Available online at www.sciencedirect.com
ScienceDirect
Physica D 237 (2008) 1157–1164
PHYSICA D
www.elsevier.com/locate/physd

The neglected pillar of material computation

Susan Stepney

Department of Computer Science, University of York, YO10 5DD, UK

Available online 9 February 2008

Abstract

Many novel forms of computational material have been suggested, from using slime moulds to solve graph searching problems, to using packaging foam to solve differential equations. I argue that attempting to force such novel approaches into the conventional Universal Turing computational framework will provide neither insights into theoretical questions of computation, nor more powerful computational machines. Instead, we should be investigating matter from the perspective of its natural computational capabilities. I also argue that we should investigate nonbiological substrates, since these are less complex in that they have not been tuned by evolution to have their particular properties. Only then we will understand both aspects of computation (logical and physical) required to understand the computation occurring in biological systems.

Keywords: Material computation

1. Introduction

Today's computing, *classical* computing, is an extraordinary success story. However, there is a growing appreciation that it encompasses an extremely small subset of all computational possibilities. A variety of paradigms encompass classical computing, and their assumptions need to be carefully scrutinized. The UKCRC's Grand Challenge exercise [1] includes the Grand Challenge of Non-Classical Computation (GC-7) [1–4], whose task it is to challenge and move beyond the various classical computational paradigms, thereby broadening and enriching the subject area.

GC-7 identifies and challenges the classical paradigms [3], of which the Turing paradigm is arguably the most prominent one. The Turing machine model assumes that computation is a logical or mathematical property, and that the computational substrate is merely an implementation detail. As Deutsch [5] neatly sums it up:

"Turing hoped that his abstracted-paper-tape model was so simple, so transparent and well defined, that it would not depend on any assumptions about physics that could conceivably be falsified, and therefore that it could become the basis of an abstract theory of computation that was independent of the underlying physics."

Deutsch goes on to argue that Turing's model is in fact falsified with respect to quantum physics. Other assumptions of the Turing model can be questioned [6] and modified, in particular considering the effect of interacting with a physical real-time environment [7].

The Turing model, relevant to computers as they are conventionally engineered, is a *designed* logico-mathematical computational model. People also see computation occurring naturally. Neural networks, immune systems, evolving populations, ecosystems, termites building massive and complex homeostatic mounds, ant colonies finding shortest paths to food, bacteria swimming up concentration gradients towards nutrients, and more: all these biological systems are said to compute.

But what does it mean, from this classical Turing perspective, to say that biological systems compute? Turing computation, *designed* computation, is about halting, computability and universality; it is symbolic, discrete and closed (pre-defined); it is deterministic and sequential (in the sense that probabilistic or parallel variants provide no additional computational power); it finds many interesting problems infeasible to compute in general; its calculations are exceedingly fragile to small changes or errors. On the other hand, biological computation, *found* computation, is about not halting (halting equates to system death); it is (mostly) nonsymbolic, continuous and open (constantly adapting and evolving due to the continual flow of

E-mail address: susan@cs.york.ac.uk.

0167-2789/$ - see front matter
doi:10.1016/j.physd.2008.01.028

matter, energy and information through the system); it is essentially stochastic and massively parallel; it finds feasible near-optimal solutions to many classes of classically infeasible problems; and it is robust to many classes of errors. (Indeed, it is these properties of near-optimal solutions and robustness [8] that attract many to the domain of bio-inspired computing.) So, how can biological systems, with properties so different from classical Turing computation, be considered to be computing? And if these systems are computing, what is giving them these properties not seen in classical computational systems?

These questions arise because half the picture is missing. Biological systems are different from the logico-mathematical Turing view of computation: they directly exploit their material substrate to perform their own style of computation. This contrasts with the case of classical computation, which is deliberately abstracted away from the implementation substrate precisely to make it substrate-independent. The classical computational virtual machine can be implemented on any suitable substrate, no matter how 'unnatural' that implementation may be for the substrate (for example, analogue transistors being run saturated to make them act as digital switches).

In this paper, I argue that we need to consider the physical, material aspect of computation before we can understand biological computation. To do this, we should not start with biological substrates, since these have been finely tuned by evolution to have their particular properties. Instead, we should investigate various nonbiological substrates, and examine their essential computational capabilities.

2. The effect of the substrate

2.1. The constraints of the substrate

The Turing model abstracts away from the details of the physical substrate. Nevertheless, a realization of a computation must occur in some physical device. Any physical structure labours under the *physical constraints* of the material from which it is constituted. These constraints include the general constraints affecting all materials arising from the basic laws of physics, which involves consideration of: the speed of light; conservation of energy; entropy; energy and mass density; quantum limits; the size of the observable universe. These provide ultimate physical limits to computation [9–13], general limits that can be considered the physical analogue of computability constraints in classical computation. (There are also some philosophical arguments whereby some logico-mathematical computability constraints maybe also appear as physical constraints in the form of fundamental physical laws.)

¿From the point of view of a biological system, or any other open system interacting with its environment, there are constraints due to natural length scales and timescales governing the interaction dynamics.

More interestingly, from the point of view of this paper at least, there are also specific physical properties local to the substrate, such as its strength, plasticity, elasticity, electrical and thermal resistance, and so on. These provide problem and substrate limits to computation, specific limits that can be considered the physical analogue of feasibility constraints in classical computation.

2.2. The power of the substrate

Constraints are not always a disadvantage, however. The state space is constrained to a smaller region, and the computation to particular trajectories. If those regions and trajectories are desired ones, the constraints can help the performance of computation. For example, elasticity may enable some form of return to equilibrium, or 'reset', to occur without further specific implementation [14].

Additionally, the physical properties are not merely constraints. Certain physical properties may enable certain computational aspects to happen "for free", compared to implementing them in a classical computer: the computational system may be able to transfer some of its computational burden (be it memory or processing) to the substrate. Exploitation of the specific substrate may allow new problems to be solved in new ways; physically rich substrates may exhibit vast, if specific, computational power. Indeed, this is one of the claimed advantages of the whole field of analogue computation: the underlying physical substrate implements the required computation *directly*, rather than via levels of 'virtual machine' that abstract away from the substrate and implement the logical computational model. When there is a good match between the physical properties and the desired computation, then very efficient results may be obtained. For example: the spaghetti sorting computer [15]; constructing Steiner minimal trees [16] using the surface tension properties of a soap film as a surface area minimiser [17]; performing 2D Fourier transforms directly using the optical properties of lenses. (It should be noted that one of the drawbacks to these analogue methods often glossed over is that they do not scale: for example, there is often a loss of precision with large problems. However, analogue computers have been very widely used: the slide rule was ubiquitous for over a century.)

2.3. Simulating the substrate

Exploiting the power of the substrate should be contrasted with the subdiscipline of "nature-inspired computing", where a physical, chemical or biological process is interpreted as computation, abstracted from the substrate, and implemented by analogy in a classical manner. For example, simulated annealing [18], various artificial chemistries [19], generic algorithms [20], ant colony optimization [21], to name but a few, have all been abstracted from the underlying substrate and implemented in classical terms. These have proved highly successful in their new abstract domain, but have lost something in the translation. For example, there are often discussions on how to *implement the constraints*, such as conservation of mass in artificial chemistries, or decay of pheromones in ant systems. When we are working with the substrate, the substrate does such implementation "for free". (Of course, one of the advantages of abstracted algorithms is that one can

then implement *different* physical laws, unphysical laws, if so desired.)

3. Example substrates

3.1. Biological substrates

There is active research in the use of biological substrates to perform "wet" unconventional computation. Possibly the best recognized, and best developed, is DNA computing [22], exploiting the molecular properties of base-pair complementarity, originally to build a Hamiltonian path finding device. Another relatively well-developed application of a bio-molecule is the use of the bacteriorhodopsin protein as a three dimensional optical storage memory [23,24]. There is also active research in exploiting whole biological cells as components in computational devices, from leech neurons [25] to bacteria and slime moulds [26].

This research is interesting and productive, but does it tell us anything deeper about computation? I would submit not. There are two main reasons for this. Firstly, the applications chosen are usually classical and digital, and not naturally suited to the analogue substrates. Secondly, and more profoundly, the biological substrate is extremely complex and complicated, having evolved over billions of years to exploit specific properties. In some sense, biological substrate is as far (or further!) removed from a primitive substrate as are our own designed abstract digital computational media. This makes it extremely difficult to develop any abstract models of biological material computation, or any concepts of how to exploit (program) such material.

Hence, in order to understand, develop and exploit computation *in materio* [27], we need to move to simple (that is, unevolved) materials: move out of the domain of biology, and into that of chemistry and physics.

3.2. Known physical substrates

Turing introduced reaction-diffusion systems as an hypothesized morphogenic mechanism underlying animal coat patterns [28]. Reaction-diffusion systems comprise chemicals that react with each other locally, and diffuse spatially at different rates through the system. This complicated nonlinear process can lead to waves and spots of activity throughout the substrate. This activity can be modulated by applying spots of chemicals, or illuminating the substrate with spatially varying patterns of light. Reaction-diffusion systems are now considered more generally to be computational systems [29]. For example, a 2D chemical substrate can be prepared to solve a 2D Voronoi diagram problem (given a set of p_n points in space, divide the space into n regions, one per point, such that every point in a region i is closer to p_i than it is to any other p_j). Put a spot of chemical at each point p_i; the chemical diffuses out; at the boundaries of the Voronoi regions diffusing waves meet and react, leaving a chemical trace marking the boundaries. RD systems can also be used to implement logic gates (see for example [30]) and hence Turing machines.

Mills [31] implements (approximations to) Rubel's extended analogue computer, using a variety of materials as the computational substrate: conductive surfaces and solids including conductive plastic foam, and gelatin doped with sodium chloride (that is, salted jelly[1]).

3.3. Novel physical substrates

These are known substrates being exploited. Where should we look to novel substrates that might tell us something about computation? The idea that the 'edge of chaos' [32] is connected with maximal complexity and computational power is at least suggestive (despite its details having been questioned [33]). This idea is that maximal computational power occurs near a phase transition (in some parameter of the system) [34], from a 'solid' form (where the system has structure/memory, but no dynamics/processing) to a 'fluid' form (where it has plenty of dynamics, but no structure). In other words, to get interesting computational properties, we want to look for phases of matter with both complex dynamics and complex structure over a wide range of time scales and length scales. It is also helpful to look at cases where we could exploit existing commercial laboratory technology [35].

Consider liquid crystals, a form of matter that lies on the boundary between solids and fluids (sometimes called 'the fourth phase of matter'). A liquid crystal has both dynamics (the molecules can flow and rotate) and structure (the molecules are ordered on length scales much bigger than their individual sizes). Can such materials perform computation? The answer is a definite yes. Harding and Miller have demonstrated that a liquid crystal chip can be programmed to act as a tone discriminator and as a robot controller [36–38]. It is currently unclear precisely how the material performs these computations: in these cases the matter was programmed using an evolutionary algorithm. One reason these researchers chose to experiment with liquid crystals in the first place was the commercial availability of devices conveniently packaged with electrical contacts: liquid crystal displays. Liquid crystals are just one form of "soft (condensed) matter". The whole field appears to be ripe for computational exploration, all the more so because one end of the spectrum of this complex form of matter includes bio-materials.

Consider nuclear spins, which are manipulated by magnetic and radio frequency fields in the discipline of nuclear magnetic resonance (NMR) and magnetic resonance imaging (MRI). Materials manipulated in this was have complex structure and dynamics in terms of the interacting spin states, which may be analysed in computational terms [39], and has the advantage of commercially available spectrometers sitting in many university Chemistry departments. (This is different from the existing field of NMR quantum computing, which exploits spin entanglement in single molecules, rather than complex properties of bulk matter.)

Consider plasmas (ionized gases, *also* sometimes called 'the fourth phase of matter'), especially as they occur in

[1] Or 'salted Jello', depending on your side of the Atlantic.

experimental fusion reactor plasmas. This matter certainly has dynamics (much of the research is dedicated to controlling this dynamics), and turbulent structure over many length scales. The performance of a fusion reactor depends on the pressure gradient that can be sustained in the plasma, which is determined by the rate at which energy leaks out of the plasma due to turbulence. These fine-scale turbulent processes can generate a large scale sheared flow, which feeds back to suppress the turbulence, influencing the pressure gradient in a complicated way. This non-linear system can yield experimentally observable phenomena such as bifurcations, depending on the choice of parameters. For example, we might model the pressure gradient as a pile of sand, with the plasma heating at its core analogous to adding grains of sand to the top of the pile. Instabilities in the plasma can be triggered when the pressure gradient exceeds a certain value. The resulting redistribution of pressure destabilizes another region, and then another, leading to an avalanche process affecting the system over a much larger length scale than the initial perturbation at the unstable region. So the fusion plasma has structure and dynamics, and can be analysed as a complex system. Can it be analysed as a computational system? What are the natural computational modes for such a system?

4. From substrate to computer

4.1. Programmability

There are many forms of matter that have complex structure and dynamics. Each could be analysed as a computational system, relating its computational properties and power to physical properties of the material substrate. What is also required is a more unified theory of various computational models of matter with structure and dynamics. This will tell us something fundamental about the very concept of computation, and how we could exploit the different natural computational properties of different kinds of material substrates.

Computation implies more than simply allowing matter to slosh around in a complex manner, however. It requires programmability: we want to make the matter compute for us, to solve our problems, not (just) its own.

Any matter used to implement a computer needs to be programmable on at least two levels: first to implement the model of computation, second to implement a particular program (including input and output).

4.2. Resist the lure of Universality

A curious compulsion overcomes many researchers in unconventional computation: no matter how inappropriate their substrate, they seem compelled to use it to implement logic gates, and thereby demonstrate that it can perform Turing universal computation.

Whilst it might be intellectually interesting to realize just how many weird ways there are to implement extremely inefficient Turing machines, what does this teach us about computation that is new? I would submit that, while using a reaction-diffusion computer to generate Voronoi diagrams in a natural manner suited to the medium is interesting, using it to implement unnatural and glacially slow logic gates is rather less so.

This compulsion seems to stem from a fear that conventional digital colleagues will disparage nonuniversal devices; however, a long history of successful nonuniversal analogue computation would not seem to support this fear. We should be exploiting novel materials for what they can naturally do well, not restricting them to do what other materials can do so much better.

Zauner and Conrad [40] argue strongly for consideration of the advantages of "anti-universal" machines, that can solve only a certain class of problems. The argument partly reduces to how much of the "program" resides in the logical state (software), and how much in the physical state (hardware), of the machine. When is physical reconfiguration to be considered as building a different computer, and when is it merely reprogrammming the hardware? The boundary here is blurring even in the case of classical digital hardware, with the growing used of "programmable hardware" such as Field Programmable Gate Arrays (FPGAs), and will blur even more as we use other complex matter as a computational substrate.

Zauner and Conrad [40] further argue that it may even be advantageous to consider one-shot "instance machines", that can solve only a single instance of a problem. By avoiding the requirement for resetting a machine to its initial configuration, the computation can irreversibly alter the state of the system (often a consequence of using a complex biological substrate, for example).

As we extend our notion of what is a computational device away from the requirement for universality, we will need to consider more dimensions in complexity and cost analysis [41], especially the setup time (initializing the logical and physical state), and the material cost (which may no longer be able to be amortized over multiple uses).

4.3. Implementing the computational model

Abstract mathematical models of computation need to be implemented in physical devices that have been engineered to behave in a manner isomorphic to those models. That engineering can be extremely intricate, since there is no reason to believe *a priori* that physical material will behave in accordance to some independent, unrelated abstract model.

For example, Charles Babbage implemented abstract mathematical laws of arithmetic using exquisitely designed and arranged collections of tens of thousands of brass gears and other parts. (Although Babbage himself never actually completed his Difference Engine, Swade and colleagues at the Science Museum built a working implementation, completed in 1991, to Babbage's original design and within the engineering tolerances of the time [42].)

Similarly, implementation of a Turing machine requires implementation of the "moving parts" that represent the current state, the state machine transitions, the moving head, the tape, and the tape symbol reading, writing and erasing mechanisms. Although one *could* do this (micro-)mechanically [43,44], the usual implementation is the classical digital electronic

computer. This requires exquisitely designed and arranged collections of many millions of transistors and other parts.

The *raison d'être* of computation *in materio*, however, is to be able to forego this implementation step. Rather than start with some abstract mathematical model, and then delicately engineer the substrate to implement it, one starts with the computation that the substrate does naturally. So there is little or (ideally) no engineering required to implement the model of computation itself. One is instead computing "close to the physics", doing what comes naturally, and therefore (hopefully) efficiently. This does not mean, however, that there is no engineering required at all: one still has to be able to program the device, else it is just a block of material.

4.4. Ballistic programming

The Turing machine model is a "ballistic" style of computation: that is, the program is loaded, the initial configuration (input) is set up, then the computation "fires off" and proceeds with no further input from the external world. If and when the machine eventually halts, the final state (output) can be read from the tape. The Turing model implements a black box (partial) function evaluator.

An *in materio* ballistic analogue would require its initial condition of the material to be set in a particular input state, then the material would be left to compute, and the final state of the material would contain (an encoding of) the output. Different initial conditions, corresponding to different inputs, would yield correspondingly different outputs. But what of the "program"?

At the simplest level, the "program" is just the laws of physics, as realized in matter with complex structure and dynamics. The substrate follows these natural laws, in a complex manner from its given initial conditions. To change the program, one cannot change the laws of physics, however, so one changes the structure and dynamics of the substrate.

One could do this by changing the substrate itself for one with different structure and dynamics; essentially, changing to a different computer.

One could design the path of the computation itself to affect the material, and hence its future behaviour, through feedback, including: temperature and other physical changes; chemical composition changes; biological growth processes. Depending on how resettable such changes were, this form of programmability might imply a one-shot "instance machine". (One of the current problems with using biological substrates is that they are extremely more difficult to "program" in this manner than nonbiological substrates, due to them having evolved intricate structure and dynamics for their own purposes.)

Or one could directly modulate the substrate's behaviour by application of some form of external field (for example, electrical potentials [36], magnetic fields [39], incident light). Since one would probably wish to alter the modulation depending on the current state of the computation, this final option fits in with an interactive style of programming, where the modulation might also be chosen depending on the current state of the environment.

4.5. Interactive programming

Computational models such as CSP [45] and π-calculus [46] are less concerned with halting, and more concerned with ongoing interaction with an environment. Wegner [47] discusses interaction machines, "Turing machines extended by addition of input and output actions that support dynamic interaction with an external environment" (and claims that they are more powerful than Turing machines). Interaction machines are therefore more akin to a "guided missile" than a "ballistic" metaphor. They provide a more natural way of describing and reasoning about certain forms of computation: real-time, embedded, embodied, interactive applications that need to respond to their changing environment, which itself may be changing in response to intermediate outputs from the computation, in some closely coupled feedback loop [7]. This style of computation has rather different requirements from classical discrete "payroll processing" style applications (which are well suited to classical computation).

For example, the computation should match the timescales and precision of the environment with which it is interacting, and be tolerant of stochastic noisy analogue input. A material system that operates on the same timescales and precision as the environment, and which is itself also stochastic, noisy and analogue, may well be more naturally suited than a high-precision digital system. The interactive style of computation furthermore allows continual minor corrections "in flight", rather than requiring exquisitely precise initial setup (which is furthermore impossible even in principle in an unpredictably changing environment). One would wish to work with the natural dynamics of the substrate, so that minor perturbations due to noise naturally return to the original computational trajectory, and that environmental (including programmatic) inputs naturally move the system into other trajectories. In other words, we are working with, rather than against, the natural dynamics of the substrate. We can even exploit the noise as a computational resource in its own right [48].

The means of programming an interactive *in materio* machine would naturally be by modulating externally imposed fields, which could well be integrated with environmental input mechanisms (via suitable transducers).

4.6. The programming model

A natural question arises: how to chose the right field modulations to achieve the desired computation? In other words: what is the programming model? Harding et al. [38] argue that material substrates are so complicated and their dynamics so ill-understood that the best approach is to treat each substrate as a "black box", and use an genetic algorithm approach to evolve the desired programmed inputs. They use this technique successfully to program their liquid crystal applications [36,37].

There are two objections to their proposal, however. First, they use fixed inputs for a computational run, and it is not clear that their approach scales to more sophisticated time-varying inputs. Second, and more importantly, although such an

approach suits their application-orientated emphasis, it would not yield any new insights into the nature of computation itself. Miller [private communication] has suggested that an evolutionary approach could be adapted to investigate the computational properties, in a more white-box approach: this seems entirely plausible.

As for the argument that the dynamics is too complicated: although the detailed microscopic properties probably are too complicated to model or simulate to any fine degree, it might be hoped that a *computational* abstraction would be more amenable to analysis. In the same way that it is possible to abstract the detailed semiconductor physics of a transistor into a model of a switch (albeit because the implementation was designed to support that very abstraction), it might be possible to abstract the detailed dynamics of the substrate into higher level trajectories through phase space and attractor basins, and thence to form the basis of a computational, and ultimately, a programming, model. For more classical computational examples: DNA computing exploits only a small subset of the attributes of the system (specific pairing of the complementary bases), so the detailed chemical processes do not need to be considered from the perspective of the computational model; a computational model of a biological regulatory network may consider it to be a logical switching network, abstracting from its underlying biological and chemical complexity.

4.7. Gradients and flux

A computer, like any nonequilibrium device, requires a flow of matter, energy, or information through it to function. Classical digital computers ingest electricity and excrete waste heat.

Material computers also require such a gradient and flow: slime moulds require nutrients; chemical reaction diffusion systems require influx of fresh chemicals and removal of waste products; packaged liquid crystal displays run on electrical power; and so on. Applying and controlling a suitable gradient (for example, extracting specific waste chemical products, or controlling environmental fluctuations such as temperature changes) might be quite challenging.

Biological organisms have evolved to exploit information gradients. A full computational theory of biological systems should be able to explain and exploit the roles of both material and information gradients.

4.8. An architecture

Fig. 1 sketches an architecture for a programmable *in materio* computer. The main component is the material substrate, with complex structure and dynamics behaving according to the laws of physics. Through this substrate run pervasive programmable fields, to modulate the substrate's structure and dynamics. (The precise nature of these fields will be determined by the requirement for programmability, and by the specific details of the material: electromagnetic fields are one obvious candidate.) Environmental inputs, and program instructions, are provided via these fields. Information

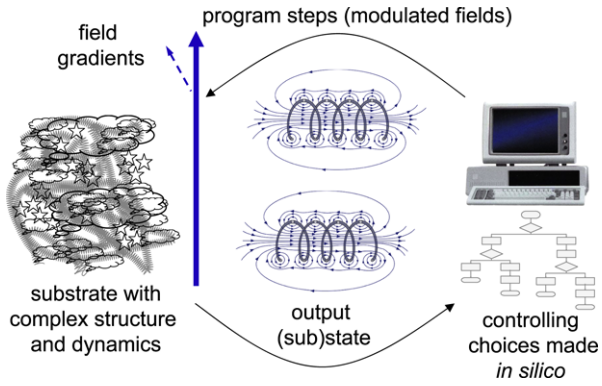

Fig. 1. An architecture for a programmable *in materio* computer.

from the substrate can be read out, by natural emissions, or by modulation of pervasive fields. A programmable device (possibly a classical digital computer) controls the pervasive fields, thereby providing program instructions. The precise instructions provided may depend on details of the output from the substrate. Hence the structure and dynamics of the substrate are modulated by environmental inputs and program instructions, which in turn may be modulated by the material's state, in a closely coupled feedback loop.

Simpler special purpose "ballistic" *in materio* computers may not require all these components, in particular not the hybrid classical computer to control the programming.

If we want to shoehorn this architecture into a classical Turing-like model, we might think of the *in materio* component as an (interactive) "oracle". However, this relegation of most of the interesting computation into a mysterious black box seems to lose much of the power and interest of considering the material to be implementing a (nonclassical) computational model in its own right.

5. Biological substrates

The kinds of lightly-engineered bulk matter mentioned here all implement what might be called "diffusion communication" mechanisms: all interactions are with local, neighbouring material, and so information can only diffuse through the substrate. Diffusion is slow.

Wires (pipes, nerves) are a means to implement nonlocal communication: information can move in larger leaps through the substrate, resulting in faster communication, and more complex dynamics. As we begin to build models of the computational capabilities natural to materials, we should be in a position to consider the effect of varying degrees of inhomogeneity in the materials, including the capability for long-range interactions.

The ultimate in inhomogeneous natural computation material is biological material. It is easy to be misled by cartoon pictures of cells with a blob of nucleus floating in some thin liquid. Yet cells are intricately structured, with nested compartmentalization providing gating and control of bulk diffusion, and highly dynamic, with cytoskeletal dynamic infrastructure supporting system-wide migration, self-assembly and self-reproduction. The cartoon should look more like

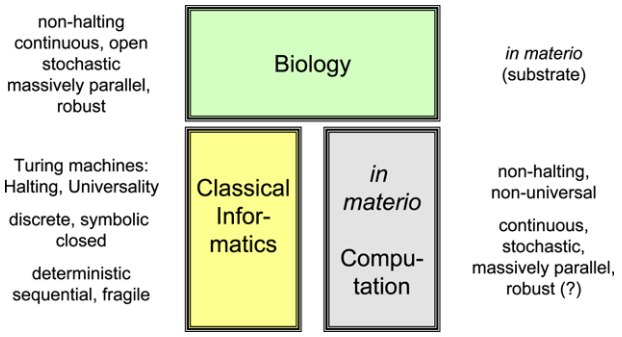

Fig. 2. The informatic (logical) and the material (physical) pillars of computation, supporting biology.

minestrone soup than salt water; inside the nucleus is more complicated still. Moreover, there are many emergent levels of structure and dynamics between basic physical or chemical processes and a fully functioning biological cell.

This is why starting at biology in an attempt to get computational models other than Turing machines seems too great a first step. A route that first starts with understanding the computational properties of (relatively) simple materials, in terms of their complex structure and dynamics, and then moves towards more complicated materials, ultimately to materials evolved to exploit information gradients, seems more likely to succeed. (I am by no means saying that researchers should stop working on biological computation from an applications perspective: that has a different goal.)

6. Summary and conclusions

Fig. 2 shows the "classical informatics" pillar of computation, the subject matter of conventional computer science. It also shows the neglected pillar of material computation, the computation that physical matter performs "naturally". The overarching subject of biological computation rests on both these pillars: evolved material exploiting an information gradient. I have argued that only by fully understanding both pillars will we be able to produce a theory of biological computation. And on the way up the material pillar, we will discover fascinating and powerful new models of material computation.

Acknowledgements

My thanks to Howard Wilson for introducing me to plasmas as complex systems; to Matthias Bechmann and Angelika Sebald for arguing that water could compute in an NMR context; and to Julian Miller and Martin Bates for making me look at liquid crystals in a new way.

Some of the ideas presented here are as a result of lively discussions with Simon Harding, Viv Kendon, Julian Miller, Klaus-Peter Zauner, and other participants of Unconventional Computing 2007 Bristol, UK, after presentation of a preliminary version. Thanks also to Leo Caves, Fiona Polack, and the anonymous referees for detailed comments.

References

[1] UKCRC Grand Challenge website: http://www.ukcrc.org.uk/grand_challenges/index.cfm.
[2] GC in Non-Classical Computation website: http://www.cs.york.ac.uk/nature/gc7/.
[3] S. Stepney, S.L. Braunstein, J.A. Clark, A. Tyrrell, A. Adamatzky, R.E. Smith, T. Addis, C. Johnson, J. Timmis, P. Welch, R. Milner, D. Partridge, Journeys in non-classical computation I: A grand challenge for computing research, Int. J. Parallel Emergent Distributed Syst. 20 (1) (2005) 5–19.
[4] S. Stepney, S.L. Braunstein, J.A. Clark, A. Tyrrell, A. Adamatzky, R.E. Smith, T. Addis, C. Johnson, J. Timmis, P. Welch, R. Milner, D. Partridge, Journeys in non-classical computation II: Initial journeys and waypoints, Int. J. Parallel Emergent Distributed Syst. 21 (2) (2006) 97–125.
[5] D. Deutsch, The Fabric of Reality, Penguin, 1997.
[6] S. Stepney, Non-classical hypercomputation, Int. J. Unconventional Comput. 5 (2009) (in press).
[7] S. Stepney, Embodiment, in: D. Flower, J. Timmis (Eds.), In Silico Immunology, Springer, 2007, pp. 265–288 (Chapter 12).
[8] H. Kitano, Biological robustness, Nat. Rev. Genetics 5 (11) (2004) 826–837.
[9] R. Landauer, Dissipation and heat generation in the computing process, IBM J. Res. Dev. 5 (1961) 183–191.
[10] F. Dyson, Time without end: Physics and biology in an open universe, Rev. Modern Phys. 51 (3) (1979) 447–460.
[11] S. Lloyd, Ultimate physical limits to computation, Nature 406 (2000) 1047–1054.
[12] L.M. Krauss, G.D. Starkman, Universal limits on computation, arXiv:astro-ph/0404510v2.
[13] L.B. Levitin, T. Toffoli, Thermodynamic cost of reversible computing, arXiv:quant-ph/0701237v2.
[14] J.A.S. Kelso, Dynamic Patterns: The self-organization of brain and behavior, MIT Press, 1995.
[15] A.K. Dewdney, On the spaghetti computer and other analog gadgets for problem solving, Sci. Amer. 250 (6) (1984) 19–26.
[16] R. Hwang, D. Richards, P. Winter, The Steiner Tree Problem, in: Annals of Discrete Mathematics, vol. 53, North Holland, 1992.
[17] W. Miehle, Link-length minimization in networks, Operations Research 6 (2) (1958) 232–243.
[18] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.
[19] P. Dittrich, J. Ziegler, W. Banzhaf, Artificial chemistries — a review, Artificial Life 7 (3) (2001) 225–275.
[20] D.E. Goldberg, Genetic Algorithms in Search, in: Optimization and Machine Learning, Kluwer, 1989.
[21] A. Colorni, M. Dorigo, V. Maniezzo, Distributed optimization by ant colonies, in: Proc. First European Conference on Artificial Life, MIT Press, 1992, pp. 134–142.
[22] L.M. Adleman, Molecular computation of solutions to combinatorial problems, Science 266 (11) (1994) 1021–1024.
[23] R.R. Burge, Protein-based optical computing and memories, Computer 25 (11) (1992) 56–67.
[24] R.R. Burge, Protein-based computers, Sci. Amer. 272 (3) (1995) 66–71.
[25] P. Fromherz, A. Offenhausser, T. Vetter, J. Weis, A neuron-silicon junction: A Retzius cell of the leech on an insulated-gate field-effect transistor, Science 252 (1991) 1290–1293.
[26] S. Tsuda, K.-P. Zauner, Y.P. Gunji, Robot control with biological cells, BioSystems 87 (2007) 215–223.
[27] J.F. Miller, K. Downing, Evolution *in materio*: Looking beyond the silicon box, in: Proc. NASA/DoD Conference on Evolvable Hardware, July 2002, Alexandria, VA, USA, IEEE Press, 2002, pp. 167–176.
[28] A. Turing, The chemical basis of morphogenesis, Philos. Trans. Roy. Soc. London B 237 (641) (1952) 37–72.
[29] A. Adamatzky, B.D.L. Costello, T. Asai, Reaction-Diffusion Computers, Elsevier, 2005.

[30] A. Adamatzky, B.D.L. Costello, Experimental logical gates in a reaction-diffusion medium: The XOR gate and beyond, Phys. Rev. E 66 (2002) 046112.
[31] J.W. Mills, M. Parker, B. Himebaugh, C. Shue, B. Kopecky, C. Weilemann, Empty space computes: The evolution of an unconventional supercomputer, in: Proc. 3rd conference on Computing Frontiers, Ischia, Italy, ACM Press, 2006, pp. 115–126.
[32] C.G. Langton, Computation at the edge of chaos, Physica D 42 (1990) 12–37.
[33] M. Mitchell, J.P. Crutchfield, P.T. Hraber, Dynamics, computation, and the 'edge of chaos': A re-examination, in: G.A. Cowan, et al. (Eds.), Complexity: Metaphors, Models, and Reality, Addison Wesley, 1994, pp. 497–513.
[34] J.P. Crutchfield, The calculi of emergence: Computation, dynamics, and induction, Physica D 75 (1994) 11–54.
[35] N. Murphy, T.J. Naughton, D. Woods, B. Henley, K. McDermott, E. Duffy, P.J.M. van der Burgt, N. Woods, Implementations of a model of physical sorting, in: From Utopian to Genuine Unconventional Computers, Luniver Press, 2006, pp. 79–100.
[36] S.L. Harding, J.F. Miller, A tone discriminator in liquid crystal, in: Proc. CEC 2004: International Conference on Evolutionary Computation, June 2004, Portland OR, USA, IEEE Press, 2004, pp. 1800–1807.
[37] S.L. Harding, J.F. Miller, Evolution *in materio*: A real-time robot controller in liquid crystal, in: Proc. NASA/DoD Conference on Evolvable Hardware, Washington DC, USA, IEEE Press, June 2005, pp. 229–238.
[38] S.L. Harding, J.F. Miller, E.A. Rietman, Evolution in materio: Exploiting the physics of materials for computation, arXiv:cond-mat/0611462.
[39] M. Bechmann, J.A. Clark, A. Sebald, S. Stepney, Unentangling nuclear magnetic resonance computing, in: UC'07 [49], pp. 1–18.
[40] K.-P. Zauner, M. Conrad, Parallel computing with DNA: Toward the anti-universal machine, in: Proc. Parallel Problem Solving from Nature, PPSN IV, in: LNCS, vol. 1141, Springer, 1996, pp. 696–705.
[41] E. Blakey, On the computational complexity of physical computing systems, in UC'07 [49], pp. 95–115.
[42] D. Swade, The Cogwheel Brain: Charles Babbage and the Quest to Build the First Computer, Little, Brown, 2000.
[43] E. Shapiro, A mechanical Turing machine: Blueprint for a biomolecular computer, in: E. Winfree, D. Gifford (Eds.), Proc. Fifth International Meeting on DNA Based Computers, American Mathematical Society, 1995, pp. 229–230.
[44] W.M. Stevens, A kinematic Turing machine, in: UC'07 [49], pp. 29–62.
[45] C.A.R. Hoare, Communicating Sequential Processes, Prentice Hall, 1985.
[46] R. Milner, Communicating and Mobile Systems: The π-calculus, Cambridge University Press, 1999.
[47] P. Wegner, Why interaction is more powerful than algorithms, Comm. ACM 40 (5) (1997) 81–91.
[48] S. Dasmahapatra, J. Werner, K.-P. Zauner, Noise as a computational resource, Int. J. Unconventional Computation 2 (4) (2006) 305–320.
[49] Unconventional Computing 2007, Bristol, UK, July 2007, Luniver Press, 2007.