Towards Computing with Proteins

Ron Unger[1]* and John Moult[2]
[1]*Faculty of Life Science, Bar-Ilan University, Ramat-Gan, Israel*
[2]*Center for Advanced Research in Biotechnology, University of Maryland, Rockville, Maryland*

ABSTRACT **Can proteins be used as computational devices to address difficult computational problems? In recent years there has been much interest in biological computing, that is, building a general purpose computer from biological molecules. Most of the current efforts are based on DNA because of its ability to self-hybridize. The exquisite selectivity and specificity of complex protein-based networks motivated us to suggest that similar principles can be used to devise biological systems that will be able to directly implement any logical circuit as a parallel asynchronous computation. Such devices, powered by ATP molecules, would be able to perform, for medical applications, digital computation with natural interface to biological input conditions. We discuss how to design protein molecules that would serve as the basic computational element by functioning as a NAND logical gate, utilizing DNA tags for recognition, and phosphorylation and exonuclease reactions for information processing. A solution of these elements could carry out effective computation. Finally, the model and its robustness to errors were tested in a computer simulation. Proteins 2006;63:53–64.**

INTRODUCTION

In recent years there has been significant interest in exploring the possibilities of biological computation. A large number of studies have investigated various ideas for using biological molecules to carry out various types of calculations and computations.[1–6]

Biological systems perform computations in living organisms on multiple levels, from the cognitive to the molecular. Examples range from the brain's ability to perform numerical calculations or analyze images to the immune system's ability to identify intruders. Other cellular activities, such as maintaining homeostatic levels of vital parameters and controlling expression levels of genes, are also forms of computation.

In contrast to these natural processes, the term *biological computation* usually suggests the use of biological molecules to carry out a general-purpose computation, that is, a computation that can be considered to be a digital computation outside the realm of the biological world. One of the ultimate goals is to build a computer, quite similar in its basic operation to current silicon-based machines, with its underlying hardware (or better said, wetware) based on biological components.

Considering the superb performance of silicon-based computers, one can question the need for biological alternatives. The advantages of a biological computer might be related to smaller size (Angstroms vs. microns), much lower energy consumption [the model we present here requires hydrolysis of several ATP molecules per basic logical operation (about 10^{-19} J), compared with more than 10^{-9} J per operation for current supercomputers[1]], and ease of production of the components by genetic engineering. However, biological systems have significant disadvantages compared to silicon-based systems, including slower speed of computation (gigahertz for silicon-based computers compared with microseconds to milliseconds for biological reactions), durability (most biological components have limited half-lives), and reliability (most biological reactions are prone to a nonnegligible error rates).

Thus, it is reasonable to suggest that the appropriate use of biological computational devices will be in environments where they naturally belong, for example, in medicine, where such devices can be encapsulated within a semipermeable membrane, and installed inside a living body. In such a device, inputs might be biological signals and the output might trigger biological processes. A biological device would also have the significant advantage of being able to use internal energy resources, like ATP molecules, rather then being dependent on external or rechargeable energy sources. An example might be an insulin regulation system, where the input would reflect glucose levels and oxygen demand, and the output would be used to trigger insulin production onsite. Such a biological system may offer several advantages over current continuous pump systems,[7] which are based on standard electronics. In this mode, a biological computer offers a useful combination of natural interface to biological processes with the strength of digital computation to achieve accuracy and precision.

Most of the current studies of biological computation have focused on DNA-based systems. In such systems, the underlying computational element is the hybridization of single-stranded DNA molecules to a complementary strand with high specificity. The computational paradigm takes advantage of the huge number of available DNA molecules

*Correspondence to: Ron Unger, Faculty of Life Sciences, Bar-Ilan University, Ramat-Gan, 52900, Israel. E-mail: ron@biomodel.os.biu.ac.il

Received 26 April 2005; Accepted 29 September 2005

Published online 24 January 2006 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/prot.20886

to carry out, in effect, a parallel exhaustive search of the solution space. This idea originated with the pioneering study of Adelman[1] on solving the Hamiltonian path problem. It has been shown to work on other NP-hard computational problems, for which no efficient polynomial time algorithm is likely. (For a recent example of solving a 3-SAT problem, see Braich et al.[3]). These studies clearly demonstrated that DNA-based computations are feasible. Nevertheless, these methods require the use of an exponential number (in the size of the problem) of molecules. While this exponential dependency may be unavoidable in dealing with NP-hard problems, it will lead to a very inefficient solution to more tractable problems, where a more direct and efficient approach might be more appropriate. In addition, these systems require specific encoding and implementation for each problem, and thus, in a practical sense, they do not offer a way of utilizing such procedures as a generic way to solve general computational problems.

In an advance from DNA-only based computation, the Shapiro group[4,5] demonstrated how a finite automaton can be built from restriction enzymes and ligases working on input presented as double-stranded DNA. The automaton was able to distinguish between strings with an odd versus an even number of input symbols. The computational devices described by Shapiro and colleagues[4,5] are finite automata. The authors consider this as a first step towards building a Turing machine based on biological components. A Turing machine[8] is a general computational device which is the abstraction of all other known digital computational devices. While the model and its biological implementation are elegant, Turing machines are not efficient computational devices, and programs written for Turing machines are long and cumbersome.

Recently, these authors[6] have demonstrated that their approach can be used in a biological and medical setup when they design a system where the inputs are mRNA molecules which are marker for diseases. After a digital computation which depends on the input, the output of the system is the production of a single-strand DNA molecule with therapeutic effects.

Various other possibilities for biological devices for digital computing have been explored. One direction is focused on designing biological wires. Braun and colleagues[9] demonstrated that silver-plated DNA strands can be used as conducting wires. RecA was used[10] to bind to DNA in a sequence-dependent manner and thus control the conductivity patterns of DNA molecules. This approach may lead to a hard wire (or wet wire) form of biological computing. Nevertheless, such a system will depend on a conventional power supply and regular electronic switching devices. In contrast, our aim is to design a biological computing system where the biological elements operate in a way similar to their natural biological counterparts, that is, to design a system based on interactions of diffusing molecules that compute by changing their biochemical state.

There is at least one well-described natural example in which biological reactions are used to achieve a switching effect: In the chemotaxis system, phosphorylation and methylation were shown to work together to achieve a switching effect on bacterial mobility.[11] This system is composed of several proteins with sophisticated feedback mechanisms. Recently, attention was drawn to demonstrating that switching networks can be designed and engineered. A significant achievement in this direction is described by Elowitz and Liebler,[12] where three transcriptional repressor systems were used to create an artificial oscillating network in *E. coli*. The network periodically, typically with periods of hours, triggered the synthesis of green fluorescent protein as a single cell read-out of its state. In Gardner and colleagues[13] a toggle switch was constructed from two repressible promoters arranged in a mutually inhibitory network. The switch can be flipped sharply between stable states using transient chemical or thermal stimuli.

Several possibilities for an elementary biology-based switching unit have been explored. One scheme uses rhodopsin molecules and their ability to change conformation in response to light. Such molecules have been shown to be particularly useful in building biological memory elements.[14] Another possibility that has been explored is to use a modified form of ribonuclease A, in which the molecular switch is constructed from a nonnatural amino-acid side-chain, containing an electron donor group and an electron acceptor group, connected to one another with a conjugated double bond bridge. The switching mechanism is based on azonium-hydrazo tautomerization, by which a charge separation induced in the excited state causing a rearrangement of the electronic structure of the molecule, resulting in the exchange of locations of single and double bonds. This rearrangement of bonds leads to different three-dimensional conformations of the switch, one of which blocks access to the enzyme active site, effectively providing an on/off switch.[15,16] While this switch design is very elegant, it is not clear how such elements can be hooked together to form a computing network.

In an inspiring paper, Bray[17] suggested the use of proteins as computational devices. Bray pointed out the diversity of roles proteins play in processing information in living cells, and suggested various possibilities for utilizing proteins to perform computational tasks. Our model explores in detail one of these possibilities: using proteins as logical gates and circuits. Logical gates are the basic components of digital computers, and building gates and circuits from proteins could open the way to general computing based on biological molecules.

In considering the practical steps necessary for the implementation of protein-based logical gates, one must address the following questions:

1. How can logical gates (i.e., switching) be implemented by a protein-based system?
2. How can wiring between gates be implemented?
3. What are the tokens of the computations? That is, how is the information in the computation conveyed through the process from input to output?
4. Because biological systems and reactions vary in time,

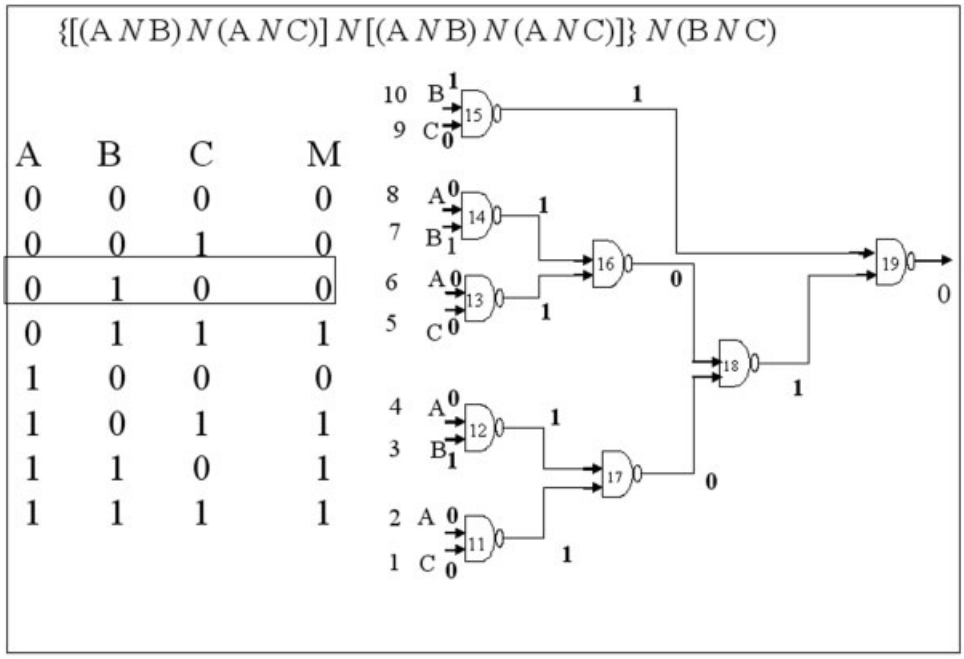

Fig. 1. The Boolean majority function expressed as a network of NAND gates. The output is true if at least two of the three inputs (A,B,C) are true. (**Top**) The function expressed in terms of the binary NAND operations (*N* stands for NAND). (**Bottom left**) The truth table of the function. For every combination of the input binary parameters A, B, and C the function output value is shown under the M column. (**Bottom right)** A logical circuit, using NAND gates that implements the function. Note that each input bit (either A, B, or C) is fed-in into the network via more than one input gate. A trace of the computation for the instance boxed in the truth table is shown. We propose to implement such networks by using protein complexes which function as NAND gates and DNA tags that function as connections between gates.

how can the timing of the computation(s) be synchronized?

5. What are the expected errors in the process and how can these errors be contained?
6. How can the design of such a computation device be automated such that for any regular electronic circuit, its biological equivalent can be efficiently constructed?

In this article, we present a detailed model that addresses these questions, and discuss a computer simulation that tests the model.

In selecting the biological mechanism on which to base a computational device, we have considered the following issues: First and foremost, we need a system in which the switching is binary, and can toggle between two well-defined and well-separated states. Second, we prefer a system in which the basic element is a single molecule and not itself a network. Third, we prefer a system which utilizes proteins that have a natural function close to that required in the computation. Thus, it should be possible to tap into the repertoire of natural reactions in order to find the most suitable starting point for the design. Fourth, we require a system in which reactions can be easily chained together to achieve a flow of computation.

We first schematically describe the system we have designed, and then turn to a more detailed account of how the biological reactions chosen can be used to implement the scheme. These reactions are certainly not the only possibilities for implementing a general design of a computational network, and we also suggest some possible alternative mechanisms. Regardless of the actual reactions that are ultimately used in engineering a practical implementation, the model proposed here is a general one, in the sense that the same design can be used to perform any logical calculation via biological computation.

THE MODEL

Boolean Algebra and Choice of Gates

Boolean algebra deals with calculating truth values (TRUE or FALSE) of logical statements and is the underlying mathematical tool of any digital circuit. Every Boolean function can be expressed using the two gates of AND and NOT (or OR and NOT). However, to keep our design simple and uniform, we prefer to use logic that is based on a single universal gate that can be combined to express any function. The NAND gate is one such gate (XOR is the other). A NAND gate outputs 1 unless its two inputs are 1, in which case its output is 0. NAND is universal because it can express the standard gates, AND and NOT:

$$\text{NOT A} \equiv (\text{A NAND A})$$

$$\text{A AND B} \equiv (\text{A NAND B}) \text{ NAND } (\text{A NAND B}).$$

An example of logical network based on NAND gates is shown in Figure 1. This circuit will be discussed in more detail later, but its general features are common to all

logical circuits. Note that each gate has two input and one output ports, gates are wired in layers in such a way that the output of one gate is the input to the next gate, and the computation propagates from the input layer to the output element according to Boolean arithmetic, as implemented by the gates.

An Abstract Molecular Computation Scheme

In this section we describe, in abstract terms, the design of a biomolecular system that is capable of carrying out a computation that follows the logic of a Boolean circuit based on NAND gates. In the next section we will discuss specific biological reactions that can implement this design.

A single species of molecule will carry out the NAND gate logic, and thereby form the basis of the computation. To achieve this goal, the molecule must perform three tasks. The first is ***recognition***, that is, only the appropriate molecules may recognize and interact with each other. For this purpose, the design requires that each molecule have three recognition sites, two to recognize incoming molecules (i.e., input sites) and one to recognize the target molecule (i.e., an output site). The second task is ***synchronization***, that is, interactions must occur only between active molecules, those that are at the appropriate stage of the computation. To enable synchronization, the design requires that the recognition sites are initially blocked or inactive, and become active at a desired time during the computation. The third task is the actual ***computation***, that is, the change in the state of the molecules such that they will carry the correct logical value. To this end, we require a two-state mechanism that provides reversible modification of the molecule, such that one state represents 0 and the other state represents 1.

The basic element will therefore be a molecule that includes two catalytic domains, performing the tasks of activation and computation, and three recognition tags to enable recognition of the molecule by the specific computational elements with which it must interact in the logical circuit. The tags are encoded such that an output tag of a given element will recognize the input tag of its designed target. Thus, a pair of complementary tags provide a "wire" connecting the output of one gate with the input of another. All molecules in the network have the same catalytic domains, but have different tag sequences that uniquely define their input and output interactions. Two tags are used to define input interactions and one is used to define an output interaction. The input tags are always active and ready to receive a signal. The output tag is initially blocked. This block is removed once the molecule acquires its logical value (either 0 or 1). The computation will take place when two active input molecules bind the element, and will depend on the logical state of the input molecules. Following the logic of a NAND gate, the output molecule will obtain the value of 0 only when both input molecules are 1; in all other cases the output molecule will assume the value of 1.

Computation takes place in a solution containing all the required molecules, which are allowed to diffuse freely. Molecules will collide randomly, but only molecules that have complimentary tags will associate to form complexes which can transfer information. At the start of the computation, only the molecules that represent input to the system, that is, the first layer in the circuit, have an accessible output tag. Thus, only these molecules can interact effectively with their targets. All other molecules will have their output tags inaccessible, preventing them from interacting with additional elements before they receive a valid input signal; however, their input tags will be accessible, making them available to receive signals from molecules that have already been activated. In subsequent phases, only molecules that have acquired an accessible output tag can interact further.

Because each molecule has two accessible input tags, three molecule complexes will form. Within each complex, the computation and the synchronization steps will take place. The computation will set the logical state of the output molecule depending on the logical state of the two input molecules according to the logic of a NAND gate, and synchronization is provided through activation of the output molecule by making its output tag accessible. Over time the output molecule will diffuse and find its target molecule, allowing the process to continue until the logical state of the molecules in the final phase of the computation is determined.

As mentioned above, each basic element is characterized by the specific combination of its input and out tags. Multiple identical copies (on the order of 10^9, see below) of each element are present in the system. Thus, for example, in Figure 1, if element number 15 is required to interact with element number 19, any one of the active copies of element number 15 can interact with any one of the copies of elements 19, based on their complementary tags. As we discuss below, this parallelism can be used to facilitate error detection and correction.

The Biological Implementation

The model proposed above is based on general ideas but must be implemented using specific biological processes and reactions. In this section, we suggest one set of reactions that could, in principle, carry out the tasks required. We discuss the feasibility of these reactions in a separate section below.

Recognition will be achieved through hybridization of complementary DNA tags. Each tag is composed of a single-stranded DNA oligomer that is covalently attached to the protein part of the molecules. Binding of complementary tags will provide a localization effect, effectively enabling the logical computation for a single gate. In a sense, these tags are used as wires in this diffusive network. To achieve ***synchronization***, the output DNA tags will be blocked by a complimentary DNA strand, which is removed only after the associated gate is formed. The removal is achieved by activation of an appropriate exonuclease that will digest the blocking strand, thereby exposing the output tag and render it active.

To achieve the ***computation*** we propose to include a phosphorylation domain that is capable of performing a

conditional phosphorylation reaction, such that the logical state of the output molecule (i.e., whether phosphorylated or not) will be dependent on the phosphorylation state of the two input molecules. The phosphorylation state of the input molecules is configured to reflect the desired input logic. See Figure 2 for a schematic view of the basic computational element.

Computation takes place in a solution containing a mixture of all the molecules, which are allowed to diffuse freely. Molecules will collide randomly, but only molecules that have complimentary DNA tags will associate by hybridization to form complexes with significant half lives. The input molecules, that is, the first layer of the circuit, are set as follows. Their output DNA tags are exposed, rendering them active; their input tags are covered preventing them from interacting. Their phosphorylation state reflects their logic value; we arbitrarily set phosphorylated molecules to the logical value of 1, and dephosphorylated molecules to 0. All other molecules will have their output tags inactive, preventing them from interacting with additional elements before they receive the proper input signal, and their input tags exposed making them available to receive signals from active molecules. The phosphorylation state of the non-input elements is set initially to be dephosphorylated, that is, logical value 0. In subsequent phases only molecules that have acquired an activated output tag (e.g., with the blocking oligomer removed in previous phases), can interact further.

Note that the biological gate is somewhat different from an electronic gate. An electronic gate is a single element that receives two input signals, calculates the appropriate logical function and produces an output signal. The biological gate is actually a complex that includes three molecules, two input molecules and one output molecule.

Each complex forms in two stages. First, one input molecule hybridizes to the first input tag of the target element to form an inactive complex. Upon binding of the second input molecule to the second input tag, the computational complex will became active and perform the following reactions: The two input molecules will interact to form an active dimer that will phosphorylate the target molecule if so required by the NAND logic, that is, the target (output) element will be phosphorylated **unless** both its input molecules are phosphorylated.

In addition, an exonuclease reaction will be activated as a result of association of the two input molecules, digesting the cover of the output tag and leaving the tag exposed, thus making the output tag available for hybridization with the input tag of the next element in the circuit.

A schematic diagram of the interaction is shown in Figure 3.

After some time, the computational complex dissociates, and the activated output molecule seeks its own target, encoded by the complimentarity of its output tag with the sequence of an input tag of another molecule. (Note that we do not require dissociation to precede formation of the subsequent complex formation). The process will continue, through successive layers of gates, until the final output gate molecules are processed. The result of the computation may then be read from the phosphorylation state of these output molecules.

There are many identical copies of each computational element, that is, molecules that have the same combination of input and output tags. These molecules are interchangeable and each copy can interact with any copy of its designated target molecule. Thus, parallel computation of the same circuit is carried out by a large number of molecules. This redundancy can be utilized as described below to achieve a high degree of robustness in the computation.

An Example — The Majority Circuit

To make the model more concrete, we discuss a simple example, a biological implementation of a circuit that calculates the majority function of its three inputs, discussed earlier. Figure 1 gives the truth table and the logical design for this circuit. The output is 1 if at least two of its three inputs are 1, otherwise it is 0. While this is a very simple calculation that can be performed by many analog processes, we will not exploit this simplicity. As will be clear from the presentation, the same approach can be used to implement any logical circuit, regardless of its complexity.

This circuit has 19 elements. Elements 1 to 10 are inputs consisting of molecules similar to the rest of the computational elements, the only difference being that they have preset phosphorylation states, reflecting the required logical values. Gate 19 is the output gate. Thus, the system consists of 19 elements, each with the same protein component, capable of performing the phosphorylation and the exonuclease reactions, but each carrying different DNA tags. There are many copies of each of these elements present in the system.

For the boxed instance in Figure 1, variables A and C have the value of 0, and B has the value 1. Hence, the computation is initialed by setting input elements 2,4,6,8 corresponding to A, and 1,5,9 corresponding to C to a dephosphorylated state while elements corresponding to B (3,7,10) will be phosphorylated. All non-input elements will be dephosphorylated. The input elements (1–10) have active output tags. All the other elements have their output tags blocked and their input tags active. Wiring is achieved by providing appropriate pairs of complementary tags. For example, the output tag of element 1 is complimentary to one of the input tags of element 11, and the output tag of element 11 is complimentary to the input tag of element 16, and so on. In the first computation layer, no element has its two input elements in state 1 (i.e., phosphorylated), so that phosphrylation reactions take place in all elements. In the second layer the inputs to element 16, provided by the output of elements 11 and 12, are both 1. Similarly, the inputs to element 17, provided by the outputs to elements 13 and 14, are both 1. Thus, elements 16 and 17 will not be phosphorylated. The computation propagates in this way until the final output element (number 19) forms a complex with elements 15 and 18. Activation of the output tag of element 19 signifies completion of the computation, and the result may be read off by,

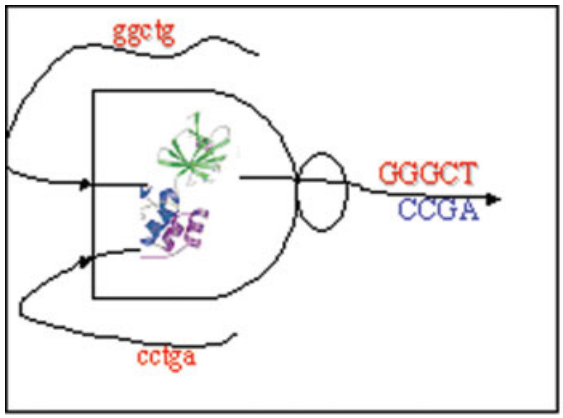

Fig. 2. A schematic view of the basic computational element. The protein molecule has two enzymatic domains to facilitate activation and computation. We propose these to be an exonuclease domain and a kinase domain, respectively. In addition, each molecule has three DNA tags to provide recognition properties, two for the input and one for output (capitol letters). The output tag will be initially blocked by a complimentary oligomer, rendering it inactive until the appropriate stage of the computation.

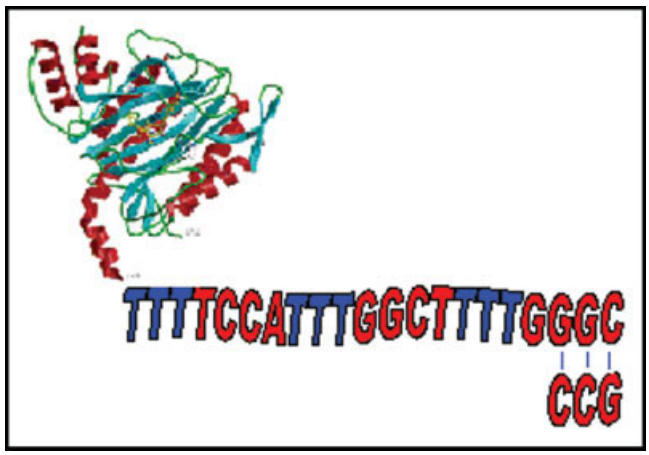

Fig. 4. A schematic view of recognition tags attached to the protein part of the active molecule. Two input recognition sequences are shown (red) separated by linker sequence (blue). The output tag is covered by a matching strand which blocks accessibility until an exonuclease is used to digest the cover. The exonuclease will only digest the DNA cover and will leave the tag DNA (or PNA, see text) intact.

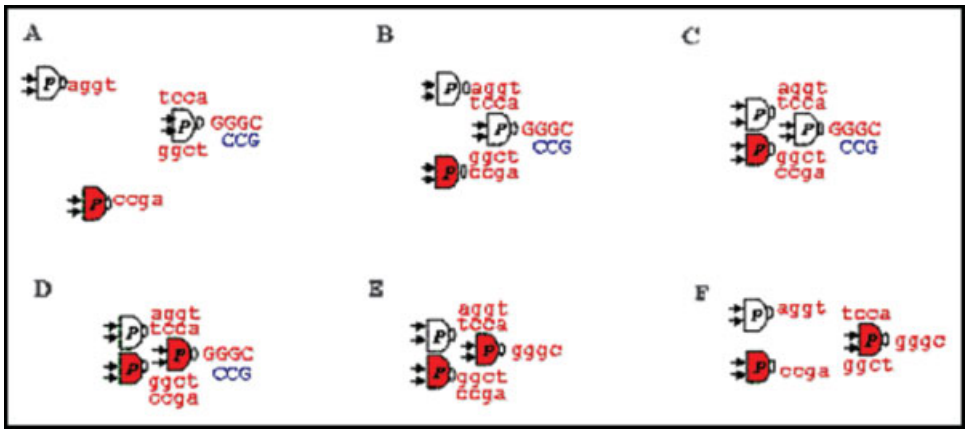

Fig. 3. A schematic view of activation of a gate complex. (**A**) The process starts with two molecules with complimentary active tags (shown in lowercase letters) defusing freely in search of the appropriate target. Note that the output tag of the target molecule is blocked by a matching oligomer, and that the target molecule is set by default to the dephosphorylated (white) (i.e., set to the 0) state. (**B**) The tags of the input molecules hybridize to the input tags of the target. (**C**) Once the two input molecules are tethered to their target, localization causes these molecules to form an active dimer. (**D**) The conditional phosphorylation reaction, representing the NAND gate: Only if both input molecules are phosphorylated (in the 1 state, red) is the target molecule not phosphorylated. In all other combinations, like the one shown here, the output molecule is phosphorylated (i.e., set to the 1 state, red). (**E**). Formation of the complex (regardless of its phosphorylation state) activates the output tag as a result of exonuclease digestion of the blocking oligomer, exposing the single stranded output sequence. (**F**) The output molecule diffuses in search of its own target.

for example, examination of a fluorescent probe attached to that tag.

Feasibility of the Model

While the ultimate proof of the feasibility of a design is its effective implementation, the basic biochemical molecules and reactions needed to implement this model are mostly within the realm of current protein engineering capabilities.

Volume and concentrations

In order to minimize the formation of incorrect interactions the system will be established at the most dilute concentration possible. We assume that a prototype system might have a volume of 1 mL. As we discuss below, we intend to utilize redundancy in terms of multiple copies of each element to allow for error correction. To achieve this purpose, we consider 10^7 copies of each molecule to be reasonable. Assuming a system size of the order of 1000 gates, this will amount to 10^{10} molecules in a volume of 1 mL, which is a concentration on the order of 0.1 nM, a suitably dilute system.

Recognition Tags

The design calls for the use of single-stranded DNA tags to produce high specificity associations between the appro-

priate gate elements. We are not aware of the use of DNA tags to associate pairs of proteins, but the necessary chemistry for fusing protein and DNA is established, for example, by covalent attachment of the DNA strand to cysteine residues.[18,19] The association constants between complementary DNA strands are also well understood and predictable,[20] forming the basis of temperature dependent melting, as used in PCR and cloning reactions. Synthesis of DNA oligomers is a routine process, partly because of these applications. For our prototype system, tags of about 20 base pairs having binding constants in the 0.1 p*M* range, would ensure almost complete complex formation at the 0.1 n*M* concentration of gate elements proposed. As discussed below, selectivity of tag binding is achieved by ensuring at least eight mismatches between any pair of noncoupled tags.

An alternative is to use PNAs (peptide nucleic acids), which have a peptidelike backbone with side-chains mimicking DNA bases. PNA is recognized by DNA binding proteins and as a single strand can hybridize to complementary DNA or PNA molecules with high specificity.[21,22] PNA tags can be attached via a peptide bond to the termini of proteins, as well as via a cystine side chain.[22] An advantage compared with DNA is that PNA is not digested by nucleases. Note that DNA would still be used to make the covers that initially block the output tags, so that the exonuclease will be able to digest them at the appropriate time.

Each gate element carries three tags. These tags can attached to three different sites, as shown schematically in Figure 2, or more conveniently, fused together with short linkers and attached to one terminus of the protein (see Fig. 4). An advantage of this arrangement is that it permits easy automated translation of any logical circuit into its biological equivalent (see below).

Activation of output tags

The output DNA tag of each gate must be blocked from premature association with the element to which it is wired until the logical operation of the gate is complete. The proposed mechanism for ensuring this is to block the tag with a complementary oligomer until activation is required. Activation is carried out by an exonuclease, which digests the complementary blocking strand. Correct timing is achieved by employing a nuclease that is functional only as a dimer. The dimer interface must be engineered so that significant dimer formation only occurs when the gate complex has been formed. Several commercially available dimeric or tetrameric exonucleases, for example, Lambda exo and exo III,[23] have the ability to digest double-strand DNA, leaving a single intact strand. In fact, a similar idea is used in a product called TaqMan that is designed for quantitative PCR measurements in which an oligonucleotide is digested by a DNA polymerase.[24]

Gate logic

Logical states are represented by the phosphorylation state of the protein components. There are two possible conventions: phosphorylation represents 0, or phosphorylation represents 1. NAND logic can be implemented for the former convention by employing a phosphatase activity (removal of a phosphate from the output element if both input elements are phosphorylated) or a kinase for the latter convention (addition of a phosphate to the output element unless both input elements are phosphorylated). In biology, control mechanisms seem to rely much more frequently on kinases than on phosphatases, and so there is a much richer choice of possible kinase enzymes to employ. Thus, we have decided to base the model on kinase activity. We require a kinase system with the following properties: First, the enzyme must be active only as a dimer. Second, the dimeric form of the enzyme must be able to phosporylate other monomers. That is, the dimer of molecules should add a phosphate to a monomeric form of the same molecule. Third, it exerts negative control, that is, it is inactive only when both subunits are phosphoraylated.

Because many kinases are active as dimers, and many are autocatalytic, the first two requirements are relatively easy to achieve. The third one poses an engineering challenge. Negative control of kinases (i.e., kinases that work only when they are not phosphorylated) seems to be relatively rare. It is much more common for kinases to be activated by phosphorylation. However, at least one such case, which might serve as a starting point for the design, has been reported,[25,26] DRP-1 of the DAP-kinase family of Ca^{2+}/calmodulin (CaM)-regulated Ser/Thr kinases. These molecules function as positive mediators of programmed cell death. The protein combines two of our desired properties — it is active only as a dimer, and it is most active when unphosphorylated. When the two subunits are phosphorylated (on Ser^{308}) there is a very significant reduction of its phosphorylation activity. While Ser^{308} is autophosphorylated by the enzyme, its primary phosphorylation target is another protein, a myosin light chain (MLC). Additional control is provided by calcium-dependent calmodulin binding. Thus, substantial protein engineering would be required to produce a suitable kinase, starting from DRP-1, or from other proteins. Although challenging, we believe such engineering is possible.

The suggestion presented here concentrates on using NAND gates because NAND is a universal gate that can be used as a single type of gate needed to implement any logical computation. It might turn out that it is simpler, protein engineering wise, to design two different biological molecules, one that emulates for example the function of a NOT gate and one that emulates the function of an AND gate. (NOT and AND gates, taken together, allow universal computation.) Such a pair of gates would require a different cascade of signaling events than the one described here.

Activation by localization

A key feature of the design is high, effective local concentration of molecules as a result of the complementarity of the tags. The enzymatic reaction is tuned such that the tag tethered molecules will be highly active, while

freely diffusing ones will be essentially inactive. This is achieved by control of dimerization. Binding of tags to the complementary sequences on a target molecule increases the local effective concentration. Assuming a protein diameter of about 50 Å, and the connecting DNA tether to be of about 250 Å (three tags of about 20 bp each and linkers), the two molecules will be contained within a sphere of about 10^8 Å^3. The effective concentration will then be of the order of 10 μM. This is 100,000-fold higher than that of the free molecules in the solution (0.1 nM). A dimer association constant of 1 μM will therefore ensure almost complete formation of active enzyme for tag hybridized molecules. At the same time, it would guarantee a very low amount of dimerization for untagged complemented molecules: At 0.1 nM concentration, only 1 in 10,000 molecules will be in dimeric form at any time at equilibrium. Similar localization principles have been evoked to explain enzyme rate enhancements,[27] and form the basis of methods of detecting naturally occurring protein–protein interactions, for example, in yeast two-hybrid assays.[28]

Timing within a gate

The output tag of a gate must not be activated until its logical operation is complete. That is, the kinase must add a phosphate to the output element, if required, before the exonuclease exposes enough of the output tag for association with the input of the next gate to occur. Indeed, typical turn over rate for kinases are around 100 to 1000 per second (see, for example, Rose and Dube[29]) while an exonuclease would cleave a mask of 20 bp in about 1 to 5 s.[30]

Speed of computation

How rapidly can these circuits carry out a computation? As mentioned above, digestion of the mask can be completed 1 to 5 s. With 10^7 copies of each element in a volume of 1 mL, collision rates are significantly faster than that. The phosphorylation rate is also significantly faster. Thus, the exonucealse step is rate limiting. So, a system of a thousand gates, which would have about 10 layers, is expected to complete computation in less than a minute.

Initialization and Resetting of the System

For a prototype system, we propose the following initialization and resetting steps: A solution of identical untagged and unphosphorylated monomeric molecules is prepared at the required concentration for computation. An aliquot containing sufficient molecules for the input layer is removed. The necessary tags are synthesized in two batches — one containing tags of the input layer (where the input tags have to be blocked and output tags exposed), the other containing all other tags (where the output tags should be exposed and output tags blocked). The appropriate tags are blocked to prevent premature hybridization between tags. The tags, already appropriately blocked, are introduced into the solutions under ligating conditions and are attached to the protein molecules.

Input elements to be initialized to the logical state 1 are identified by means of their common input tag sequence. A convenient mechanism would be to immobilize the appropriate set of complementary tags on a bead. The bead is then used to extract the corresponding elements from the solution of input layer elements, and to introduce them into a solution containing activated kinase molecules. Following phosphorylation, the elements are released back into the input layer solution by elevating the temperature to melt the tag complexes (as in a PCR reaction). Similar immobilization methods have been developed for DNA microarray preparation. This procedure facilitates resetting of the input layer for subsequent computations with different input values.

Computation is begun by adding the input layer solution to the main solution. The unmasked output tags on the input elements will permit formation of the first layer of complexes. Thereafter, the computation will run to completion automatically.

Activated output molecules can be isolated using the appropriate complementary tags mounted on a bead. Mass spectroscopy then provides a convenient means of determining their phosphorylation states.

For each logical formula, a new combination of tags needs to be assembled. This is not needed for another computation of the same formula with different input values. A computing solution can be prepared for another round of computation by resetting all elements to the unphosphorylated state (using a phosphatase immobilized on a bead), introducing a new set of masking tags, and setting input element phosphorylation states as required.

Possible Sources of Error

One of the most obvious problems to be addressed in considering computation using biological reactions is that of error in the process. While we have selected reactions that are of inherent high fidelity, biological processes are never error proof, and reactions might occur between incorrect reactants or produce an incorrect product.

The computational model presented here is sensitive to such problems because it is a tight computation, in the sense that an error in the outcome of any reaction in the circuit may lead to an incorrect result presented at the output gate.

Several types of error are possible, and could arise in recognition, synchronization, or computation:

Recognition: Hybridization of unmatched tags. In principle, the specificity of tag recognition can be made as high as desired, by increasing the length of the tags. As noted earlier, quite short tags (approximately 20 bases) are sufficient to achieve an appropriate binding constant. This size will still enable reliable distinction between tags and prevent cross-hybridization. Coding theory (see, for example, Van Lint[31]) provides upper and lower limits to the number of code words that differ by a given number of mismatches. For example, for tags of 20 bases, a lower bound on the number of different tags with at least eight mismatches to any other tag is over 5000 tags and the upper bound is over 33,000,000 tags. Even the lower bound would be enough for our prototype system.

Synchronization: Dimerization to form an active complex may occur spontaneously between molecules even without hybridization of tags. As discussed earlier, the localization provided by tag binding can be exploited to reduce this to a low level, on the order of 1 in 10,000 complexes in the prototype.

Computation: Kinase action causing phosphorylation inconsistent with the logic of a NAND gate. With an active site split between the components of the active dimer, accidental enzymic phosphorylation can be reduced to near the spontaneous level observed in the absence of enzyme.

Computation: Failure to phosphorylate when that reaction is the correct logical outcome. The primary cause would be dissociation of one or both of the input molecule tags before the enzymatic reaction takes place. There are then two possible situations. In the first, detachment could occur before full processing of the output tag mask by the exonuclease. In this case, an active complex will eventually reform, and the reaction will again have an opportunity to take place. In the other situation, detachment of an input molecule may take place after full mask processing, but before phosphorylation. In that situation, an error would be propagated. The chances of this occurring can in principle be reduced by decreasing the catalytic rate of the exonuclease.

Synchronization: Spontaneous dissociation of the tag-masking oligomer not aided by the action of the exonuclease. Because the concentration of tag masks in solution is close to zero, either a very strong interaction and/or a very long half life between the mask and its complementary tag is needed. This is important because detached tags may associate with output tags on equivalent elements where processing is complete. Spontaneous dissociation can be reduced by making the mask complementarity longer. However, this is probably not necessary because atomic force microscopy data[32] suggest that half-lives of DNA complexes are sufficiently long to minimize tag transfer.

Robustness to errors

It is clear that some errors are unavoidable in such a system, and thus a certain proportion of molecules will carry an incorrect value. On the other hand, in this model, robustness may be achieved by utilizing the fact that the same computation is performed by a large number of molecules. The redundancy of molecules carrying the result provides a mechanism for eliminating errors. For a network with N elements, and an error rate per gate of e, the probability of a correct computation is $(1-e)^N$. If the value of e is sufficiently small, a majority vote can be used to obtain a correct result. For example, a system with 100 gates and an error rate of 0.001 per gate would produce highly reliable majority vote results. For cases where the size of e is incompatible with the system size, it is not sufficient to take a majority vote on the final results, and the end result of the computation is dependent on obtaining the correct result at each stage. A correction mechanism can be based on the observation that the phosphorylation state of active molecules representing the same gate (i.e., copies of the same gate) should all carry the same value. Different values signify that one of the molecules carries an incorrect result. Because there is no way to know which one carries the correct result, the simple solution would be to eliminate both. The key here is to contain the error in such a way that it will not propagate further along the computation. Such a comparison might be implemented, for example, by a methylation reaction that would be triggered when two active, similarly tagged, molecules with different phosphorylation states interact. Methylation would then block participation of molecules in further interactions.

Automation and Production

The prototype system has a single protein molecular species, containing a kinase and a nuclease domain. A fully developed system would have additional protein components for error control and system resetting. These proteins can be produced using conventional protein expression and purification procedures, and used in the construction of all circuits. The DNA tags are circuit specific, and provide wiring between gates. A circuit would first be designed using standard gate notation. Two complementary tag sequences would be chosen for each wire connecting the output of one gate to the input of another. The sequences are random, with constraints on composition to ensure appropriate binding constants. Once all tag sequences for a circuit have been generated, an iterative procedure is run, checking to see that no two tags are too similar in sequence, and if they are, generating a new sequence for one of them. Such a library of tags can be prepared in advance and used for all circuits.

The conversion of an electronic circuit to a set of tag oligomer sequences can be fully automatic. The three tag sequences for each gate are combined into one string, with suitable linker regions between them and at the ends. These sequences will be approximately 80 nucleotides long in total, and so can be produced by the same high throughput procedures used in cloing and PCR. Stochometric amounts of protein and oligomers are then mixed, under conditions that lead to linkage between DNA and protein. In the system we outlined there are $N \times 10^7$ protein molecules, and 10^7 copies of each tag oligomer (where N is the number of gates in the circuit). It is not necessary to have exact numbers. Excess oligomers or protein are not expected to interfere with the function of the circuit.

COMPUTER SIMULATION

We have performed a computer simulation to ensure that the basic design is logically consistent, and evaluated its performance and robustness to errors.

Two types of circuit were simulated. One is a generic type whose architecture is of a full binary tree, that is, a layered structure where each gate is connected to two gates in the previous layer. A system of N levels will thus have $2N - 1$ gates. This allows for simple scalability of the system and simple measurements of the effect of varying parameters. All the input gates were set to the same logical value. (The logic of such a network of NAND gates

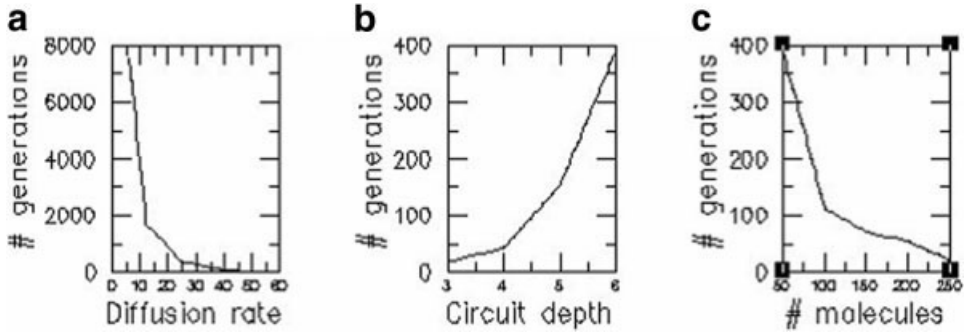

Fig. 5. The performance of the simulation with the binary tree circuit. Running time is measured as the number of generations (a generation consists of moving every molecule once) until 50% of the output molecules are activated. (**a**) Running time decreases exponentially in the diffusion rate, that is, in the distance (in grid units) each element can move in one step. (**b**) Running time increases exponentially in the depth of the circuit, but (**c**) decreases exponentially in the number of the copies for each molecule.

makes the result of the calculation alternate between levels, that is, if a system of N levels results in 1, then a system of $N + 1$ levels will results in 0.) The other circuit that was tested was the Majority function described earlier. Simulations were done on a two dimensional grid of 600×600 cells in which molecules where allowed to diffuse between cells. Total computation time is defined as the number of steps required for 50% of all the output gates to be activated. Various conditions were investigated.

Performance

First, we tested the effect of diffusion rates in terms of the maximal step size (in grid units) a molecule can take in a single step. Then, we tested the effect of changing the circuit size, and finally we tested the effect of changing the number of copies of each gate that participates in the system. The results for the binary tree are shown in Figure 5.

In the first experiment with a binary tree network, the number of copies of each gate was set to 100, and we used a network with 5 levels (i.e., 31 gates). The diffusion rate (the size of the diffusion step in lattice units) was varied from 6 to 60. (i.e., for a diffusion rate $\boldsymbol{d}$, Δx, and Δy were changed by a randomly chosen value between 0 and $\boldsymbol{d}$). As expected, the computation time in terms of the number of generations (each generation is one move of each molecule) decreased significantly as the diffusion rate increased. In the next experiment, the diffusion rate was set to 36 lattice points per move, and the size of the circuit was changed. Networks of depth 3 (i.e., binary trees with three layers, containing seven gates) to depth 6 (63 gates) were investigated. The computation time increased exponentially with the depth of the circuit, whereas in a silicon-based computation, the time increases approximately linearly with the circuit depth.

Next, we tested the time dependence on the level of parallelism in the system in terms of the number of copies of each gate. At a diffusion rate of 36 lattice points per move it can be seen [Fig. 5(c)] that the performance improves exponentially with the number of copies. This property of the protein-based system offsets the exponential time dependence on gate depth. A simple extrapolation suggests that with the intended number of copies (10^7) circuits of depths of up to 20 could be handled. However, notice that our requirement that 50% of the output elements will complete their computation before the result is over determined. In practice, with 10^7 copies of the output elements, even when 1% of the copies (i.e., 10^5) are completed, the result can be reliably determined. This will enable circuits with significantly greater depths.

TABLE I. Relationship between Gate Error Rate and the Fraction of Output Gates Providing the Correct Logical Result

		Error Detection & Elimination	
% Error rate	No Error Detection, % Correct	% Correct	% Yield
1	100	100	83
2	92	98	75
5	74	97	48
10	61	92	24
20	55	100	5

Robustness to Errors

Next, we tested the performance of the network when errors were introduced. We simulated errors, using a single parameter that specifies the probability that the result of a gate operation is not the correct NAND outcome. The results, for the majority function, with 100 copies and a diffusion rate of 36 (Table I) show the relationship between error rate and circuit accuracy.

It can be seen that up to error rate of 10%, the output accuracy is still reasonable and the correct answer can be obtained by taking the majority result over the set of output gates. If the overall error rate in each elementary calculation is higher, the percentage of the correct answer gets too close to 50% to allow reliable determination of the outcome. Thus, it is necessary to employ an active mechanism of error detection and elimination. We simulated a method for error elimination in which every active molecule undergoes a validation check, by comparing its output value with another active copy of the same molecule, as discussed above. Such a mechanism produces a dramatic

improvement in the robustness of the results (Table I). The success rate of the computation is above 90% up to the highest error rate tested, 20%. This success is achieved at the cost of eliminating some molecules and greater system complexity. With the low number of copies used here, only 24% of molecules in the output layer remain when correcting a 5% error rate, and only 5% remain with an error rate of 20% (in the simulation, all of the remaining 5% of copies yielded the correct answer, thus bringing the computation back to perfect 100% accuracy, but this is probably an artifact of the small final yield).

As discussed in the Robustness section, no single error appears to significantly affect the final outcome (i.e., error rates can be controlled to a very low level), and for small circuits, at least, error correction should not be necessary. However, error estimates are far from quantitative, so it is reassuring that high error tolerance should be possible.

DISCUSSION

We now return to the questions raised in the introduction, concerning the possibility of computing with biological components, and show how our model addresses each of them.

1. How can logical gates (i.e., switching) be implemented by a protein-based system? Logical operations are performed by phosphorylation reactions that implement the logic of a NAND gate.
2. How can wiring between gates be implemented? Wiring is implemented by single-strand DNA tags that are attached to each protein. A pair of complementary tags wire the output of one gate to the input of another.
3. What are the tokens of the computation, that is, how is the computation carried out from input to output? The tokens of the computation are defusing molecules with two different possible phosphorylation states. The phosphorylation state of these molecules carries the information transferred from the input to the output of the circuit.
4. Because the biological processes utilized in the system vary in their reaction speed, how can the timing of the computation be synchronized? Synchronization of the network is achieved by blocking the output tags of each molecule until that molecule has become associated with the appropriate input molecules. Unblocking of tags is performed by an exonuclease which is activated upon complex formation.
5. What are the expected errors in the process and how can these errors be contained? Problems that might occur have been identified and discussed. Conditions were identified to minimize the possibility of error in each process. Furthermore, the redundancy in the system (i.e., having 10^7 copies of the circuit) enables reliable computation of the entire system even when the individual reactions might be erroneous. If the error at each stage were to become too large to be contained, then more active error detection mechanisms could be added. Possible approaches to this problem are described.
6. How can the design of such a computation device be automated such that it will possible to take a layout of a regular electronic circuit and produce a biological equivalent? The process of converting a logical circuit to biological computation can be automated because the design uses a single protein species to build all the logical gates. Wiring between gates is created by synthesizing appropriate DNA tags and attaching them to the protein molecules, using standard technology.

Two major engineering challenges lay in the heart of the system. One is attaching DNA (or PNA) tags to proteins and their use to facilitate protein–protein interactions. The second is engineering a dimeric negatively controlled autokinase. While all of the protein engineering required for the system are possible in principle, it obviously requires, especially the second task, a great deal of development in practice. Note however that once a working system has been constructed, the same component design can be used for any logical circuit, and thus any technological improvement in the elementary processes will directly benefit every computation.

Clearly, these are just the first steps towards computation with biological components. We believe there is a large potential for such technology, especially in medical applications. We hope that the model presented here will stimulate further research in this field. Specifically we hope that our ideas will encourage experimentalists to investigate practical implementations.

ACKNOWLEDGMENT

We thank Zvi Kelman for very helpful ideas and discussions regarding the possible role of an exonuclease in synchronizing the computational process, and Adi Kimchi for introducing us to the family of DAP-kinases. A patent application is pending.

REFERENCES

1. Adelman LM. Molecular computation of solutions to combinatorial problems. Science 266:1021–1024.
2. Yurke B, Turberfield AJ, Mills AP Jr, Simmel FC, Neumann JL. A DNA-fuelled molecular machine made of DNA. Nature 2000;406: 605–608.
3. Braich RS, Chelyapov N, Johnson C, Rothemund PW, Adleman L. Solution of a 20-variable 3-SAT problem on a DNA computer. Science 2002;296:499–502.
4. Benenson Y, Paz-Elizur T, Adar R, Keinan E, Livneh Z, Shapiro E. Programmable and autonomous computing machine made of biomolecules. Nature 2001;414:430–434.
5. Benenson Y, Adar R, Paz-Elizur T, Livneh Z, Shapiro E. DNA molecule provides a computing machine with both data and fuel. Proc Natl Acad Sci U S A 2003;100:2191–2196.
6. Benenson Y, Gil B, Ben-Dor U, Adar R, Shapiro E. An autonomous molecular computer for logical control of gene expression. Nature 2004;429:423–439.
7. Bode BW, Sabbah HT, Gross TM, Fredrickson LP, Davidson PC. Diabetes management in the new millennium using insulin pump therapy. Diabetes Metab Res Rev Suppl 1 2002:S14–S20.
8. Turing AM. On computable numbers, with an application to the Entcheidungproblem. Proc Lond Math Soc II Ser 1936;42:230–265.
9. Braun E, Eichen Y, Sivan U, Ben-Yoseph G. DNA-templated assembly and electrode attachment of a conducting silver wire. Nature 1998;391:775–778.
10. Keren K, Krueger M, Gilad R, Ben-Yoseph G, Sivan U, Braun E.

Sequence-specific molecular lithography on single DNA molecules. Science 2002;297:72–75.
11. Morton-Firth CJ, Shimizu TS, Bray D. A free-energy-based stochastic simulation of the Tar receptor complex. J Mol Biol 1999;286: 1059–1074.
12. Elowitz MB, Leibler S. A synthetic oscillatory network of transcriptional regulators. Nature 2000;403:335–338.
13. Gardner TS, Cantor CR, Collins JJ. Construction of a genetic toggle switch in *Escherichia coli*. Nature 2002;403:339–342.
14. Chen Z, Govender D, Gross R, Birge R. Advances in protein-based three-dimensional optical memories. Biosystems 1995;35:145–151.
15. Ashkenazi G, Ripoll DR, Lotan N, Scheraga HA. A molecular switch for biochemical logic gates: conformational studies. Biosens Bioelectron 1997;12:85–95.
16. Sivan S, Lotan N. A biochemical logic gate using an enzyme and its inhibitor. 1. The inhibitor as switching element. Biotechnol Prog 1999;15:964–970.
17. Bray D. Protein molecules as computational elements in living cells. Nature 1995;376:307–312.
18. Corey DR, Schultz PG. Generation of a hybrid sequence-specific single-stranded deoxyribonuclease. Science 1987;238:1401–1403.
19. Bruick RK, Dawson PE, Kent SB, Usman N, Joyce GF. Template-directed ligation of peptides to oligonucleotides. Chem Biol 1996;3: 49–56.
20. SantaLucia J Jr. A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. Proc Natl Acad Sci U S A 1998;95:1460–1465.
21. Corey DR Peptide nucleic acids: expanding the scope of nucleic acid recognition. Trends Biotechnol 1997;15:224–229.
22. Zhang X, Ishihara T, Corey DR. Strand invasion by mixed base PNAs and a PNA-peptide chimera. Nucleic Acids Res 2000;28: 3332–3338.
23. New England Biolabs 2002–2003 catalog, p 107–108.
24. Holland PM, Abramson RD, Watson R, Gelfand DH. Detection of specific polymerase chain reaction product by utilizing the 5′–3′ exonuclease activity of *Thermus aquaticus* DNA polymerase. Proc Natl Acad Sci U S A 1991;88:7276–7280.
25. Shani G, Henis-Korenblit S, Jona G, Gileadi O, Eisenstein M, Ziv T, Admon A, Kimchi A. Autophosphorylation restrains the apoptotic activity of DRP-1 kinase by controlling dimerization and calmodulin binding. EMBO J 2001;20:1099–1113.
26. Shohat G, Shani G, Eisenstein M, Kimchi A. The DAP-kinase family of proteins: study of a novel group of calcium-regulated death-promoting kinases. Biochim Biophys Acta 2002;1600:45–50.
27. Jencks WP. From chemistry to biochemistry to catalysis to movement. Annu Rev Biochem 1997;66:1–18.
28. Golemis EA, Serebriiskii I, Law SF. The yeast two-hybrid system: criteria for detecting physiologically significant protein–protein interactions. Curr Issues Mol Biol 1999;1:31–45.
29. Rose ZB, Dube S. Rates of phosphorylation and dephosphorylation of phosphoglycerate mutase and bisphosphoglycerate synthase. J Biol Chem 1976;251:4817–4822.
30. Promega corporation, 1998. Erase-a-base technical manual. Available from: http://www.promega.com/tbs/tm006/tm006.pdf
31. Van Lint JH. Introduction to coding theory, 3rd ed., New York: Springer-Verlag; 1999.
32. Pope LH, Davies MC, Laughton CA, Roberts CJ, Tendler SJ, Williams PM. Force-induced melting of a short DNA double helix. Eur Biophys J 2001;30:53–62.