Chapter 5
Physical and Formal Aspects of Computation: Exploiting Physics for Computation and Exploiting Computation for Physical Purposes

Bruce J. MacLennan

Abstract Achieving greater speeds and densities in the post-Moore's Law era will require computation to be more like the physical processes by which it is realized. Therefore we explore the essence of computation, that is, what distinguishes computational processes from other physical processes. We consider such issues as the topology of information processing, programmability, and universality. We summarize general characteristics of analog computation, quantum computation, and field computation, in which data is spatially continuous. Computation is conventionally used for information processing, but since the computation governs physical processes, it can also be used as a way of moving matter and energy on a microscopic scale. This provides an approach to programmable matter and programmed assembly of physical structures. We discuss *artificial morphogenesis*, which uses the formal structure of embryological development to coordinate the behavior of a large number of agents to assemble complex hierarchical structures. We explain that this close correspondence between computational and physical processes is characteristic of *embodied computation*, in which computation directly exploits physical processes for computation, or for which the physical consequences of computation are the purpose of the computation.

5.1 Introduction

"Unconventional computation" is, of course, a negative term, and is defined by reference to "conventional computation," which is quite specific. Characteristics of conventional computation include digital (in fact binary) data and program representation, von Neumann architecture, (primarily) sequential program execution,

B.J. MacLennan (✉)
Department of Electrical Engineering and Computer Science,
University of Tennessee, Knoxville, TN, USA
e-mail: maclennan@utk.edu

A. Adamatzky (ed.), *Advances in Unconventional Computing*,
Emergence, Complexity and Computation 22,
DOI 10.1007/978-3-319-33924-5_5

addressable random-access memory, information processing implemented through sequential electronic binary logic, irreversible operations, classical (non-quantum) operation, etc. Unconventional computation may be defined, then, as computation that differs in one or more of these characteristics.

Given the success of conventional computation, it is reasonable to ask the reasons for studying unconventional computation. One motive is purely scientific: we would like to understand the full range of computational processes, in natural systems as well as in computers. Information processing is widespread in nature, but for the most part natural computation does not have the characteristics of conventional computation, and therefore we need to understand computation in a broader sense.

5.1.1 Post-Moore's Law Computation

The second motive for studying unconventional computation is technological, for it is apparent that Moore's Law must come to an end. First, the atomic structure of matter places limits on the smallness of electronic components and the density with which they can be assembled. Moreover, it is likely that economics will defeat Moore's Law even before it reaches these physical limits [60]. Therefore, in the post-Moore's Law world, progress in computation will depend on processing information in new ways, that is, on unconventional computation. The end of Moore's Law is on the horizon, and so it is important that we develop post-Moore's Law technologies to the point of practicality before the end is reached.

What might be the characteristics of post-Moore's Law computation? Conventional computer technology has benefitted from clearly separated hierarchical levels. Programming abstractions, such as data structures, are implemented in terms of primitive data elements, such as floating-point numbers and pointers, which are implemented in terms of many bits, each of which is represented by many electrons. Similarly, conceptually primitive operations, such as floating-point division, may be implemented by iterative algorithms, themselves implemented in sequential logic. In particular, the Boolean logic level is largely independent from those above and below it. That is, on one hand, Boolean logic can be used to implement various computer architectures, and on the other, Boolean logic can be implemented in many different technologies (e.g., relay, vacuum tube, transistor, VLSI).

Computing abstractions are implemented in terms of lower-level abstractions, and ultimately in the laws of physics, but post-Moore's Law computing technologies cannot afford these multiple hierarchical levels. To permit greater densities and speeds, computing abstractions and physical laws will need to be brought closer together, but we cannot change the laws of physics, so this assimilation of computation and physics will have to be accomplished by developing computing paradigms that are more like the laws of physics. Therefore post-Moore's Law computing will have more of the characteristics of the underlying physical processes.

For example, the laws of physics are fundamentally concurrent; individual particles respond in parallel to fields, forces, and other particles. Therefore, we expect parallel computation to be the norm in the post-Moore's Law era.

The laws of physics are expressed in differential equations (or partial differential equations), which describe continuous change in continuous quantities. Therefore, analog computation can be expected to increase in importance. Often operations can be implemented in a few analog components, which would require many digital components (see Sect. 5.3.2 below). Therefore analog representations are preferable for achieving higher densities.

It might be objected that quantum mechanics applies at the smallest scales, and therefore that digital computation is better matched to the physics at these scales. It is true that at very small scales certain quantities, such as charge, spin, and energy, are quantized. On the other hand, quantum wave functions are continuous functions of space and time, and the Schrödinger equation is a differential equation. Even qubits are continuous linear combinations of the basis states.

Conventional computation takes place in *discrete* or *sequential time* (see Sect. 5.2.3 below), in which operations take place in sequence at discrete times. (Parallel computation does not contradict the essentially sequential execution of digital computation.) At both the classical and the quantum levels, however, the laws of physics are expressed by differential equations. Therefore, as our computational processes become more like physical processes, we expect continuous-time processes to play an increasing role in post-Moore's Law computation.

As we approach very small scales, noise, uncertainty, defects, imperfections, and faults all become more likely, and ultimately unavoidable. Therefore, in the post-Moore's Law era we will have to abandon the idea that we are striving for systems that approximate ever more closely an ideal, which is perfect, noiseless, fault-free, etc. Rather, we will take these phenomena as a given, and design systems that exploit them rather than trying to avoid or mitigate them. Natural computation, which we find in living systems, has much to teach us about exploiting physical phenomena for robust and efficient information processing. For example, "noise" can be reconceptualized as a source of *free variability*, which can be used for escaping from local optima and for many other purposes [31, 37].

5.2 The Essence of Computation

Given a growing assimilation of computation to physics, one might wonder what distinguishes computational processes from other physical processes. Is every physical process a computation? What common properties distinguish information processing, computation, and control from other physical processes [40]?

All of these computational processes use physical arrangements (physical *form*) to represent something else, and use physical rearrangement (*transformation*) to represent some abstract process. What distinguishes a computation from other physical processes is that in principle the goals of the process could be achieved by any phys-

ical system that realized the same abstract transformation (a property called *multiple realizability*) [29, 35]. That is, while computation must be physically realized, it does not depend essentially on a particular physical realization.

5.2.1 The Four Whys of Computation

The preceding observation raises several issues, which are addressed best in terms of Aristotle's "four whys" (commonly known as his "four causes") (Aris., *Phys.* II 194b–195a, *Met.* 983a–b, 1013a–1014a). These are four sorts of answers to the question of *why* an object or process is what it is. In general, none of the four is sufficient on its own, and they differ in explanatory value depending on the subject matter and motivation of the question. His taxonomy is useful for classifying explanations not only in biological systems but also in artifacts, and therefore it can be applied to both natural and artificial information processing and control [40].

One of the *whys* answers the question: *What* is this? Traditionally it is called the *formal cause* because it accounts for an object or process in terms of its form or pattern: the formula that describes it. The second *why* answers: *From what* is this made? It is known traditionally as the *material cause*, since its explanation is in terms of the unformed stuff which gains its specific properties through the formal cause. These two *whys* are central to our topic, since information is realized by material forms, and information processing by physical rearrangement of these forms [24]. Although computation requires *some* material realization of its formal processes, it is independent (qua computation) of its *particular* material realization. Moreover, form and matter are relative terms, and the formed matter at one level of organization can provide the unformed matter for a higher level. For example, in conventional computers the addressable bytes and basic operations are the medium that software formally organizes, but the bytes and operations are themselves organized structures of lower-level objects and processes (logic gates).

The third *why* answers: *By what* is this object or process created and sustained? It is the familiar *efficient cause*. Computation must be powered, either by an initial supply of energy or by a continuing supply. Without its efficient cause, the computation is a potentiality that is not actualized.

The last *why* answers the question: *For the sake of what* does this object exist or this process take place? Traditionally it is called the *final cause* since it addresses the end, goal, or purpose of something. Artifacts, such as manufactured computers, are designed for some purpose, but biological systems also have functions that they fulfill. The heart circulates the blood, the immune system fights infections, the nervous system coordinates behavior and cognition, and so on.

The final cause is essential to the definition of computation, for what distinguishes computational and information processing systems from other physical systems is that their goal or function can be performed by any system with the same formal structure, independent of its material realization [31, 38, 40]. That is, their function is *information* processing as opposed to some other physical process. One test

of whether a system is computational is to ask whether it could be replaced with another system with the same formal structure and still achieve its ends (i.e., whether it is multiply realizable). Of course, many natural systems serve multiple functions (e.g., circulation of the blood distributes oxygen, but also hormones, which transmit information), and so they might not be purely computational [40]. From this perspective, it is remarkable the extent to which the function of the nervous system is pure computation and control.

The relationship between a computational or information processing system and its physical realization can be expressed by the *realization homomorphism* [31], which says that a physical system realizes an abstract computation if there is a homomorphism from the physical system to the abstract system. The significance of the homomorphism is that it preserves some of the algebraic structure of the physical system, but not all of it. This captures our intuition that the physical system can have many properties that are irrelevant to its realization of the abstract system. However, we must also recognize that many realizations are only approximate. For example, the abstract computation might involve real numbers that are only approximately realized by floating-point numbers in a physical computer.

5.2.2 The Topology of Information

Rolf Landauer reminded us that information is physical; it must be represented in some physical medium [24]. But its essence—what makes it *this* information versus *that* information—lies in its form. Therefore differences of information are differences of abstract form, which can be described by topology, the science of abstract form and similarity.

The realization homomorphism $\mathcal{H}$ is a surjection mapping the physical state space S onto the abstract state space Σ, that is, $\mathcal{H} : S \twoheadrightarrow \Sigma$. In the physical state space we distinguish the *information-bearing degrees of freedom* (IBDF) from the *non-information-bearing degrees of freedom* (NIBDF) [4]. For example, the IBDF may be macrostates representing, for example, the bits 0 and 1, while the NIBDF may include the positions, momenta, etc. of individual particles, which do not represent information and which manifest as heat, noise, etc. The IBDF are managed by the computational process in order to realize the computation, but the NIBDF are not managed or are managed only in aggregate to keep them from interfering with the computation. Let $E \subset S \times S$ be the equivalence relation between physical states that are equivalent in their IBDF, and define the quotient space $Q = S/E$, which represents the IBDF. Then the realization homomorphism can be factored $\mathcal{H} = \mathcal{A} \circ I$, where $I : S \twoheadrightarrow Q$ is a surjection from the physical state space onto the IBDF, and $\mathcal{A} : Q \leftrightarrow \Sigma$ is a bijection between the IBDF and the abstract state space.

Conventional computation is digital; it uses discrete representations of information. Formal models of digital computation, such as finite-state machines and Turing machines, use finite, discrete alphabets of symbols or states, in which each element is identical to itself and completely different from every other element. This is a

discrete topology (Σ, δ), in which the metric is defined $\delta(x, x) = 0$ for $x \in \Sigma$ and $\delta(x, y) = 1$ for all $y \neq x$. There are only two possible distances in a discrete metric space. A finite discrete space with 2^n elements is homeomorphic to the space $\{0, 1\}^n$ with the ∞-product metric[1]; this is of course the basis of binary representations on digital computers. Other, less trivial, topologies can be defined over these discrete spaces for the purposes of computation. For example, the discrete space $\{0, 1\}^n$ can represent the integers $\{0, 1, \ldots, 2^n - 1\}$ with their usual metric.

Traditionally, analog computers have operated on bounded real numbers represented by a physical quantity, but they are also capable of operating on other continuous quantities, such as complex numbers (represented, for example, by the phase and amplitude of a periodic signal). Moreover, quantum computers operate on complex linear superpositions of basis states (e.g., $z|0\rangle + z'|1\rangle$, with $z, z' \in \mathbb{C}$). Further, *analog field computers* can operate on *fields* [33], that is, spatially continuous distributions of continuous quantity (see Sect. 5.3.4). Images and continuous signals are examples of fields. All of these information spaces are *continua*, which may be defined formally as connected second-countable metric spaces. Second-countability means that they have a countable dense subset (as, for example, the rationals are a countable dense subset of the reals).

More generally, unconventional computers may be hybrid, that is, capable of operating on both discrete and continuous information spaces. These are products of spaces that are individually discrete or continuous spaces. The U-machine model encompasses both digital and analog computing (Sect. 5.3.5).

5.2.3 The Topology of Information Processing

Computation takes place in time. In conventional digital computation, operations are performed at discrete points in time. This is properly described as *sequential-time* computation since there is no implication that the operations be performed at regular time intervals, as in *discrete-time* computation [66]. More generally computation may be described by a partial order defining how later computations depend on earlier ones, thus permitting operations to be performed concurrently.

Analog computation can also be defined over either discrete or sequential time, as in the BSS model of computation over the reals [5]. Most artificial neural network models are sequential-time analog computations, since the neural operations do not happen at specific times, but require only that their inputs be available. Even most recurrent neural networks operate in sequential time, since the sequence of their outputs depends only on the sequence of their inputs. Neural networks can of course operate in discrete time, but only if the neural computations are clocked.

Traditionally, however, analog computers have operated in continuous time, integrating differential equations, which serve as programs. But there is also a continuous

[1]The ∞-product metric on the Cartesian product of two spaces (X, δ_1), (Y, δ_2) is defined $\delta_\infty[(x, y), (x', y')] = \max[\delta_1(x, x'), \delta_2(y, y')]$.

version of sequential computation, in which the computation is defined over a set of "instants" homeomorphic to a closed or half-open interval of the real numbers. That is, sequence is defined, but not duration or rate. More generally, both discrete and continuous concurrent computation can be defined by a partial order that defines the dependence of later computations on earlier ones. Operations are permitted to take place sequentially or concurrently, so long as this partial order is respected.

5.2.4 Programmability

Programmability is an important property of many computation systems. A system is *programmable* if it is capable of performing a wide range of functions depending on some finite systematic external specification (a *program*). Programmable systems are valuable because they can be used for many different purposes and their behavior can be adapted to changing circumstances, simply by changing the program.

Computational programs are usually described textually; for example sequential-time computations (digital or analog) are described by programs in programming languages, and continuous-time computations are described by ordinary or partial differential equations. Programs can also be described by diagrams, such as flow-charts for sequential-time computations and block diagrams for continuous-time processes.

The preceding are examples of discrete programs, but continuous programs are also possible. For example, some analog computers permit functions to be described by continuous graphs (Sect. 5.3.2.1). Moreover, many useful computations can be defined as *relaxation processes* in which the state descends a potential surface to approach an attractor, which is the solution to the problem (Sect. 5.3.2.3). Such processes may be continuous-, discrete-, or sequential-time depending on how the state changes as the system computes. In these cases the program, which governs the computation, is defined by the potential surface, which therefore defines a continuous program, which we might call a *guiding image* [29, 31]. The metaphors are different: instead of *writing a program*, we could say we are *drawing* or *sculpting a guiding image*. While it is certainly possible to create such a continuous program manually, more likely it will emerge from an adaptive or training process, as happens in artificial neural networks.

5.2.5 Universality

Programmability raises the issue of universality: Is a computer capable of computing anything, given an appropriate program? For example, we know the Universal Turing Machine (UTM) is capable of computing any Turing-computable function. That is, for any Turing machine there is a UTM plus program combination that is *equivalent*

(computes the same function). However, we must be careful applying these familiar ideas to unconventional computation.

When comparing the power of different models of computation, it is important to remember that all models are idealizations of the things they are modeling, and these idealizations are intended to make the model more tractable than the original system *for some purpose*. Therefore, each model exists in a (usually implicit) *frame of relevance*, which delimits the sort of questions it is suited to answer [31, 35]. A model cannot be expected to give useful answers when applied outside of its frame of relevance; indeed, the answers are often misleading, more a reflection of the model than the system under investigation.

Therefore, while it is very tempting to compare various models of unconventional computation to the Church-Turing model, we must be cautious doing so. This model was developed to address questions of effective calculability in the foundations of mathematics, and they delimit its primary frame of relevance. It makes many idealizing assumptions, such as that tokens are discrete and can be perfectly discriminated from their background and classified as to type, etc. [31]. Two machines or programs are considered equivalent in power if they compute the same input–output function. Efficiency in analyzed in terms of asymptotic complexity, which ignores constant scale factors. And so forth.

While the conventional theory of Church-Turing computation has proved enormously fruitful, there are many important issues that are outside of its frame of relevance. For example, an important question in natural computation is how brains are able to process complex, noisy sensorimotor information in real time using relatively slow, low-precision computing devices (neurons). The conventional theory of computation is not equipped to deal with issues in real-time control. Further, asymptotic analysis is not very useful because (1) the constants matter, and (2) the size of the input is usually bounded. Therefore, in many of the contexts in which unconventional computation is relevant, such as natural computation and post-Moore's Law computation, the idealizing assumptions of Church-Turing computation are inappropriate, and different models, which make different assumptions, are more useful [31]. In these contexts, it is not usually appropriate to consider two computations equivalent solely because they compute the same function, and therefore it is not very useful to measure the power of a model of computation in terms of the class of functions it computes. This is only one of the criteria by which models of computation can be compared. In the context of unconventional computing there are many dimensions for comparing the capability of computational models.

5.3 Computation for Formal Ends

In order to understand the full range of unconventional computation, it is useful to explore the relation between the computational processes and their physical realizations. In this section we address computation in its usual sense, wherein the principal goal is an abstract information process, and the realization is a means to this end.

That is, the *material* processes are serving *formal* purposes. In Sect. 5.4 we consider the opposite situation, which is less familiar.

5.3.1 General Considerations

What are the requirements for unconventional realizations of abstract information and control processes or computations? In general, any reasonably controllable, mathematically describable physical process can be used for computation, including living systems, such as slime molds and bacterial mats [1]. We can outline some more specific considerations [35]. First we need a physical process that has at least the algebraic structure of the desired computation, so that the realization homomorphism holds. Therefore, we need to have sufficient control over the physical arrangements to implement the required structure. For general-purpose computation, we will want some flexibility in making these arrangements, so that any computation in a useful class can be implemented. In this case, we also may consider programmability, that is, whether there is some systematic way to set up the physical process in accord with an abstract description (the program).

Of course, the application may place additional restrictions on the class of admissible realizations. For example, some physical processes might be too slow or consume too much energy for the application. On the other hand, many potential applications do not require high speed, and a slower physical process, which is better matched to the application requirements, may have other advantages, such as energy efficiency, power source, stability, robustness, programmability, or precision. Moreover, many applications do not require high precision or faultless operation, and computation and control in nature provide many examples of how to tolerate and even exploit noise, errors, faults, imprecision, defects, indeterminacy, etc. For example, they can be used as sources of *free variability* for escaping from local optima, breaking deadlocks, driving exploration, etc. [41].

Useful computations require *transduction*, that is, the transfer of information from the environment into the computation, and the transfer of information and control from the computation back out into the environment [31, 35]. Both computation and transduction involve the formal and material aspects of physical processes. Computation, as we've seen, is *generically* realizable; that is, it can be realized by any physical process with the required formal structure. On the other hand, transduction provides the interface between the computational medium and *specific* physical systems (e.g., a photoreceptor or temperature sensor for input, an LED display or servomotor for output). In principle, a pure input transducer transfers the form from one material (the input medium) to another material (the computational medium), and a pure output transducer transfers the form from the computational medium to the output medium. In practice, pure transducers are rare, for there is usually some (intended or unintended) change in the form in addition to the intended material change; for example, the input might be filtered, digitized, limited, etc. Thus most

transducers combine some information processing or computation with the change of medium.

5.3.2 Analog Computation

Analog computation is an important unconventional computing paradigm. Since the laws of physics are continuous, it is likely to become more important in the post-Moore's Law era, because it can be more directly realized [44]. In principle, any continuous physical quantities can be used as a medium for analog computing. Electronic analog computing, in which real numbers are represented typically by current, voltage, or charge, is most familiar, but there are many other possibilities. For example, mechanical analog computers have represented numbers by angular or linear displacement. Concentrations of substances that are continuous or approximately continuous can be used (as in reaction-diffusion computation [2]). Light is an attractive medium [3].

In choosing an analog computation medium, we must also consider the physical realization of the abstract operations required by the computation (e.g., addition, subtraction, multiplication, integration, filtering, various transcendental functions). The virtue of analog computation is that common, useful operations often have simple realizations. For example, addition can be performed by simply combining currents, charges, or light intensities; integration can be performed by charging a capacitor or by accumulation of a chemical reaction product.

One critical question in any analog computing technology is *precision*, which refers to the quality of a representation. Precision has two major components: *resolution*, which refers to the fineness of the representation, and *stability*, which refers to its ability to maintain its value over the duration of the computation. Precision can be expressed as a fraction of *full-scale variation* of a variable (the difference of its maximum and minimum values). Doubling the precision of an analog representation or computation can be very expensive compared to doubling digital precision (add one more bit), since it requires higher quality devices [32]. Fortunately, high precision is not required for many applications and for some approaches to analog computing, such as neural networks. In general, natural computation provides many examples of the utility of low-precision analog computing.

5.3.2.1 Programming Techniques

Certain basic operations are simple to implement in many analog computing technologies. As mentioned, direct combination of physical quantities can often be used to implement analog addition, $u(t) = v(t) + w(t)$. Some physical quantities are signed (e.g., voltage, current, charge) and can be used directly to represent signed quantities, others are not (e.g., intensities, concentrations of chemicals). In the latter case, signed quantities can be represented as differences of positive quantities. That is,

instead of one signed variable $v(t)$, we use two non-negative variables, $v^+(t)$ and $v^-(t)$, that implicitly represent $v(t) = v^+(t) - v^-(t)$. The analog algorithm must be re-expressed in terms of the differential quantities. Given a signed representation, subtraction $[u(t) = v(t) - w(t)]$ and negation $[u(t) = -v(t)]$ are easy to implement.

Positive constant multiplication, $u(t) = cv(t)$ for $c > 0$, can be implemented by passive attenuation or active amplification. Signed constant multiplication can be implemented directly or in terms of the signed operations. The assumption here is that the scale factor c must be programmed, either externally (e.g., by adjusting a potentiometer) or internally (e.g., by programming a floating-gate transistor), and that this is a relatively slow process, which might not be under analog program control. Therefore, we contrast it with full variable multiplication, $u(t) = v(t) \times w(t)$, in which both factors can be the result of ongoing analog computation. Direct analog implementation can be more difficult than constant multiplication, but it can be accomplished. For example, a squaring operation can be used to implement multiplication by [56, p. 92]:

$$v \times w = 0.25[(v + w)^2 - (v - w)^2].$$

Squaring can be implemented directly without multiplication [56, chap. 3]. This illustrates an important principle of analog computing: we cannot transfer our digital intuitions about what is simple into the analog domain. In the analog domain, apparently complicated operations, such as square, square-root, logarithm, and exponential, can have more direct implementations than apparently simpler operations, such as multiplication. Certain nonlinear and transcendental functions can be built into an analog computer as basic operations.

Division, $u(t) = v(t)/w(t)$, has to be handled carefully, since a small divisor can saturate the quotient register. Similarly, although analog implementation of differentiation, $u(t) = \dot{v}(t)$, is generally simple, the operation is problematic since it is sensitive to high-frequency noise, which it amplifies. One solution is to apply a low-pass filter to the differentiator's input. Alternatively, analog computations involving differentiation can be recast as integrations.

Integration usually has a straightforward implementation as the accumulation of some quantity:

$$u(t) = u_0 + \int_0^t v(\tau)\mathrm{d}\tau.$$

The integrator is initialized to the constant of integration u_0 at the beginning of the computation. This implements a differential equation $\dot{u}(t) = v(t)$ with an initial value $u(0) = u_0$.

For some applications (such as real-time control programs) the integration will be with respect to real time. In others, time in the analog computer will be independent of time in the abstract computation; it might integrate slower or faster. To ensure accurate results, the rate of analog integration has to be considered, since if it is too fast it may exceed the high-frequency response of the integrator, and if it is too slow, quantities will drift. Therefore analog integration often involves *time scaling*,

in which time t in the computer is related to time τ in the abstract computation by $t = b\tau$ for some $b > 0$. To integrate the abstract differential equation $\dot{u}(\tau) = v(\tau)$, that is, $u(\tau) = \int_0^\tau v(\tau')\mathrm{d}\tau'$, the analog computer uses the scaled integration $u(t) = b^{-1} \int_0^t v(t')\mathrm{d}t'$. In electronic analog computers this can be accomplished by decreasing the integrator input gain by a factor of b.

Since analog computing represents abstract quantities directly by physical quantities, *magnitude scaling* is another important consideration. A variable x in the abstract computation, with a certain range of values, must be mapped into a physical quantity v, with a dynamic range and precision limited by the physical device. Exceeding the device's operating range can lead to inaccuracy through distortion. Magnitude scaling is accomplished by choosing a scale factor, $v = ax$, which is small enough to stay within the device's dynamic range, but not so small that important differences are less than the device's resolution. Therefore, the variables in the abstract computations have to be scaled, and differential equations (or integrations) need to be adjusted to incorporate the scale factors. Moreover, in addition to the explicit variables, there are implicit variables corresponding, for example, with derivatives $\dot{x}$, $\ddot{x}$, etc. These too need to be scaled with the equations adjusted accordingly.

Some analog computers provide tunable band-pass filters, which can be used to perform a discrete Fourier transform on a signal. Others provide analog matrix–vector multiplication in which the elements of the matrices and vectors are continuous quantities, and the multiplications and additions are implemented by analog computation. That is, $\mathbf{u}(t) = \mathrm{M}\mathbf{v}(t)$, where $u_j(t) = \sum_{k=1}^n M_{jk} v_k(t)$. This operation can be used to implement linear operators, such as filters. Another useful operation is a noise generator, which produces Gaussian white noise, which can be adjusted and filtered to have desired characteristics. Randomness is useful in some analog algorithms. Simple decision making can be implemented by sigmoid functions:

$$\sigma(s) = \frac{1}{1 + e^{-\beta s}}.$$

Then a differential equations such as $\dot{x} = \sigma(s - \theta)F(x, y, \ldots) + \sigma(\theta - s)$ $G(x, y, \ldots)$ will be governed by $F(x, y, \ldots)$ if s is above the threshold, $s > \theta$, and by $G(x, y, \ldots)$ if $s < \theta$, with β controlling the sharpness of the transition.

Some analog computers provide means for computing arbitrary functions by means of a continuous version of table lookup. This mechanism allows the computation of functions for which there is no known closed-form description, or that would be too complicated to compute from their closed forms. To implement such a function, its graph $\{(x, f(x)) | x \in [x_{lwb}, x_{upb}]\}$ is represented in a suitable two-dimensional medium. When this medium is loaded in the computer, it can compute $u(t) = f[v(t)]$. Similarly, an arbitrary binary function g can be computed by representing its graph $(x, y, g(x, y))$ in a suitable three-dimensional medium. These are examples of guiding images, i.e., continuous representations of analog algorithms (Sect. 5.2.4 above).

5.3.2.2 General-Purpose and Universal Computation

Universality is an important question for any computing paradigm, for it tells us what are the minimal requirements for performing any computation in a large class of possible computations. Claude Shannon proved fundamental universality theorems for the differential analyzer, which were completed, corrected, and extended by Pour-El, Lipshitz, and Rubel [25, 57, 62, 63].

A related question is the power of analog computing relative to Turing computability, but it presents an immediate paradox. On the one hand, it is easy to show that the ability to operate on arbitrary real numbers confers super-Turing power (e.g., there is a real constant whose bits encode the solutions to the Halting Problem). On the other hand, analog computers are routinely simulated on ordinary digital computers, suggesting that analog computers have no more than Turing power. There are a variety of theorems in the literature, proving sub-Turing, Turing, or super-Turing power depending on the premises (representative citations can be found elsewhere [44]). The resolution of these apparently contradictory conclusions is that analog computation is not in the frame of relevance of Church-Turing computation (recall Sect. 5.2.5), and therefore the results are more a reflection of the idealizing assumptions of the various models than of the computational systems being modeled (more details are provided elsewhere [35]).

There are a number of ways to program analog computers. Sequential analog computations can be described in programming languages similar to those for digital computers, the principal difference being that the primitive operations are analog rather than digital. However, some caution is necessary. For example, exact equality and inequality tests, which are unproblematic in digital computation, may be infeasible in the analog domain, where infinite precision would be required. In the context of analog computing, it is more reasonable to test if the difference of two numbers is less than some ε.

Continuous-time analog computations are most often described by differential equations. They are also represented by block diagrams in which the differential equations are recast as explicit integrations (e.g., Fig. 5.1).

In principle, analog programs can contain constants that are not rational or even Turing-computable. Such constants cannot be represented finitely in discrete symbols, but they can be represented directly as continuous quantities. In a practical sense, however, due the limited precision of analog computing, constants can be represented digitally to the accuracy required. Nevertheless, it is important to broaden the notion of a program to include representations that are not textual, such as guiding images (Sect. 5.2.4 above).

5.3.2.3 Dynamical Systems

Dynamical systems are an attractive approach to analog computation; the system is defined so that the point attractors are solutions to the problem. Examples include analog solutions to traditional digital problems, such as sorting [8] and Boolean sat-

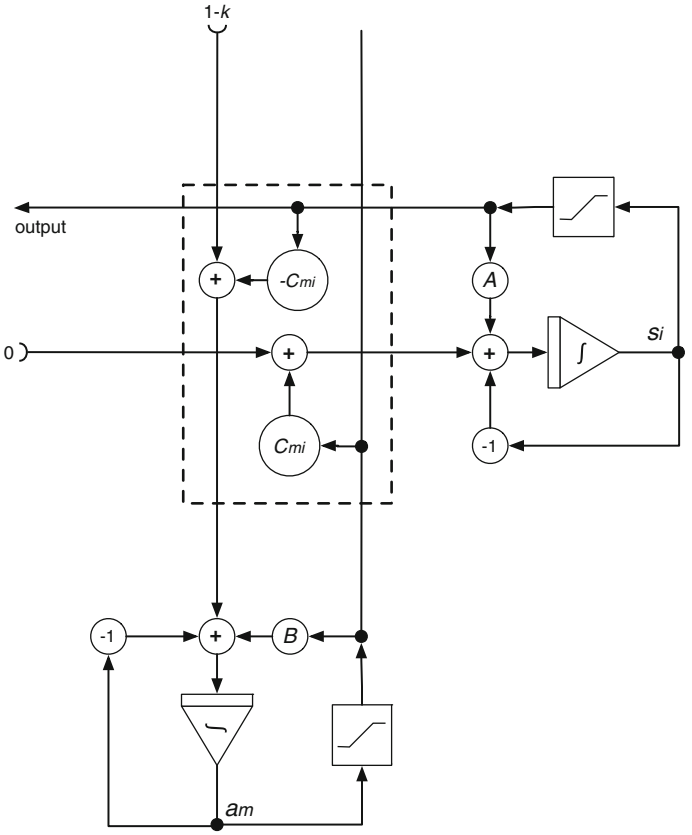

Fig. 5.1 Example analog algorithm implementing a dynamical system Boolean satisfiability [12]. The block enclosed in *dotted lines* is repeated for $m = 1, \ldots, M$ and $i = 1, \ldots, N$

isfiability [17, 48]. In the latter case, to solve a k-SAT Boolean satisfiability problem with M clauses and N variables, Ercsey-Ravasz and her colleagues define a dynamical system by the differential equations:

$$\dot{s}_i(t) = -s_i(t) + Af[s_i(t)] + \sum_{m=1}^{M} c_{mi} g[a_m(t)],$$

$$\dot{a}_m(t) = -a_m(t) + Bg[a_m(t)] - \sum_{i=1}^{N} c_{mi} f[s_i(t)] + 1 - k.$$

A particular problem instance is defined by the c_{mi} matrix elements: $c_{mi} = +1$ if variable i is positive in clause m, $c_{mi} = -1$ if variable i is negative in clause m, and $c_{mi} = 0$ if variable i is not in clause m. The f and g activation functions are linear

squashing functions that map the s and a values into $[-1, 1]$ and $[0, 1]$, respectively. The s_i converge on a solution to the problem, if one exists.

Figure 5.1 displays an analog algorithm for implementing this dynamical system [12]. The overall structure is a cross-bar between the M integrators for the a_m and the N integrators for the s_i; thus $M + N$ integrators are required. A particular instance is programmed by setting the c_{mi} and $-c_{mi}$ connections as required for the problem. The integrators are initialized to small values to start the computation; non-zero offset or noise in the hardware integrators might have the same effect.

5.3.3 Quantum Computation

Quantum computation is another promising approach to post-Moore's Law computing. Because the units of information representation are *qubits* (quantum bits), it is often supposed that quantum computation is a species of digital computation, but in fact it is hybrid analog–digital computation. Quantum computation gets its power from being able to operate simultaneously on superpositions (complex linear combinations) of digital basis states. Quantum operations are unitary operators that operate on the continuous complex coefficients of the basis states. Fundamentally, "binary" quantum computation is computation over finite-dimensional complex vector spaces. One of the remarkable properties of quantum computation, which gives it an advantage over classical analog computation, is that it is possible to do error correction to eliminate noise in the complex coefficients [52, Sect. 10.6.4]. Some quantum computation takes place in continuous time, such as adiabatic quantum computing and quantum annealing [13, 61]. *Continuous-value quantum computation* is another approach to analog quantum computation [26].

5.3.4 Field Computation

While ordinary differential equations (ODEs) are adequate for describing some systems, spatially extended systems normally require partial differential equations (PDEs). Although most historical analog computers processed ODEs, already in the nineteenth century there were developments such as the "field analogy method" [33, 44]. Sometimes the state was represented in a continuous medium, such as a rubber sheet or an electrolytic tank; in other cases a sufficiently dense array of discrete components was used. Therefore, we may define a *field* as either a spatially continuous distribution of continuous quantity, or a discrete distribution that is sufficiently dense to be treated as continuous. (Physicists similarly distinguish *physical fields*, which are literally continuous, such as electromagnetic fields, from *phenomenological fields*, which can be treated as though continuous, such as fluids.) Thus we can have real- or complex-valued scalar fields or vector fields (more generally, fields over any continuous algebraic field).

Field computation, then, may be defined as computation in which the state is represented by one or more fields [27, 33]. It is also a natural way of describing image processing or other computation with spatially extended data, and field computers have operations, such as convolution, that operate in parallel on entire fields. The original motivation for the theory of field computation was to describe neural information processing in regions of cortex large enough to be considered fields (typically 0.1 mm^2 or larger) and in neurocomputers with comparable numbers of spatially organized neurons [27].

Mathematically, fields are treated as continuous functions over some spatial domain Ω. More precisely, they are elements of a Hilbert space of square-integrable functions on Ω, which we denote $\Phi(\Omega)$. Its metric is determined by the inner product; for $\phi, \psi \in \Phi(\Omega)$,

$$\langle \phi \mid \psi \rangle = \int_\Omega \overline{\phi(u)} \psi(u) \mathrm{d}u,$$

where $\overline{\phi(u)}$ denotes the complex conjugate (in case the fields are complex-valued). *Field transformations* are functions (linear or nonlinear) that map fields into fields; that is, they are operators on Hilbert spaces. One especially useful field transformation is the *field product* $\Psi\phi \in \Phi(\Omega')$, for $\Psi \in \Phi(\Omega' \times \Omega)$ and $\phi \in \Phi(\Omega)$, which is defined by the Hilbert–Schmidt integral, $(\Psi\phi)(u) = \int_\Omega \Psi(u, v)\phi(v)\mathrm{d}v$, for all $u \in \Omega'$. It is the field analog of a matrix–vector product. The outer product also has a field analogue: if $\phi \in \Phi(\Omega)$ and $\psi \in \Phi(\Omega')$, then $\phi \wedge \psi \in \Phi(\Omega \times \Omega')$ is defined $(\phi \wedge \psi)(u, v) = \phi(u)\psi(v)$. Other useful operations include the gradient, Laplacian, convolution, cross-correlation, and point-wise arithmetic operations between fields.

Two questions immediately arise: Are there universal field computers? And (more practically), what operations should be provided by a general-purpose field computer? These questions can be answered in the context of approximation theory for operators on Hilbert spaces. For example, there is a sort of field-polynomial approximation based on an analogue of Taylor's theorem for functional derivatives [27, 28, 30, 33]. Also, since a field can be considered a continuum of (infinitesimal) neurons, many neural network approximation theorems can be adapted to field computation [20, pp. 166–168, 219–220, 236–239, 323–326]. For example, one universal set of operations is the field product (Hilbert–Schmidt integral), pointwise addition, and scalar multiplication [33, 44].

5.3.5 The U-Machine

We commonly classify computation as digital or analog, or as "hybrid" if it combines both, but does digital (computation on discrete spaces) and analog (computation on continua) exhaust the possibilities of computation? What other topologies might there be for information and computation?

We have explored computation on second-countable metric spaces because they include both discrete spaces and continua, and have developed a corresponding

machine model, the *U-machine* [38]. It gets its name from Urysohn's theorem, which states that any second-countable metric space is homeomorphic (topologically equivalent) to a subset of a Hilbert space [51, pp. 324–326]. Therefore, computations in second-countable spaces have realizations in Hilbert spaces, that is, they can be implemented by field computations. Indeed, the details of the Urysohn embedding imply that they can be approximated by computations over finite-dimensional vector spaces (and, in particular, neural networks).

Because the Urysohn embedding is a homeomorphism, any continuous computational process in a second-countable metric space has a continuous image in the subset of the Hilbert space. Further, for any continuous function on a second-countable space, there is a corresponding continuous function on the Hilbert space. Therefore, computations in second-countable spaces can implemented by computations in these Hilbert spaces, which can be implemented via the various universal approximation theorems on Hilbert spaces (Sect. 5.3.4). These provide the basic operations required for general-purposes computation on the U-machine.

Other sorts of physical media can be used to realize computational processes, for example, molecular computation. Next, however, I will address a different aspect of unconventional computation: how computation can be used directly to control physical processes.

5.4 Computation for Material Ends

In computation we have a relationship between a physical system and a formal system in which the formal system is a (typically incomplete) description of the physical system; this is the import of the realization homomorphism (Sect. 5.2.1). In conventional computation, as well as in the unconventional computation discussed in Sect. 5.3, the end (goal) is the abstract formal process, and the physical process is a means to that end. Furthermore, in a programmable computer, the program controls the physical processes so that they realize the abstract process described by the program. In particular, in the process of computation, matter and energy is reorganized in the computer, and this reorganization is under control of the program. Therefore we can look at the formal-material relationship from a different perspective in which the end is the physical process and the computation is the means to this end; that is, we have formal processes serving material purposes.

The tradeoffs are different. When the material processes are serving formal purposes, we usually try to minimize the energy and matter reorganized by computation, in order to decrease size, power requirements, and computation time. In contrast, when formal purposes are used to serve material ends, we might want to rearrange *more* matter or energy.

Computation for material purposes is different from a traditional control system, in which information processes (realized physically) govern a separate physical system via transducers. Here, we are describing a situation in which the physical realization of the computation *is* the physical process that is the goal. In particular, there are

no transducers because there is no distinction between the information system and the controlled system; they are simply the formal and material aspects of the same process.

A simple example is provided by chemical reaction-diffusion (RD) systems [2]. On the one hand, an RD system can be viewed as a formal process and analyzed mathematically, as Turing did [65]. On the other hand, RD systems can be realized chemically so that the chemical reaction and diffusion processes are essential to both the computation and the physical patterns it creates. Such processes underlie patterns in animal skin colors and hair coats [46]. Algorithmic assembly by DNA is another example in which the molecules realize a process that computes a desired physical structure [58, 59, 64].

In general, programs are hierarchical structures that, when executed, generate complex dynamics, which is capable of generating complex structures. That is, complex hierarchical temporal patterns can generate complex hierarchical spatial patterns. When we look at the physical realization of a computation, we realize that these intricate data structures are realized in correspondingly intricate arrangements of matter and energy.

5.4.1 Programmable Materials

The value of this inverted perspective on computations and their realizations is that it is an approach to *programmable matter*, that is, to controlling systematically the properties and behavior of physical systems on a small scale [19, 45].

A step in this direction is provided by what can be called *programmable materials*, that is, materials whose physical properties vary widely and can be controlled systematically (i.e., programmed) [45]. Some of the many properties we might like to control are hardness, elasticity, flexibility, density, relative resistance, permittivity, photoconductivity, opacity, and refractive index. Moreover, we would like a combinatorially rich code for determining these physical properties; by analogy with biology, we may call the code the *genotype*, and the physical substance the *phenotype*.

It might seem unlikely that such a versatile material could exist, but nature provides an example: proteins. Proteins are coded by the four nucleotide bases of DNA and so, effectively, by strings over the alphabet {A, C, T, G}, a simple, but combinatorially rich code. Nevertheless, proteins, which are the primary elements of living things, have an enormous range of physical properties and have both active and passive functions. Proteins are the constituents of keratin (the material of horns, nails, and feathers), connective tissue (collagen and elastin), cellular skeletons (microtubules), enzymes, ion channels, signaling molecules, receptor and sensor molecules (such as rhodopsin), transporter and motor proteins, and so forth. The DNA code defines long sequences of a few different building blocks (amino acids), but the resulting polymers fold into complex three-dimensional shapes that give them a wide variety of physical properties. Some allosteric protein molecules even make simple decisions, responding to various combinations of regulators [7, pp. 63–65, 78–79]. One

approach to programmable materials builds on proteins (natural and artificial), but once we understand the principle by which a simple, but combinatorially rich code can create structures with diverse physical properties, we can design new programmable materials based on different substrates.

5.4.1.1 Artificial Morphogenesis

Programmable materials may be very valuable, but much of the behavioral richness of living things comes from their complex hierarchical structure: from cells up to tissues, organs, and organisms, and from cells down to vesicles, membranes, and molecules (including proteins). There are many applications for which we would like to be able to build complex systems hierarchically structured from the microscale up to the macroscale. For example, we would like to be able to build robots with artificial nervous systems of comparable complexity and density to mammalian nervous systems, with similarly complex sensors and effectors to permit fluent, real-time behavior [42].

This raises the question of how to coordinate the self-assembly of vast numbers (millions or billions) of microscopic components into macroscopic complex systems. The problem might seem hopeless, but once again nature proves that it can be done. A human body has trillions of cells, yet during embryological development the cells self-organize into tissues, organs, and other structures. This suggests that embryological *morphogenesis*—the creation of physical form—can provide a model for the self-assembly of complex systems [6, 14, 15, 18, 23, 49, 50, 64]. Artificial systems may be very different from biological systems, but we can abstract the formal computation and control processes of morphogenesis from their biological realizations and apply them in artificial systems.

Our own approach to artificial morphogenesis is directed to the development of self-assembly processes that scale up to very large numbers of components (hundreds of thousands to millions or more) [34, 36, 37, 39, 41–43, 45]. To reach this goal, we describe morphogenetic processes by partial differential equations, effectively treating tissues as continuous media, and we use the mathematics of continuum mechanics. This is a reasonable approximation if the number of cells or agents is large, and is in fact commonly used in embryology and developmental biology. Using PDEs effects a useful separation of scales. The algorithms are developed and operate in terms of the dimensions of the object under assembly; this is the basis for determining parameters such as diffusion rates and agent velocities. These morphogenetic processes are independent of the scale of the "particles" (cells, agents, microrobots, etc.) constituting the medium, so long as it can be approximated as a continuum. Therefore, the algorithms do not depend on the size or number of agents; they scale.

To facilitate the expression of morphogenetic programs, we have developed a PDE-based programming language, which can be realized by computer simulation

or, in principle, by microscopic physical agents [34]. The notation is designed to be interpretable in discrete or continuous time in order to facilitate a variety of realizations in simulation and physical agents.

5.5 Embodied Computation

Artificial morphogenesis is an example of *embodied computation*, which may be defined as "computation in which the physical realization of the computation or the physical effects of the computation are essential to the computation" [41]. The term is inspired by the theories of *embodied cognition* and *embodied artificial intelligence*, which call attention to the role that the body plays in control and information processing in humans and other animals [9–11, 16, 21, 22, 47, 53–55]. Formal structures emerge from the possibilities of physical interaction between a body and its environment, and these physical processes can substitute for information processes, thus decreasing the computational load on the nervous system.

In embodied computation, the formal and material aspects are not so separable as they are in conventional computation. On the one hand, information processing and control may depend for its correctness and effectiveness on realization in a specific kind of physical system. However, the specifics of the physical systems also limit the purpose of the computation, that is, the final cause, since the computation is not required to operate in other situations. The specifics may also provide material realizations of the computation that are available for the specific computational systems, but not necessarily for others. That is, a specific embodiment restricts the final, material, and efficient causes (e.g., possible energy sources), but these same restrictions may afford a wider range of formal causes (i.e., information and control processes) to accomplish its purpose. To take an example from nature, the specific embodiment of *E. coli* and the properties of its environment facilitate its use of chemical gradients to control its metabolically-powered movement toward more favorable locations. Indeed, all living systems use embodied computation, and they suggest ways of designing artificial embodied computation systems.

5.6 Conclusions

Computation is physical, but conventional computing technology has been built on a hierarchy of abstractions. In the post-Moore's Law era, computational processes will need to be more like the physical processes by which they are realized, which implies a greater role for analog, parallel, and stochastic models of computation. The increasing assimilation of computation to physics raises the question: What distinguishes computational processes from other physical processes? The answer is that the purpose of the system could be accomplished as well by other physical realizations with the same formal structure but different material realizations (mul-

tiple realizability). Therefore, any formal process can be considered computation (information processing, control), and it is apparent that there is a wide variety of possible unconventional computing paradigms. The computational state space can be discrete or continuous, and information processing can proceed in continuous, discrete, or sequential time, either serially or concurrently. As we journey out from the familiar domain of conventional computation, we must leave behind familiar notions of programming and universality, whose assumptions may be misleading outside of their frame of relevance. Promising unconventional computing paradigms include analog computation, quantum computation, field computation, and computation over second-countable metric spaces (which subsumes both analog and digital computation).

Traditionally, the purpose of a computation is a certain formal process, and the accompanying physical processes are merely a means to that end. However, we may turn the tables, and use computation for the sake of these physical processes, using the formal power and flexibility of computation to control the assembly and behavior of physical objects. This approach provides a path towards programmable matter and artificial morphogenesis. More generally, embodied computation takes advantage of a closer assimilation of computation to physics by exploiting physical processes more directly for computation, and by using computational techniques to govern physical processes.

References

1. Adamatzky, A.: Physarum Machines: Computers from Slime Mould. World Scientific Series on Nonlinear Science Series A, vol. 74. World Scientific, Singapore (2010)
2. Adamatzky, A., De Lacy Costello, B., Asai, T.: Reaction-Diffusion Computers. Elsevier, Amsterdam (2005)
3. Ambs, P.: Optical computing: A 60-year adventure. Adv. Opt. Technol. 2010, Article ID 372,652 (2010). doi:10.1155/2010/372652
4. Bennett, C.H.: Notes on Landauer's principle, reversible computation, and Maxwell's demon. Stud. Hist. Philos. Mod. Phys. **34**, 501–510 (2003)
5. Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and Real Computation. Springer, Berlin (1998)
6. Bourgine, P., Lesne, A. (eds.): Morphogenesis: Origins of Patterns and Shapes. Springer, Berlin (2011)
7. Bray, D.: Wetware: A Computer in Every Living Cell. Yale University Press, New Haven (2009)
8. Brockett, R.: Dynamical systems that sort lists, diagonalize matrices and solve linear programming problems. In: Proceedings of the 27th IEEE Conference Decision and Control, pp. 799–803. Austin, TX (1988)
9. Brooks, R.: Intelligence without representation. Artif. Intell. **47**, 139–159 (1991)
10. Clark, A.: Being There: Putting Brain, Body, and World Together Again. MIT Press, Cambridge (1997)
11. Clark, A., Chalmers, D.J.: The extended mind. Analysis **58**(7), 10–23 (1998)
12. Connor, R.J., Holleman, J., MacLennan, B.J., Smith, J.M.: Simulation of analog solution of Boolean satisfiability. Technical Report UT-EECS-15-735, University of Tennessee, Department of Electrical Engineering and Computer Science, Knoxville (2015)

13. Das, A., Chakrabarti, B.K.: Colloquium : quantum annealing and analog quantum computation. Rev. Mod. Phys. **80**, 1061–1081 (2008). http://link.aps.org/doi/10.1103/RevModPhys.80.1061
14. Doursat, R.: Organically grown architectures: creating decentralized, autonomous systems by embryomorphic engineering. In: Würtz, R.P. (ed.) Organic Computing, pp. 167–200. Springer, Heidelberg (2008)
15. Doursat, R., Sayama, H., Michel, O. (eds.): Morphogenetic Engineering: Toward Programmable Complex Systems. Springer, Heidelberg (2012)
16. Dreyfus, H.L.: What Computers Still Can't Do. MIT Press, New York (1992)
17. Ercsey-Ravasz, M., Toroczkai, Z.: Optimization hardness as transient chaos in an analog approach to constraint satisfaction. Nature Phys. **7**, 966–970 (2011)
18. Giavitto, J., Spicher, A.: Computer morphogenesis. In: Bourgine, P., Lesne, A. (eds.) Morphogenesis: Origins of Patterns and Shapes, pp. 315–340. Springer, Berlin (2011)
19. Goldstein, S.C., Campbell, J.D., Mowry, T.C.: Programmable matter. Computer **38**(6), 99–101 (2005)
20. Haykin, S.: Neural Networks and Learning Machines, 3rd edn. Pearson Education, New York (2008)
21. Iida, F., Pfeifer, R., Steels, L., Kuniyoshi, Y.: Embodied Artificial Intelligence. Springer, Berlin (2004)
22. Johnson, M., Rohrer, T.: We are live creatures: Embodiment, American pragmatism, and the cognitive organism. In: Zlatev, J., Ziemke, T., Frank, R., Dirven, R. (eds.) Body, Language, and Mind, vol. 1, pp. 17–54. Mouton de Gruyter, Berlin (2007)
23. Kitano, H.: Morphogenesis for evolvable systems. In: Sanchez, E., Tomassini, M. (eds.) Towards Evolvable Hardware: The Evolutionary Engineering Approach, pp. 99–117. Springer, Berlin (1996)
24. Landauer, R.: The physical nature of information. Phys. Lett. A **217**, 188 (1996)
25. Lipshitz, L., Rubel, L.A.: A differentially algebraic replacment theorem. Proc. Am. Math. Soc. **99**(2), 367–372 (1987)
26. Lloyd, S., Braunstein, S.L.: Quantum computation over continuous variables. Phys. Rev. Lett. **82**, 1784–1787 (1999). http://link.aps.org/doi/10.1103/PhysRevLett.82.1784
27. MacLennan, B.J.: Technology-independent design of neurocomputers: the universal field computer. In: Caudill, M., Butler, C. (eds.) In: Proceedings of the IEEE First International Conference on Neural Networks, vol. 3, pp. 39–49. IEEE Press (1987)
28. MacLennan, B.J.: Field computation in the brain. In: Pribram, K. (ed.) Rethinking Neural Networks: Quantum Fields and Biological Data, pp. 199–232. Lawrence Erlbaum, Hillsdale (1993). http://web.eecs.utk.edu/~mclennan
29. MacLennan, B.J.: Continuous formal systems: A unifying model in language and cognition. In: Proceedings of the IEEE Workshop on Architectures for Semiotic Modeling and Situation Analysis in Large Complex Systems, pp. 161–172. Monterey, CA (1995). http://web.eecs.utk.edu/+mclennan and http://cogprints.org/541
30. MacLennan, B.J.: Field computation in natural and artificial intelligence. Inf. Sci. **119**, 73–89 (1999). http://web.eecs.utk.edu/~mclennan
31. MacLennan, B.J.: Natural computation and non-Turing models of computation. Theor. Comput. Sci. **317**, 115–145 (2004)
32. MacLennan, B.J.: Analog computation (chap. 1, entry 19). In: Meyers, R. et al. (ed.) Encyclopedia of Complexity and System Science, pp. 271—294. Springer, Heidelberg (2009). doi:10.1007/978-0-387-30440-3_19. Reprinted in *Computational Complexity: Theory, Techniques, and Applications*, ed. by Meyers, R.A. et al., Springer, 2012, pp. 161–184
33. MacLennan, B.J.: Field computation in natural and artificial intelligence (chap. 6, entry 199). In: Meyers, R. et al. (ed.) Encyclopedia of Complexity and System Science, pp. 3334–3360. Springer, Heidelberg (2009). doi:10.1007/978-0-387-30440-3_199
34. MacLennan, B.J.: Preliminary development of a formalism for embodied computation and morphogenesis. Technical Report UT-CS-09-644, Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN (2009)

35. MacLennan, B.J.: Super-Turing or non-Turing? Extending the concept of computation. Int. J. Unconv. Comput. **5**(3–4), 369–387 (2009)
36. MacLennan, B.J.: Models and mechanisms for artificial morphogenesis. In: Peper, F., Umeo, H., Matsui, N., Isokawa, T. (eds.) Natural Computing, Springer series, Proceedings in Information and Communications Technology (PICT) vol. 2, pp. 23–33. Springer, Tokyo (2010)
37. MacLennan, B.J.: Morphogenesis as a model for nano communication. Nano Commun. Netw. **1**(3), 199–208 (2010). doi:10.1016/j.nancom.2010.09.007
38. MacLennan, B.J.: The U-machine: a model of generalized computation. Int. J. Unconv. Comput. **6**(3–4), 265–283 (2010)
39. MacLennan, B.J.: Artificial morphogenesis as an example of embodied computation. Int. J. Unconv. Comput. **7**(1–2), 3–23 (2011)
40. MacLennan, B.J.: Bodies – both informed and transformed: Embodied computation and information processing. In: Dodig-Crnkovic, G., Burgin, M. (eds.) Information and Computation. World Scientific Series in Information Studies, vol. 2, pp. 225–253. World Scientific, Singapore (2011)
41. MacLennan, B.J.: Embodied computation: applying the physics of computation to artificial morphogenesis. Parallel Process. Lett. **22**(3) (2012)
42. MacLennan, B.J.: Molecular coordination of hierarchical self-assembly. Nano Commun. Netw. **3**(2), 116–128 (2012)
43. MacLennan, B.J.: Coordinating massive robot swarms. Int. J. Robot. Appl. Technol. **2**(2), 1–19 (2014). doi:10.4018/IJRAT.2014070101
44. MacLennan, B.J.: The promise of analog computation. Int. J. Gen.Syst. **43**(7), 682–696 (2014). doi:10.1080/03081079.2014.920997
45. MacLennan, B.J.: The morphogenetic path to programmable matter. Proc. IEEE **103**(7), 1226–1232 (2015)
46. Meinhardt, H.: Models of Biological Pattern Formation. Academic Press, London (1982)
47. Menary, R. (ed.): The Extended Mind. MIT Press, Cambridge (2010)
48. Molnár, B., Ercsey-Ravasz, M.: Asymmetric continuous-time neural networks without local traps for solving constraint satisfaction problems. PLoS ONE **8**(9), e73,400 (2013). doi:10.1371/journal.pone.0073400
49. Murata, S., Kurokawa, H.: Self-reconfigurable robots: shape-changing cellular robots can exceed conventional robot flexibility. IEEE Robot. Autom. Mag. pp. 71–78 (2007)
50. Nagpal, R., Kondacs, A., Chang, C.: Programming methodology for biologically-inspired self-assembling systems. In: AAAI Spring Symposium on Computational Synthesis: From Basic Building Blocks to High Level Functionality (2003). http://www.eecs.harvard.edu/ssr/papers/aaaiSS03-nagpal.pdf
51. Nemytskii, V.V., Stepanov, V.V.: Qualitative Differential Equations, Reprint of 1960 Princeton Univ, Press edn. Dover, New York, NY (1989)
52. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information, 10th anniversary edn. Cambridge University Press, Cambridge (2010)
53. Pfeifer, R., Bongard, J.: How the Body Shapes the Way We Think – A New View of Intelligence. MIT Press, Cambridge (2007)
54. Pfeifer, R., Lungarella, M., Iida, F.: Self-organization, embodiment, and biologically inspired robotics. Science **318**, 1088–93 (2007)
55. Pfeifer, R., Scheier, C.: Understanding Intelligence. MIT Press, Cambridge (1999)
56. Popa, C.R.: Synthesis of Computational Structures for Analog Signal Processing. Springer, New York (2011)
57. Pour-El, M.: Abstract computability and its relation to the general purpose analog computer (some connections between logic, differential equations and analog computers). Trans. Am. Math. Soc. **199**, 1–29 (1974)
58. Rothemund, P., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biol. **2**(12), 2041–2053 (2004)
59. Rothemund, P., Winfree, E.: The program-size complexity of self-assembled squares. In: Symposium on Theory of Computing (STOC), pp. 459–68. Association for Computing Machinery, New York (2000)

60. Rupp, K., Selberherr, S.: The economic limit to Moore's law. IEEE Trans. Semicond. Manuf. **24**(1), 1–4 (2011). doi:10.1109/TSM.2010.2089811
61. Santoro, G.E., Tosatti, E.: Optimization using quantum mechanics: quantum annealing through adiabatic evolution. J. Phys. A: Math. Gen. **39**(36), R393 (2006). http://stacks.iop.org/0305-4470/39/i=36/a=R01
62. Shannon, C.E.: Mathematical theory of the differential analyzer. J. Math. Phys. Mass. Institute Technol. **20**, 337–354 (1941)
63. Shannon, C.E.: Mathematical theory of the differential analyzer. In: Sloane, N.J.A., Wyner, A.D. (eds.) Claude Elwood Shannon: Collected Papers, pp. 496–513. IEEE Press, New York (1993)
64. Spicher, A., Michel, O., Giavitto, J.: Algorithmic self-assembly by accretion and by carving in MGS. In: Proceedings of the 7th International Conference on Artificial Evolution (EA '05), no. 3871 in Lecture Notes in Computer Science, pp. 189–200. Springer, Berlin (2005)
65. Turing, A.: The chemical basis of morphogenesis. Philos. Trans. R. Soc. B **237**, 37–72 (1952)
66. van Gelder, T.: Dynamics and cognition (chap. 16). In: Haugeland, J. (ed.) Mind Design II: Philosophy, Psychology and Artificial Intelligence, revised & enlarged edn., pp. 421–450. MIT Press, Cambridge (1997)