Rauch E., Sussman G.J. & Weiss R., Amorphous Computing. *Communications of the ACM* **43**, 74-82 (2000).

11. McAdams H.H. & Arkin A., Simulation of Prokaryotic Genetic Circuits. *Annual Review of Biophysical and Biomolecular Structure* **27**, 199-224 (2002).

12. Hastings J.W. & Nealson K.H., Bacterial bioluminescence. *Annual Review of Microbiology* **31**, 549-595 (1977).

13. Bassler B.L. How bacteria talk to each other: regulation of gene expression by quorum sensing. *Current Opinion in Microbiology* **2**, 582-587 (1999).

14. Engebracht J., Nealson K.H., & Silverman M., Bacterial bioluminescence: isolation and genetic analysis of the functions from *Vibrio fischeri*. *Cell* **32**, 773-781 (1983).

15. Hanzelka B.L. & Greenberg E.P., Quorum Sensing in *Vibrio fischeri*: evidence that S-adenosylmethionine is the amino acid substrate for autoinducer synthesis. *Journal of Bacteriology* **31**, 5291-5294 (1996).

16. Sipper M. The Emergence of Cellular Computing. *Computer* **32**, 18-26 (1999).

17. Abelson H. & Forbes N. Amorphous Computing. *Complexity* **5**, 22-25 (2000).

18. Heath J.R., Kuekes P.J., Snider G.S. & Williams R.S., A Defect-Tolerant Computer Architecture: Opportunities for Nanotechnology. *Science* **280**, 1716-1721 (1998).

19. Nagpal R. Organizing a Global Coordinate System from Local Information on an Amorphous Computer. *MIT Artificial Intelligence Laboratory memo no.1666*, Aug. 1999.