https://doi.org/10.1142/9789811235726_0001

Chapter 1

Mapping the Territory of Computation Including Embodied Computation

Bruce J. MacLennan

Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, Tennessee 37996, USA
maclennan@utk.edu

Investigation of alternatives to conventional computation is important both in order to have a comprehensive science of computing and to develop future computing technologies. To this end, we consider the full range of computational paradigms that is revealed when we relax the familiar assumptions of conventional computation. We address the topology of information representation, the topology of information processing, and unconventional notions of programmability and universality. The physics of computation is especially relevant in the post-Moore's law era, and so we focus on *embodied computation*, an alternative computing paradigm that focuses on the physical realization of computation, either making more direct use of physical phenomena to solve computational problems, or more directly exploiting the physical correlates of computation to implement some intended physical process. Examples of the exploitation of physical processes for information processing include analog computation, quantum computation, and field computation. Examples of the use of embodied computation for physical purposes include programmable matter and artificial morphogenesis.

1.1. Unconventional Computation

Unconventional computation, non-standard computation, alternative computation: I take these to be synonyms, but what do they mean? They are all negative terms, defined more by what they *are not* than by what they *are*; they refer to computation that is not conventional or standard, or that is an alternative to what is conventional and

standard. Therefore, a computing paradigm may be "alternative" by deviating from the common characteristics of computing as we have come to know them. Among these are binary digital electronics, sequential electronic logic, von Neumann architecture, discrete data representation, discrete memory units randomly addressable by natural numbers, modifiable memory, programs stored in this memory, sequential execution from memory, deterministic processing, conditional and iterative control mechanisms, and hierarchical program organization. You can no doubt think of more, and some otherwise quite conventional computing systems might lack one or another of these characteristics, but they indicate possible directions for alternative computing paradigms.

The conventional computing paradigm has been wildly successful, and it is reasonable to question the value of investigating alternatives, but there are at least two motivations. The first motivation, and I think the most important, is scientific. Computer science is the science of computing and so it should investigate computing in all its manifestations, artificial and natural. Restricting attention to conventional computation would be akin to biologists restricting their study to bacteria (because they are so numerous) while ignoring all other living things.

Computation may be defined as a physical process with the function or purpose of processing information (see Refs. [1, 2] for more on distinguishing computing from other physical processes). Throughout history, many computational techniques have been developed, including manual arithmetic, slide rules, abacuses, and similar devices, but also geometric constructions and tools (e.g., pantographs). Also included are formal logical techniques, including logical calculi and various sorts of logic diagrams. Over the centuries machines have been designed to do more-or-less automatic computation, including (electro-)mechanical arithmetic calculators, mechanical analog integrators and Fourier analyzers,[3,4] and (electro-)mechanical analog differential analyzers.[5,6] In the realm of electronic computation, we have the modern digital computer, but also analog computers, which were once as common as digital machines and are returning to importance for some applications,

such as neural networks.[7] It behoves us to study all the ways computing has been done in the past, both to understand it more completely, but also to better understand the possibilities for future, alternative computing technologies.

Computer science also investigates the manifestations of computing and information processing in nature, not only in the brains of individual animals, but also in the "group brains" of social animals (including human beings). Computational principles are involved in the cooperation and coordination of flocks of birds, schools of fish, and herds of land animals. Social insects (and much simpler organisms, such as slime molds) solve optimization problems and construct complex nests and colonies simultaneously fulfilling multiple functions. Evolution by natural selection is itself information processing, searching a complex space for designs with selective advantage. The discipline of *natural computation* studies these instances of computation in nature, but also takes inspiration from them to develop future computing methods.

It is apparent that computation in nature has little in common with conventional computation: it is rarely binary or even digital, but more often analog; it is rarely strictly sequential and more commonly asynchronous and parallel; typically it does not separate memory and processing or make use of stored programs; it is often nondeterministic; it operates reliably and robustly under conditions of significant and unavoidable uncertainty, errors, defects, faults, and noise; and so on.[1] Nevertheless, natural computation is very effective; it has facilitated the evolution of innumerable species, including our own. This demonstrates that computing is much more than our familiar digital von Neumann devices. Therefore, in Section 1.2, we will consider computation in the broad sense.

Another important motivation for investigating alternative computing is the inevitable end of Moore's Law. Although this empirical law results from a complex interaction of technological and economic factors, it is apparent that it cannot continue forever. Due to the atomic structure of matter and the laws of physics, there is a limit to the smallness, density, and speed of electronic components. The semiconductor industry continues to squeeze out incremental

improvements, but they are coming at an increasing cost and the end is in sight.

A more distant, but harder barrier is posed by the von Neumann-Landauer bound, which arises from the fact that information must be represented physically.[8] As a consequence, erasing, destroying, or losing a bit of information must dissipate a certain minimum amount of energy, specifically, $k_B T \ln 2$, where k_B is Boltzmann's constant and T is ambient temperature; this minimum energy is about 18 meV at 300 K. The von Neumann–Landauer limit, which was originally established theoretically, has been recently confirmed empirically.[9] Therefore, any computational operations that forget information (in either computation or control) must dissipate at least this much energy, which usually appears as heat. The only way to avoid this limitation is to avoid discarding information during computation, which is accomplished by *reversible computing*, an alternative computing paradigm that has been explored both theoretically[10–13] and practically.[14]

1.1.1. *Implications*

Conventional computing technology is built upon many cleanly separated levels of abstraction. Primitive data abstractions, such as integers, characters, and real numbers, are implemented in terms of bits, which are realized by physical devices with continuous state spaces but operated in saturation, so as to simulate binary states. Primitive data processing operations, such as addition, division, and comparison, are realized by sequential digital logic, which is realized by devices obeying continuous physical laws, but switching quickly between saturated states. This clean separation of abstraction levels has facilitated independent progress on each level. For example, the same basic sequential digital logic has survived while switching technology has progressed from mechanical, to electromechanical (relays), to vacuum tubes (valves), to discrete transistors, to integrated circuits, to very large scale integration. Algorithms (e.g., Newton's algorithm, sorting algorithms) that ran on the earliest computers can be and are run on the latest computers. Therefore,

progress on these various levels has not required us to abandon the accumulated technological knowledge on other levels, which has enabled rapid progress in computer science.

Unfortunately, we are reaching the limits of this hierarchical computing technology, with its many levels between our programming abstractions (embodied, e.g., in high-level programming languages) and the physical laws governing our basic computing devices. To achieve greater component densities and speeds, the number of layers of abstraction must be decreased, since each layer introduces an "abstraction cost". The only way to accomplish this, since the laws of physics are invariable, is to bring our programing abstractions closer to the underlying physics.[15] That is, our programming abstractions should be more like the physical processes that realize them.

The laws of physics are continuous — differential and partial differential equations — and therefore one implication of this increasing assimilation of computing to physics is a greater dependance on analog models of computation, that is, computation that is continuous in state space and perhaps also dynamics. Analog computing avoids the inefficiency of implementing continuous computation in terms of digital computation that is in turn realized by continuous physical processes. This conclusion applies as well to quantum computing, which is founded on a continuous wave function, continuous (complex valued) superpositions of basis states, and continuous dynamics (Schrödinger's equation).

The clean separation of levels of abstraction has depended on the accuracy of simulation at each level; for example, our digital switches really behave like perfect switches. This has been facilitated by the largeness of our devices (compared to atomic dimensions), by redundancy in our digital state representations, and by the relative slowness of our digital processes compared to the underlying physics (e.g., allowing processes to reach saturation in much less than a clock cycle). Our technology has been built on a stack of idealized abstractions, in which the idealizations have been close enough to reality to be reliable. As we push toward smaller devices and higher densities, however, and toward higher speeds, these idealizations break down.

In particular, noise, inaccuracy, imperfections, faults, uncertainty, and other deviations from our idealized models will be unavoidable. Instead of striving to completely eliminate them (which would increase cost and inefficiency), we should embrace these inevitable physical phenomena as sources of *free variability*, which can be exploited for computational purposes (e.g., escape from deadlock, exploration, non-determinism). *Natural computation*, that is, computational processes in nature, teaches us ways to use this free variability, since noise and error are unavoidable in nature, and living systems, which are imperfect, have evolved to survive under these circumstances.

1.2. Computation in General

Since we will need to "think outside of the Boolean box" to develop future alternative models of computation, it will be worthwhile to explore the landscape and boundaries of the idea of computation. We will consider first the range of possibilities for information representation, and second the variety of dynamical processes by which information might be processed.

1.2.1. *Topology of information representation*

Rolf Landauer coined the slogan "Information is physical" to remind us that information must be embodied in physical reality (at least if it is to be used in any way), and therefore that information is not independent of physical properties and limitations.[8] (I have already mentioned the von Neumann–Landauer limit as an example.)

In order to explore the physical nature of information, it is convenient to use the Aristotelian distinction of *form* and *matter* (*hylomorphism*).[15] For our purposes, the *form* (Grk. *morphē*, *eidos*; Lat. *forma*) is some discernable or discriminable arrangement or structure of an underlying medium or substrate, the *matter* (Grk. *hulē*; Lat. *materia*), which might be physical matter or energy. Form and matter are relative terms, in which form refers to physical properties that are intended or used to represent the information, and matter refers to those physical properties that are not. Although

information must be instantiated in some physical substrate, it is, *qua* information, independent of the matter and exists only in the form. This independence leads to the misleading idea of disembodied or non-physical information, to which Landauer objected. On the other hand, information is *multiply realizable* in that the same or an equivalent form can be realized in various material substrates, so long as they support the fundamental differences of form required to represent the information. (Thus, information is potentially separable from matter, but in actuality always materially realized.)

Most material substates possess a large number of physical properties, many degrees of freedom, only some of which are used to represent information. Therefore it is useful to distinguish *information-bearing degrees of freedom* (*IBDF*) from *non-information-bearing degrees of freedom* (*NIBDF*).[13] The distinction is grounded in the information processing, in the computational processes to which the information is subject. Which are relevant to the computation? Which irrelevant? For an example, we may consider a simple electrical implementation of a bit: if all the free electrons are on one plate of a capacitor, it represents a 0; if they are on the other plate, a 1. This is the IBDF; the positions and velocities of the electrons within the plate are NIBDF, for they are irrelevant to whether a 0 or 1 is represented.

The relevant distinctions of form that constitute an information space can be expressed often in terms of the topology of the space. For example, a conventional digital computer operates on the discrete topology on $\{0, 1\}$ (or any homeomorphic space); an analog computer might operate on continuous variables with states in the continuum $[0, 1]$ (or spaces homeomorphic to it). Another might operate on a bounded region of the complex plane. These all are supported by a variety of physical realizations.

More generally, the topology of an information space defines relationships of similarity and difference, of approximation and distance. It is the background on which information processing and control takes place. Second-countable metric spaces (i.e., metric spaces with a countable base) are a useful framework for many models of computation, for they include both discrete spaces and continua.[16]

Information often has a complex constituent structure, which is exemplified by the data structures used in conventional computer programming. In general, the topology of these composite information spaces is in some relevant sense the product of the topologies of its constituent spaces. In conventional digital computation, the components of a data structure are discrete, for example, the homogeneous elements of an array or the heterogeneous elements of a structure (record); memory in a von Neumann machine is an indexable array of discrete bytes or words. Many analog computers similarly operate on a discrete set of continuous variables or on vectors or matrices of continuous variables, and so they have a discrete constituent structure. However, analog computers have also made use of continuous distributions of continuous information, that is, of continuous *fields* of data. Early analog computers used a *field analogy method* for solving partial differential equation problems [6, p. 34], and fields provide a model for optical and quantum computing technologies.[17–20] For these information spaces, Hilbert spaces often provide the relevant topology.[21] In general, information spaces may be function spaces on discrete or continuous domains.

1.2.2. *Topology of information processing*

Information is physical, as Landauer said, and therefore information processing is also physical, which gets at the heart of computing. Computation is a physical process, but what distinguishes it as computation from other physical processes is that (in hylomorphic terms) it depends only on the form of its state and not on its matter to fulfill its purpose[1,2]; we may call these *information processes*. Therefore, an information process (computation) is multiply realizable, since it can be realized by any physical system that supports the formal distinctions and transformations used in the computation.

In general, a computation may be considered a dynamical system coupled with its environment. The state space of the composite system is a product of the external state space (at least as manifest in the interface between the coupled systems) and the computational state space; either may be discrete or continuous. If the state space is discrete, then the dynamics is necessarily discrete as well. Often in a

discrete-time dynamical system, the state changes as fixed intervals, perhaps controlled by a clock, but another possibility is *sequential dynamics*, in which the sequence of states is defined, but not the state transition times.[22] Conventional digital computer programs exhibit sequential dynamics, for the sequence of operations is defined, but there is no presumption that they take equal amounts of time. If, on the other hand, the state space is a continuum, then the dynamics can be either continuous or discrete (including sequential). In all these cases, the information relationships within the computational system constrain the dynamics of the composite system to fulfill some function or purpose.

Within this broad framework, there are many alternatives. Are states strictly ordered or only partially ordered? Are state changes deterministic or probabilistic? Are they reversible (as in Brownian and quantum computing)? In states with either a finite discrete set of components or with a continuum of components (such as a field), how are the dynamics of the components related? Sequential? Synchronous parallel? Asynchronous parallel? Stochastic? The dynamics of information processing in natural systems, non-living as well as living, suggest many possibilities.

1.2.3. *Programmability*

Conventional computation is generally associated with the idea of programmability, but computers in general do not need to be programmable; we can have special purpose computers designed for a single computation. Nevertheless, programmability is important, for it allows a single computer to be easily reconfigured for different computational purposes, but we must consider programmability in a more general sense. There is a tendency now to define programmability in relation to the universal Turing machine or its equivalents (lambda calculus, etc.), but as will be discussed later (Section 1.2.4), alternative models of computation require alternative definitions of computability.

If we think about the variety of ways that we use the verb "program", it is apparent that it refers to a process by which the behavior of some system can be controlled by some abstract means. That is, a

more general class of possible behavior is restricted to some desired subclass by means of an abstract specification, the program. To put it in hylomorphic terms, the programmable system is the *matter*, which has a broad class of possible behaviors, from which a subclass is selected by imposition of a *form*, which is the program. The form (program) organizes the computational substrate or medium to have the desired dynamical behavior. The form becomes operative (is executed) by its embodiment in the computational medium. It is *actualized* (becomes active) by embodiment in an appropriate computational medium, otherwise it is only a *potential* program so long as its form is embodied in a non-computational medium (e.g., a piece of paper or a text file).

We are most familiar with textual and diagrammatic representations of programs, that is, programs expressed in programming languages or flow diagrams. Both have a discrete constituent structure, which represents the dynamics in terms of basic computational processes. The space of possible programs for a computer is generally a formal language (over a finite, discrete alphabet) defined by a formal grammar. Of course, programs are often translated from one form to another, for example, from a program for an abstract computer to an equivalent program for a physical computer.

Programs need not belong to discrete spaces, they can belong to continua. For example, the dynamics of a computation can be governed by a Hamiltonian function or a potential surface. In a simple case, the input is encoded in the initial state, and the output is encoded in a fixed-point attractor to which the system converges, governed by the potential function.

Such continuous programs may be described in a discrete language; for example, a potential function might be defined by a finite set of equations, describing a problem Hamiltonian, as is done in quantum annealing and similar optimization techniques. In this case, the discrete program implicitly defines a continuous program. In other cases, a continuous program might be expressed directly; the appropriate metaphor might not be *writing* a program, but rather *sculpting* a program.[1] More commonly, continuous programs will not

be created explicitly by some programmer, but will emerge from machine learning.

1.2.4. *Universality*

Any discussion of alternative, unconventional computing paradigms must address the issue of Church–Turing computability, which has been the de facto definition of computability for the better part of a century. It is so familiar that it is difficult to imagine other definitions, and the assumptions on which it is built are largely forgotten. We must remember that Church, Turing, and the others who created the theory of computation were trying to formalize the notion of effective calculability in the foundations of mathematics. The assumptions built into the model (discrete symbols, exact matching, sequential dynamics, finiteness of representation and processing, etc.) were appropriate for the problems they were addressing, and it is quite remarkable that it has been applicable to conventional computing more generally.

Nevertheless we must recall that the Turing machine is a *model* of computation, and like all models it is useful because it makes simplifying assumptions that are unproblematic for the domain of questions it is intended to address. We may call this domain of questions and issues the model's *frame of relevance*.[1,23] Models generally give incorrect or misleading answers when applied outside of their frame of relevance or near to its boundaries, where its simplifying assumptions affect the answers. Near or beyond the boundaries, we are in danger of obtaining answers that have more to do with the model and its assumptions than with the system being modeled. Questions of computing *power* (e.g., whether a computing paradigm has the power of a universal Turing machine), depending on how they are framed (e.g., in terms of the class of functions that can be computed), might or might not be in a model's frame of relevance. For example, the Church–Turing model is generally ill-equipped to answer questions about real-time performance and real-time emulation, which are relevant to some notions of computing power.

A more pragmatic model of computational universality, which is sometimes used, may be termed *logic-gate computational universality*. It is based on the observation that all our conventional, von Neumann computers are constructed from a few logic gates, and therefore any computing paradigm that can implement arbitrary finite circuits of these gates can, in principle, do anything a conventional digital computer can do. (This is, of course, only an approximation to Church–Turing universality, which also requires an unbounded memory; conventional digital computers are finite-state machines.) This model is suitable for addressing questions of binary computation, but is less useful for questions relevant to unconventional computation (e.g., analog computing), which are outside of its frame of relevance.

In natural computation and many other alternative computing paradigms (see, e.g., Section 1.3), other notions of universality may be more relevant than the universal Turing machine.[1] In general, we are asking what class of dynamical systems can be implemented in a particular computational medium (*matter*) by a specified space of programs (*forms*). Since this dynamical system might be intended to interact with the external, non-computational environment, issues of real-time performance, accuracy, physical resources, and energy dissipation might be relevant. In other words, the question of whether a programmed system is "equivalent" to some hardware might not be a simple matter of computing the same class of functions; it might be more than this in some respects, and less in others. We must beware of being seduced by our familiar models and theories.

1.3. Embodied Computation

1.3.1. *Definition*

If future progress in computation, especially post-Moore's law computation, will require a greater assimilation of computation to physics, then we need to look deeper into the relation between computations and their physical realizations. As discussed above, computation has been viewed traditionally as an abstract process largely independent of its material embodiment. This parallels a

Cartesian approach to cognition, which treats it as information processing independent of the brain and of the body more generally. Cartesianism has been found inadequate in cognitive science, and *embodied cognition* approaches cognition from the perspective of information processes realized in a biological brain and with a principal function of controlling a physical body in its physical environment.[24] When cognition is approached from this perspective, many problems become easier to solve. Neurological processes in the physical brain simplify some information processing tasks, as does the fact that it is controlling a body with specific physical properties.

Embodied computation applies these insights to information processing more generally by thematizing and exploiting the relation between computation and its physical realization. On one hand, it makes more direct use of physical processes to implement information processes, thereby achieving a closer assimilation of computation to physics. On the other, it provides a framework for understanding and designing systems in which the goal is not information processing *per se*, but in which information processes are essential to some intended physical process. With these considerations in mind, we have proposed the following definition of embodied computation:

> *Embodied computation* 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.[25]

(Reference [26] uses "embodied computation" in a different, but related sense. Other related ideas are *material computation* and *in materio computation*.[27]) We consider first the use of physical processes for computational purposes, which is more familiar and easier to understand, and then we turn to the use of computational processes for physical purposes.

1.3.2. *Physics for computational purposes*

Embodied computation can exploit physical processes for more direct and effective realization of a computational process when the physical process has a mathematical structure that is the same or closely related to the computational process. This is of course

analog computation in the original sense, in which a target system is simulated by a more convenient analogous system with the same mathematical structure. This is especially advantageous when the physical system has a very large number of information-bearing elements (e.g., atoms) or when the information is represented by a continuous field, for in these cases the simultaneous interaction of the spatial elements can directly implement what would be a very expensive computation on a conventional computer (e.g., solving a system of PDEs).

Since all computation must be physically realized, the reader may wonder how embodied computation differs from conventional computation. It is in fact a matter of degree. Conventional computers provide a *generic physical realization* which is universally adequate for a broad class of computations (roughly, Turing-computable functions). These are the familiar electronic bits and binary operations discussed previously (Section 1.1). Since embodied computation depends on more direct physical realizations of computational processes, or because it is in direct physical interaction with its environment, the possible physical realizations may be more limited. For example, there may be few *specific realizations* that have an appropriate mathematical structure for a computation (e.g., obey appropriate PDEs) and that also operate at a rate suitable for environmental interactions. In other words, in embodied computation we cannot ignore physical realization to the same degree that we have in conventional computation, and different specific realizations might be more or less suitable for different embodied computations.

A hallmark of conventional computation is *multiple realizability*, which means that, in principle, any computation can be realized on any computer that has the power of a universal Turing machine, and this independence of specific physical realization depends on the many levels of abstraction between computations and the physical processes implementing them. Multiple realizability is also important in embodied computation, and so we look for computational abstractions that can be realized — more directly than in conventional computation — by a variety of physical processes, thus expanding the range of usable specific realizations.

Therefore, one challenge in embodied computation is to identify or design physical processes that have the same mathematical structure as useful computations, while also having desirable physical characteristics (e.g., speed and controllability). It is not essential that the physical process have the same mathematical structure as the computation, so long as it can be easily "programmed" (in the sense of Section 1.2.3) to simulate the desired computation without many levels of abstraction.

1.3.2.1. *Transduction*

Input and output *transduction*, which is a transformation between the generic computational realization and a specific input or output medium, is somewhat different in embodied computation than in conventional computation.[1] This is easiest to understand in the context of conventional *embedded* computers, which will have a number of sensors and actuators that allow the computer to receive information from the physical system in which it is embedded and to have physical effects on that system. Sensors are specific to the form of matter or energy that they sense, and thus they transduce information with a specific physical realization into information internal to the computer, which is generically realized, that is, in principle realizable by any other appropriate physical process with the same mathematical structure. In contrast, the specific physical realization of the input cannot be changed, or the sensor will be sensing the wrong information. For example, an electronic analog computer might represent quantities by voltages; that is the generic computational realization. It is generic because it could, in principle, be changed to another realization (e.g., fluid pressure) and still accomplish the computer's function. On the other hand, a light sensor has to detect light intensity; it cannot be changed to something else, such as temperature, without changing the function of the computer. The situation is similar for actuators: generically realized information in the computer is transduced into a specific physical form so that it can have the intended effect in the embedding physical system and environment. So the computational medium (e.g., voltage or fluid

pressure) is converted to a specific medium (e.g., mechanical force or light intensity) as required for its purpose.

In an ideal transduction process, only the *matter* of the information is changed, not its *form*, as it is transferred from a specific realization to the computational realization or vice versa. In practice, there is also some change of form (for example, a continuous signal might be digitized or limited in range). We can think of ideal input transduction as the process of removing the units from a physical quantity and turning it into a pure number, and output transduction as the inverse process of turning a pure number into a physical quantity by applying appropriate physical units.

In conventional computation, transduction has a bow tie organization, with a single computational realization being the target of multiple input transductions and the source of multiple output transductions. The issue of transduction is more complex in *embodied* computation since we might not have the benefit of a single computational representation as either the target or the source of transductions. Rather, we will need to transduce physical input information into the specific physical realization that will be used for the computations to be applied to it. Conversely, output signals will need to be transduced from the physical realization of the computation that produced them to the appropriate physical realization of the output signals.

1.3.2.2. *Analog computation*

Analog computation originally referred to computation in which some convenient physical system was used as a model to simulate some target system of interest. However, since most analog computers (whether mechanical, electrical, or some other medium) used continuous physical processes, and because they were most often used to model physical systems obeying continuous laws (expressed as systems of ordinary or partial differential equations), the term "analog computation" soon came to refer to computation in continuous media, as opposed to "digital" computation, implemented by discrete (generally binary) processes. They are more properly called *continuous computation* and *discrete computation*, respectively.

Mathematically, analog computers are continuous-time dynamical systems, typically defined by a system of ordinary or partial differential equations (ODEs or PDEs). When applied to more conventional computational problems, the inputs may be encoded in the initial state of the system, and the corresponding outputs are encoded in the final states to which they converge (attractors). All analog computation is physically realized, but we call it *embodied* to the extent that there is an ongoing interaction between the computation and its physical environment, in which inputs may define boundary conditions for the computation or define a subset of extrinsic variables, and a subspace of the internal state space determines the outputs. Analog computing, therefore, applies physical processes to computation by identifying or implementing a dynamical system that is able, relatively directly, to solve the computational problem. There are many examples of such dynamical system solutions, even applied to discrete combinatorial optimization problems, such as Boolean satisfiability.[28,29] The challenge for embodied analog computation is to find a physical dynamical system that can be easily configured to solve the problem.

The problem of finding or designing a physical system to realize a desired dynamical system points to the need for *general-purpose analog computers* (*GPACs*), as was also apparent in an earlier generation of analog computation.[6] Ideally, we would like a model of universal analog computation comparable to the universal Turing machine (UTM) in conventional computation, but it does not yet exist. Embodied computation is outside of the frame of relevance of the Church–Turing model of computation, and so the UTM is not a very useful basis for a universal analog computer (see Section 1.2.4).[1]

One promising approach to universal analog computation and GPACs is provided by various universal approximation theorems in mathematics, which typically establish how compositions of a restricted set of *basis functions* can be used to approximate as closely as required a given function in a large and interesting class.[17,18,30] Polynomial and Fourier series approximations are familiar examples, but not necessarily the most useful for GPACs. Already in the 1940s Claude Shannon proved theorems establishing the computational

capabilities of GPACs inspired by the (mechanical) differential analyzer,[31,32] which were later corrected.[33–36] More useful perhaps are universal approximation theorems that show how GPACs can be designed around sigmoidal neural networks and radial basis function networks [30, pp. 166–168, 219–220, 236–239, 323–326].

1.3.2.3. *Quantum computation*

Quantum computation is a paradigmatic example of embodied computation, for it makes direct use of the phenomena of quantum mechanics to perform computations that would be prohibitively expensive to solve on conventional computers. This is because quantum computers have the potential to perform an exponential number of conventional computations in quantum superposition. In addition to the digital computer-inspired "circuit" or "logic-gate" model of quantum computation,[37] there are also models that treat the complex amplitudes of quantum states as continuous variables, thereby providing a kind of analog quantum computation.[38] Hilbert spaces provide the mathematical framework for quantum computation.

1.3.2.4. *Field computation*

Conventional computers have a discrete address space comprising discrete variables (bits, bytes, words) in a regular array, typically indexed by natural numbers. As discussed in Section 1.2.1, some alternative models of computation provide *fields*, that is, continua of continuous quantity. In principle, individual values are indexed by real numbers, but in practice *field computation* operates on entire fields in parallel.[17,18] Mathematically, fields are treated as functions belonging to Hilbert spaces, field operations are (possibly nonlinear) Hilbert space operators, and dynamical systems are defined by PDEs.[21] Embodied field computation makes use of physical processes operating on physical fields and defined by PDEs. Computational fields may be realized by physical fields that are literally continuous (such as electromagnetic fields) or by *phenomenological fields*, such

as fluids, which are composed of discrete elements, but of sufficient number and density to be treated as continua.

Many physical processes, including electrical, optical, and chemical processes, are described by PDEs operating on fields, and they are potential realizations of field computation. For example, physical diffusion can be used for broadcasting information, optimization, and parallel search.[39–44] Large populations of microorganisms (e.g., bacteria, slime molds) can be used to solve some problems (rather slowly).[45] Indeed, quantum computation is a kind of field computation implemented by linear operations on the wave function. Functional analysis provides a series of universal approximation theorems that are a basis for programmable general purpose field computers.[18] These include approximation by Taylor series over Banach spaces, generalized Fourier series, and methods analogous to convolutional neural networks.[18,21]

1.3.3. *Computation for physical purposes*

We have seen how the idea of embodied computation suggests ways that the physical realization of a computation can be exploited to better fulfill its computational purposes. Here we will shift the focus to see how computation can be used to better fulfill the purposes of some physical system. What makes a physical system an example of embodied computation is that it uses computational concepts and techniques in an essential way in order to achieve desired physical behavior and effects. Computation is a physical process, but in these cases it is the physical process that fulfills the purpose of the system, and the computation is a means to that end. In hylomorphic terms, the computation's *formal* relations evolving in time impose a desired physical process on a relatively unstructured material substrate.

In other words, when a computation takes place, matter and energy are transported and transformed within a physical system, such as a computer. In embodied computation we make use of this fact to achieve some desired series of physical states by means of the computations that are realized by them.

These applications of *embodied* computation may sound like *embedded* computation, in which a computational system is a part of a physical system, which it helps to control, but in embedded computation the computer has its own physical realization (e.g., in electronic circuits) separate from the physical system it is controlling. In embodied computation, the physical realization of the computation *is* the system being controlled. There is no distinction between controller and controlled.

Reaction-diffusion systems provide a simple example of embodied computation for physical effect.[46] They are fundamentally mathematical systems: a system of 2D PDEs combining diffusion with nonlinear reaction terms. Under a variety of conditions, studied by Turing, they evolve into stable arrangements of spots and stripes now called Turing patterns.[47] As computational systems they can be realized in a variety of media, but when they are realized by morphogens in the skin of a developing embryo, they lead to the characteristic hair color and skin pigmentation patterns of various species (the leopard's spots, the tiger's stripes, etc.).[48–50] Therefore, if for some application we want to arrange matter or energy in a Turing pattern, we can do this by using the required materials to realize an appropriate reaction-diffusion system.

More generally, developing embryos provide many examples of embodied computation for physical effect. The proliferating cells in an embryo communicate and coordinate with each other to control their movement, differentiation, and adhesion to create an animal's complex physical body.[51] The communication and coordination of the cells is an information process because it could, in principle, be realized by different physical substances and still fulfill its function. Other examples of naturally occurring embodied computation for physical effect include the communication and control within social insect colonies by which they organize their group behavior and construct their nests.[50]

Embodied computation for physical applications has different tradeoffs and requirements than conventional computation (including embedded computation). For example, conventional computation and embodied computation applied to information processing both

exhibit some degree of multiple realizability: the physical realization does not matter too much so long as it supports the required computations. (We don't care what matter and energy is moved around so long as the pattern of its movement realizes the computation.) With embodied computation, the computation must be realized by physical processes that are appropriate for the application, which will often dictate the physical realization.

When the purpose of computation is information processing, we usually want it to execute as quickly as possible, with a minimum expenditure of energy and other resources, which are just physical means to the computational end. Therefore, progress in computer technology has been measured by a decrease in the amount of matter and energy involved in basic computational operations: from relatively massive relay armatures, to the substantial currents in vacuum tubes and the write currents for ferrite cores, to regularly decreasing operating voltages and charges in semiconductor devices (a major factor in Moore's Law). Much of the technological progress in computing has been directed at representing bits and implementing bit operations with less matter and energy. When embodied computation is used for physical applications, however, we may want to move more matter and energy rather than less, since the system itself or the desired physical effects may be large. Physically bigger bits, for example, may be more suitable to the application.

1.4. Programmable Matter

Embodied computation for physical effect is exemplified by *programmable matter*, in which computational methods are used to control the physical properties of a material.[52,53]

Programmable matter has many potential applications, including in radically reconfigurable systems. Reconfigurable systems are valuable because they allow a system to be adapted, for example, for a new mission, new circumstances, or for damage recovery, instead of being replaced, which may be economically or physically infeasible. A conventional reconfigurable system may be reconfigured by changing the connections among a fixed set of components, but

the range of reconfiguration is limited by the builtin components. Examples include field-programmable gate arrays (FPGAs) and field-programmable analog arrays (FPAAs). A *radically reconfigurable* system goes beyond this by allowing the physical properties of the components to be changed.[54] That is, rather than rearranging a fixed set of hardware resources, radical reconfiguration changes the hardware resources. This requires systems whose physical properties are under programmatic control.

One way to accomplish limited radical configurability is by implementing a random-access *configuration memory* with cells whose (possibly non-binary) states have distinct physical properties (e.g., conductance, reflectance, capacitance, switching, amplification, mechanical force). Then, as a program runs with at least some of its storage in the configuration memory, the properties of the cells will change under program control. We may call this *externally programmed* configuration or assembly, because the configuration memory is controlled by a separate program execution unit. Of course, the same can be accomplished by a conventional computer connected to the configuration memory as an I/O device, but then the computation would not be embodied; to be embodied the configuration memory should have an important role in the computational process (a matter of degree, of course).

Even though such a system gives some control over the physical properties of a system, it is limited by the state space of the memory cells, a function of their design, and we might wonder if there is a way to programmatically control a potentially unbounded variety of physical properties. This might seem unlikely, but nature has provided an existence proof: proteins.[25,54] Among many other things, proteins include the keratin of feathers, horns, and nails, the elastin and collagen of connective tissue, the tubulin that forms the cellular cytoskeleton (enabling rigidity and movement), and signaling molecules, ion channels, and enzymes. There are also active proteins, such as the rhodopsin, which senses light, motor proteins, proteins that make decisions by changing their conformation, and of course proteins that correct errors in DNA, and transcribe RNA to assemble other proteins. In summary, proteins are an effectively infinite class

of compounds with a very wide range of active and passive physical properties. What is also important for our purposes is that they are all produced by a common process from a limited set of basic components (20 amino acids).

Two other factors give proteins their generality and diversity. First, they are composed of long chains of amino acid residues, which relax into complex conformations that determine their physical properties (and hence their chemical and biological properties). It is the complexity of protein folding that gives them their enormous diversity of properties. Second, the amino acid sequences are encoded in DNA, which allows general programmability of the proteins' structure and facilitates a common set of information (computational) processes for exploring the space of DNA sequences (e.g., mutation, crossover, deletion, transposition, duplication, explored through evolution by natural selection). So the combinatorial diversity in DNA sequences translates to diversity of physical behavior in the proteins.

These ideas can be applied to artificial protein-like molecules as well. The first requirement is a class of polymers composed of a small fixed set of components so that the chains relax or fold into complex configurations with a very wide variety of physical properties. Second, we require a combinatorially rich data structure, such as a string, that represents the component sequence, and a means for translating the string into physical polymers (the embodied part of the computation). Aping biology again, the space of possible sequences can be searched by genetic algorithms, for example, to find polymer sequences that fold into useful structures. Radical reconfiguration, then, is able to change the sequences to assemble polymers with the required properties.

1.5. Artificial Morphogenesis

Proteins and protein-analogs use an information structure to determine a polymer chain, which then folds passively under physical forces into a configuration with functional properties. The resulting individual molecules may self-assemble into relatively homogeneous tissues or other materials with a statistically regular structure.

For more complex heterogeneous and irregular structures, we can use externally programmed assembly, but the resulting structures will be limited by the fixed physical arrangement of the cells, whose states are under program control. For more complex structures, we may turn to *internally programmed assembly*, in which the individual components need not react purely passively, but can have their own behavioral program by which they self-assemble into the required structure. An example of this approach is algorithmic assembly by DNA.[55–57]

For a general and robust approach for assembling more complex physical structures, especially those with a hierarchical structure spanning many length scales, perhaps from microns up to meters, we can look again to biology for inspiration because developing embryos accomplish this, coordinating an ever-increasing number of cells (ultimately in the trillions) to coordinate their movement, differentiation, and interaction to produce reliably a specific complex body form. This is certainly an example of embodied computation for physical effect, since the goal of the information processes embodied in the cells' interactions is for them to assemble themselves into a physical body.

In embryology *morphogenesis* refers to the process by which embryos develop 3D structures, and so the technological application of these ideas may be termed *artificial morphogenesis* or *morphogenetic engineering*. Projects investigating this technology have taken a variety of approaches, often differing in how closely they model the biological processes.[58–67]

Our approach to artificial morphogenesis follows a common embryological practice of using PDEs to describe morphogenetic processes. This is appropriate in biology because of the large number and small size of cells compared to the tissues they constitute, and also because developing tissues often have the characteristics of soft-matter (visco-elastic materials), which may be described by continuum mechanics. In artificial morphogenesis, PDEs have the advantage of being suited to describing processes involving very large numbers of very small agents (e.g., microrobots, synthetic microorganisms). Descriptions are also relatively independent of the size and

number of the agents, so long as the continuum approximation is close enough. This serves our goal of having morphogenetic algorithms that scale well to very large numbers of agents and that are relatively independent of the agents' size.

To facilitate and test morphogenetic algorithms, we have developed a morphogenetic programming language, which allows "substances" to be defined, with properties and behaviors described by PDEs.[68,69] We can execute these algorithms on a conventional computer, but if they were realized in an appropriate physical medium (e.g., a massive swarm of programmed microrobots or genetically engineered microorganisms), then the result of execution would be the desired physical structure. We are currently developing *global-to-local compilation* algorithms to translate from the PDEs (which describe the behavior of continua of infinitesimal particles) to behavioral programs for finite numbers of agents of a specific size while maintaining a large range of scale independence.[70]

Through simulation we have demonstrated the application of artificial morphogenesis to several problems, including the routing of dense fiber bundles between regions of an artificial brain, and the assembly of an insect-like body frame, with specified numbers of spinal segments, legs, and leg segments.[69–71]

1.6. Conclusions

Computation is a physical process, but of a very special kind in which the physical interactions can be described as information processes. For both scientific and technological reasons we should be exploring the full range of computational paradigms, both artificial and natural. One direction for future alternative computing paradigms is embodied computation, which focuses on the physical realization of computational processes. On the one hand, this suggests new ways that physical processes can be used for computation, thus providing directions for post-Moore's law computing. On the other, it points towards applications of computing in which the physical processes realizing the computation are the purpose of the computation, thereby using programs as a general means of directly determining

physical processes. In particular, the physical properties of materials can be controlled programmatically, and physical systems with complex hierarchical structures over many size scales can be assembled by morphogenetic algorithms.

References

1. B. J. MacLennan, Natural computation and non-Turing models of computation. *Theoretical Computer Science* **317**, 115–145 (2004).
2. B. J. MacLennan, Super-Turing or non-Turing? Extending the concept of computation. *Int. J. Unconvent. Comput.* **5**(3–4), 369–387 (2009).
3. J. Lipka, *Graphical and Mechanical Computation* (Wiley, New York, 1918).
4. W. Thomson, Harmonic analyzer, *Proc. Roy. Soci.* **27**, 371–373 (1878).
5. A. B. Clymer, The mechanical analog computers of Hannibal Ford and William Newell. *IEEE Ann. Hist. Comput.* **15**(2), 19–34 (1993).
6. J. S. Small, *The Analogue Alternative* (Routledge, London & New York, 2001).
7. B. J. MacLennan, The promise of analog computation. *Int. J. Gen. Syst.* **43**(7), 682–696 (2014). doi: 10.1080/03081079.2014.920997.
8. R. Landauer, Irreversibility and heat generation in the computing process. *IBM J. Res. Develop.* **5**(3), 183–191 (1961). Reprinted, Vol. 44 No. 1/2, Jan./March 2000, pp. 261–269.
9. A. Berut, A. Arakelyan, A. Petrosyan, S. Ciliberto, R. Dillenschneider, and E. Lutz, Experimental verification of Landauer's principle linking information and thermodynamics. *Nature* **483**, 187–189 (2012). doi: 10.1038/nature10872.
10. C. H. Bennett, Logical reversibility of computation. *IBM J. Res. Develop.* **17**(6), 525–532 (1973).
11. C. H. Bennett, The thermodynamics of computation — A review. *Int. J. Theo. Phys.* **21**(12), 905–940 (1982).
12. E. F. Fredkin and T. Toffoli, Conservative logic. *Int. J. Theo. Phys.* **21**(3/4), 219–253 (1982).
13. C. H. Bennett, Notes on Landauer's principle, reversible computation, and Maxwell's Demon. *Stud. Hist. Phil. Mod. Phys.* **34**, 501–510 (2003).
14. M. P. Frank, Introduction to reversible computing: Motivation, progress, and challenges. In *CF '05: Proceedings of the 2nd Conference on Computing Frontiers, Ischia, Italy*, May 4–6, 2005.
15. B. J. MacLennan, Bodies — Both informed and transformed: Embodied computation and information processing. In G. Dodig-Crnkovic and M. Burgin (eds.), *Information and Computation: Essays on Scientific and Philosophical Understanding of Foundations of Information and Computation*, vol. 2, *World Scientific Series in Information Studies* (World Scientific, Singapore, 2011), pp. 225–253.

16. B. J. MacLennan, The U-machine: A model of generalized computation. *Int. J. Unconvent. Comput.* **6**(3–4), 265–283 (2010).
17. B. J. MacLennan, Technology-independent design of neurocomputers: The universal field computer. In M. Caudill and C. Butler (eds.), *Proceedings of the IEEE First International Conference on Neural Networks*, vol. 3, (IEEE Press, 1987), pp. 39–49.
18. B. J. MacLennan, Field computation in natural and artificial intelligence. *Inform. Sci.* **119**, 73–89 (1999).
19. B. J. MacLennan, Field computation: A framework for quantum-inspired computing. In S. Bhattacharyya, U. Maulik, and P. Dutta (eds.), *Quantum Inspired Computational Intelligence: Research and Applications*, Chapter 3 (Morgan Kaufmann/Elsevier, Cambridge, MA, 2017). pp. 85–110. doi: http://dx.doi.org/10.1016/B978-0-12-804409-4.00003-6.
20. B. J. MacLennan, Topographic representation for quantum machine learning. In S. Bhattacharyya, I. Pan, A. Mani, S. De, E. Behrman, and S. Chakraborti (eds.), *Quantum Machine Learning*, Chapter 2 (De Gruyter, Berlin/Boston, 2020).
21. B. J. MacLennan, *Foundations of Field Computation*. URL http://web.eecs.utk.edu/~bmaclenn/FFC.pdf.
22. T. van Gelder. Dynamics and cognition. In J. Haugeland (ed.), *Mind Design II: Philosophy, Psychology and Artificial Intelligence*, Chapter 16 (MIT Press, Cambridge, MA, 1997), pp. 421–450, revised & enlarged edn.
23. B. J. MacLennan, Transcending Turing computability. *Minds Mach.* **13**, 3–22 (2003).
24. M. Johnson and T. Rohrer, We are live creatures: Embodiment, American pragmatism, and the cognitive organism. In J. Zlatev, T. Ziemke, R. Frank, and R. Dirven (eds.), *Body, Language, and Mind*, vol. 1 (Mouton de Gruyter, Berlin, 2007), pp. 17–54.
25. B. J. MacLennan, Embodied computation: Applying the physics of computation to artificial morphogenesis, *Parallel Process. Lett.* **22**(3), 1240013 (2012).
26. H. Hamann and H. Wörn, Embodied computation. *Parallel Process. Lett.* **17**(3), 287–298 (2007).
27. S. Stepney, The neglected pillar of material computation. *Physica D* **237**(9), 1157–64 (2008).
28. M. Ercsey-Ravasz and Z. Toroczkai, Optimization hardness as transient chaos in an analog approach to constraint satisfaction. *Nat. Phys.* **7**, 966–970 (2011).
29. B. Molnár and M. Ercsey-Ravasz, Asymmetric continuous-time neural networks without local traps for solving constraint satisfaction problems. *PLoS ONE* **8**(9), e73400 (2013). doi: 10.1371/journal.pone.0073400.
30. S. Haykin, *Neural Networks and Learning Machines*, 3rd edn. (Pearson Education, New York, 2008).
31. C. E. Shannon, Mathematical theory of the differential analyzer. *J. Math. Phys. Mass. Inst. Technol.* **20**, 337–354 (1941).

32. C. E. Shannon. Mathematical theory of the differential analyzer. In N. J. A. Sloane and A. D. Wyner (eds.), *Claude Elwood Shannon: Collected Papers* (IEEE Press, New York, 1993), pp. 496–513.
33. M. Pour-El, 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).
34. L. A. Rubel, The brain as an analog computer. *J. Theoret. Neurobiol.* **4**, 73–81 (1985).
35. L. Lipshitz and L. A. Rubel, A differentially algebraic replacment theorem. *Proc. Am. Math. Soci.* **99**(2), 367–72 (1987).
36. L. A. Rubel, Some mathematical limitations of the general-purpose analog computer. *Adv. Appl. Math.* **9**, 22–34 (1988).
37. M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, 10th anniversary edition (Cambridge University Press, Cambridge, 2010),
38. S. Lloyd and S. L. Braunstein, Quantum computation over continuous variables. *Phys. Rev. Lett.* **82**, 1784–1787 (1999). doi: 10.1103/PhysRevLett.82.1784. URL http://link.aps.org/doi/10.1103/PhysRevLett.82.1784.
39. O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Robot. Res.* **5**, 90–9 (1986).
40. M. Miller, B. Roysam, K. Smith, and J. O'Sullivan. Representing and computing regular languages on massively parallel networks. *IEEE Trans. Neural Netw.* **2**, 56–72 (1991).
41. E. Rimon and D. Koditschek. The construction of analytic diffeomorphisms for exact robot navigation on star worlds. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation, Scottsdale AZ* (IEEE Press, New York, 1989), pp. 21–6.
42. O. Steinbeck, A. Tóth, and K. Showalter, Navigating complex labyrinths: Optimal paths from chemical waves. *Science* **267**, 868–71 (1995).
43. P. Ting and R. Iltis, Diffusion network architectures for implementation of Gibbs samplers with applications to assignment problems. *IEEE Trans. Neural Netw.* **5**, 622–38 (1994).
44. J. W. Mills, B. Himebaugh, B. Kopecky, M. Parker, C. Shue, and C. Weilemann, "Empty space" computes: The evolution of an unconventional supercomputer. In *Proceedings of the 3rd Conference on Computing Frontiers* (ACM Press, New York, 2006), pp. 115–26.
45. A. Adamatzky, *Physarum Machines: Computers from Slime Mould.* World Scientific Series on Nonlinear Science Series A: Volume 74 (World Scientific, Singapore, 2010).
46. A. Adamatzky, B. De Lacy Costello, and T. Asai, *Reaction-Diffusion Computers* (Elsevier, Amsterdam, 2005).
47. A. Turing, The chemical basis of morphogenesis. *Philos. Trans. Roy. Soci.* **B 237**, 37–72 (1952).
48. J. D. Murray, *Lectures on Nonlinear Differential-Equation Models in Biology* (Oxford, Oxford, 1977).

49. P. K. Maini and H. G. Othmer (eds.), *Mathematical Models for Biological Pattern Formation* (Springer-Verlag, New York, 2001).
50. S. Camazine, J. Deneubourg, N. R. Franks, G. Sneyd, J. Theraulaz, and E. Bonabeau, *Self-organization in Biological Systems* (Princeton University Press, Princeton, New Jersey, 2001).
51. G. Forgacs and S. A. Newman, *Biological Physics of the Developing Embryo* (Cambridge University Press, Cambridge, UK, 2005).
52. T. Toffoli and N. Margolus, Programmable matter: Concepts and realization. *Physica D: Nonlinear Phenomena* **47**(1), 263–272 (1991). ISSN 0167-2789. doi: https://doi.org/10.1016/0167-2789(91)90296-L. URL http://www.sciencedirect.com/science/article/pii/016727899190296L.
53. S. C. Goldstein, J. D. Campbell, and T. C. Mowry, Programmable matter, *Computer* **38**(6), 99–101 (June, 2005).
54. B. J. MacLennan, The morphogenetic path to programmable matter, *Proceedings of the IEEE* **103**(7), 1226–1232 (2015). doi: 10.1109/JPROC.2015.2425394.
55. P. Rothemund and E. Winfree. The program-size complexity of self-assembled squares. In *Symposium on Theory of Computing* (*STOC*) (Association for Computing Machinery, New York, 2000), pp. 459–468.
56. R. Barish, P. Rothemund, and E. Winfree, Two computational primitives for algorithmic self-assembly: Copying and counting. *Nano Lett.* **5**, 2586–92 (2005).
57. P. Rothemund, N. Papadakis, and E. Winfree, Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology* **2**(12), 2041–53 (2004).
58. H. Kitano. Morphogenesis for evolvable systems. In E. Sanchez and M. Tomassini (eds.), *Towards Evolvable Hardware: The Evolutionary Engineering Approach* (Springer, Berlin, 1996), pp. 99–117.
59. R. Nagpal, A. Kondacs, and C. Chang. Programming methodology for biologically-inspired self-assembling systems. In *AAAI Spring Symposium on Computational Synthesis: From Basic Building Blocks to High Level Functionality* (March, 2003). URL http://www.eecs.harvard.edu/ssr/papers/aaaiSS03-nagpal.pdf.
60. A. Spicher, O. Michel, and J. Giavitto. Algorithmic self-assembly by accretion and by carving in MGS. In *Proceedings of the 7th International Conference on Artificial Evolution* (*EA '05*), number 3871 in Lecture Notes in Computer Science (Springer-Verlag, Berlin, 2005), pp. 189–200.
61. S. Murata and H. Kurokawa, Self-reconfigurable robots: Shape-changing cellular robots can exceed conventional robot flexibility. *IEEE Robot. Autom. Magaz.* 71–78 (2007).
62. R. Doursat. Organically grown architectures: Creating decentralized, autonomous systems by embryomorphic engineering. In R. P. Würtz (ed.), *Organic Computing* (Springer, 2008), pp. 167–200.
63. B. J. MacLennan, Morphogenesis as a model for nano communication, *Nano Commun. Netw.* **1**(3), 199–208 (2010). doi: 10.1016/j.nancom.2010.09.007.

64. P. Bourgine and A. Lesne (eds.), *Morphogenesis: Origins of Patterns and Shapes* (Springer, Berlin, 2011).
65. J. Giavitto and A. Spicher. Computer morphogenesis. In P. Bourgine and A. Lesne (eds.), *Morphogenesis: Origins of Patterns and Shapes* (Springer, Berlin, 2011), pp. 315–340.
66. R. Doursat, H. Sayama, and O. Michel, A review of morphogenetic engineering. *Nat. Comput.* **12**(4), 517–535 (2013). ISSN 1567-7818, 1572-9796. doi: 10.1007/s11047-013-9398-1.
67. H. Oh, A. Ramezan Shirazi, C. Sun, and Y. Jin, Bio-inspired self-organising multi-robot pattern formation: A review. *Robot. Auton. Syst.* **91**, 83–100 (2017). ISSN 09218890. doi: 10.1016/j.robot.2016.12.006. URL https://linkinghub.elsevier.com/retrieve/pii/S0921889016300185.
68. B. J. MacLennan, 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).
69. B. J. MacLennan, A morphogenetic program for path formation by continuous flocking. *Int. J. Unconvent. Comput.* **14**, 91–119 (2019).
70. B. J. MacLennan and A. C. McBride, Swarm intelligence for morphogenetic engineering. In A. Schumann (ed.), *Swarm Intelligence: From Social Bacteria to Human Beings* (Taylor & Francis/CRC, 2020).
71. B. J. MacLennan. Coordinating swarms of microscopic agents to assemble complex structures. In Y. Tan (ed.), *Swarm Intelligence, Vol. 1: Principles, Current Algorithms and Methods*, PBCE 119, chapter 20 (Institution of Engineering and Technology, 2018), pp. 583–612.