The Nature of Computing — Computing in Nature

DRAFT

B. J. MacLennan
Department of Computer Science
University of Tennessee
Knoxville, TN 37996-3450, USA
www.cs.utk.edu/~mclennan

Abstract

My goal in this presentation is to recontextualize the concept of computation. I will begin by reviewing the historical roots of Church-Turing computation to remind us that the theory exists in a *frame of relevance*, which underlies the assumptions on which it rests and the questions it is suited to answer. I will argue that, although this frame of relevance is appropriate in many circumstances, there are many important applications of the idea of computation for which it is not relevant. These include *natural computation* (computation occurring in or inspired by nature), *nanocomputation* (computation based on nanoscale objects and processes), and computation based on quantum theory. As a consequence we need, not so much to abandon the Church-Turing model of computation, as to supplement it with new models based on different assumptions and suited to answering different questions. Therefore I will discuss alternative frames of relevance more suited to the interrelated application areas of natural computation, emergent computation, and nanocomputation. Central issues include continuity, indeterminacy, and parallelism. Finally, I will argue that once we understand computation in a broader sense than the Church-Turing model, we begin to see new possibilities for using natural processes to achieve our computational goals. These possibilities will increase in importance as we approach the limits of electronic binary logic as a basis for computation. They will also help us to understand the computational processes in the heart of nature.

I. Introduction

My goal in this presentation is to recontextualize the concept of computation. I will begin by reviewing the historical roots of Church-Turing computation to remind us that the theory exists in a *frame of relevance*, which underlies the assumptions on which it rests and the questions it is suited to answer. I will argue that, although this frame of relevance is appropriate in many circumstances, there are many important applications of the idea of computation for which it is not relevant. These include natural computation, nanocomputation, and quantum computation. As a consequence we need, not so much to abandon the Church-Turing model of computation, as to supplement it with new models based on different assumptions and suited to answering different questions. Therefore I will discuss alternative frames of relevance more suited to the interrelated application areas of *natural computation*, *nanocomputation*, and quantum computation. Finally, I will argue that once we understand computation in this broader sense, we begin to see new possibilities for using natural processes to achieve our computational goals, which will increase in importance as we approach the limits of electronic binary logic.

II. Weaning Ourselves Away from Church-Turing Computability

It is important to keep in mind that the Turing machine is a *model* of computation. Like all models, its purpose is to facilitate describing or reasoning about some other class of phenomena of which it is a model. A model accomplishes this purpose by being similar to its object in relevant ways, but different in other, irrelevant ways, and it is these differences that make the model more tractable than the original phenomena. But how do we know what is relevant or not? Every model is suited to pose and answer certain classes of questions but not others, which we may call the *frame of relevance* of the model. Although a model's frame of relevance often is unstated and taken for granted, we must expose it and make it explicit in order to understand the range of applicability of a model and to evaluate its effectiveness within its frame of relevance. What, then, is the frame of relevance of the Turing-machine model of computation?

As we know, the Church-Turing theory of computation was developed to address issues of formal calculability and provability in axiomatic mathematics. The assumptions that underly Church-Turing computation are reasonable in that context, but we must consider them critically before applying the model in other contexts. For example, for addressing the questions of what is, in principle, effectively calculable or formally provable, it is reasonable to require only that there be a finite number of steps requiring finite resources. As a consequence, according to this model, something is computable if it can be accomplished *eventually with unlimited resources*.

Another consequence of the historical roots of the Church-Turing theory of computation is its definition of computing power in terms of classes of functions. A function is computable if, given an input, we will get the correct output eventually, given unlimited resources. Of course, the theory can address questions of computation time and space, but the framework of the theory limits its applicability to asymptotic complexity, polynomial-time reducibility, and so forth. The roots of the idea that computation is a process of taking an *input*, calculating for a while, and producing an *output* can be found in the theory of effective calculability as well as in contemporary applications of the first computers, such as ballistics calculations, code breaking, and business accounting.

The Church-Turing model of computation, like all models, makes a number of *idealizing assumptions* appropriate to its frame of relevance. Many of these assumptions are captured by the idea of a calculus, but a phenomenological analysis of this concept is necessary to reveal the background of assumptions. Although there is not time today to discuss them in detail, it may be worthwhile to mention them briefly.

The model of information representation derives from mathematical formulas. As idealized in calculi, representations are *formal*, *finite*, and *definite*. We assume that *tokens* can be definitively discriminated from the background and from one another, and we assume that they can be classified by a mechanical procedure into exactly one of a finite number of *types*. The texts, or as we might say, data structures, are required to be finite in size, but also *finite in depth*; that is, as we divide it into parts, we will eventually reach its smallest, atomic constituents, which are tokens. These and other assumptions are taken for granted by the Church-Turing model of computation. Although the originators of the model discussed some of them, many people today do not think of them as idealizing assumptions that are not appropriate in some frames of relevance.

A similar array of idealizing assumptions underly the Church-Turing model of information processing, in particular, that it is *formal*, *finite*, and *definite*. Applicability of basic operations depends only on mechanically determinable properties of the tokens, and there is no ambiguity in the applicability of these operations (although it is permissible for several operations to be applicable). Each basic operation is atomic, and a computation terminates after a finite number of these atomic operations. Also, it is definitely determinable whether a computation has terminated. Again, these are largely unquestioned assumptions of the Church-Turing model.

Since there is not time here to go into the details, I will only remark that it is the assumptions underlying these notions of discrete representation and processing that permit the processes to be described by rules that can be represented themselves as finite discrete structures. This is the foundation of the ability of one calculus to emulate another based on a description of the latter in rules, notable examples of which are the universal Turing machine and other universal computing systems.

III. Alternative Frames of Relevance

It certainly could, and has, been argued that the notion of computation was vague before that pioneering work of Church, Turing, and others, and that what they did was analyze, define, and express this previously informal notion in more precise terms. However, I want to argue that there are important concerns connected with computation that are outside the frame of relevance of the Church-Turing model. Therefore we need to consider these alternative frames and the models suited to them.

First, I will discuss *natural computation*, which may be defined as computation occurring in nature or inspired by that occurring in nature. Natural computation occurs, of course, in the brains of many species, but also in the control systems of microorganisms and in the self-organized collective behavior of groups of animals, such as insect colonies, flocks of birds, and so on. Natural computation is an important research area because robust, efficient, and effective natural systems can show us how to design better artificial computational systems, and our artificial systems, in turn, can suggest models of computational processes in nature. Such fruitful interchange is already apparent in the theories of neural nets and complex systems. However, natural computation occurs in a different frame of relevance from conventional computing, and so it answers questions about the relative power and equivalence of computing systems from a different perspective, and at the same time it asks entirely new questions.

First of all, natural computation is often more like realtime control than the computation of a function. In nature the purpose of computation is frequently to generate continuous control signals in response to continuous sensor inputs. The system continues to compute so long as the natural systems exists.

Therefore, eventual termination is generally irrelevant in natural computation. Typically, we are concerned with whether a result, such as a decision to act, can be delivered with a *fixed realtime bound*, or whether continuous control signals can be generated in realtime. Therefore, we are not concerned with speed in terms of a number of abstract computation steps, but with realtime, and therefore how the rate of computation relates to the rates of the physical processes by which it is implemented. Also, in the usual theory the criterion is whether a correct output will be produced eventually, whereas in natural computation it may be more important to know how closely a preliminary result approximates to the correct one, since the system may be forced by realtime constraints to use the preliminary result.

Asymptotic complexity is not so important in natural computation as in conventional computation, because in applications of natural computation the size of the inputs are often fixed, for example, they are determined by the physical structure of the sensors. Other bases for comparing computational processes are more relevant. For example, for fixed-sized inputs and processing resources, which computational process has a greater *generality* of response, that is a wider range of inputs to which it responds well?

A closely related criterion relevant to natural computation is how *flexibly* systems respond to novel inputs, that is, to inputs outside its range. For an artificial system this range is the set of inputs it was designed to process correctly; for a natural system it is the set of inputs that it has evolved to process in its environment of evolutionary adaptedness.

Adaptability is another relevant basis of comparison for natural computation systems, that is, how well do they accommodate themselves to changing circumstances. In this regard we are interested in the range, quality, speed, and stability of adaptation.

Finally, the function of natural computation is to allow imperfect physical agents to operate effectively in the extremely complex, messy natural world. Therefore, in natural computation, we must take noise, uncertainty, errors, faults, and damage as givens, not peripheral issues added on as an afterthought, if considered at all, as is often done in conventional computing. These considerations affect both the structure of natural computations and the criteria by which they are compared.

It will be apparent that the Church-Turing model is not particularly suited to addressing many of these issues, and in a number of cases begs the questions or makes assumptions incompatible with addressing them.

There is not enough time to say much about nanocomputation, so I will just mention some of the issues that are relevant when computing with nanoscale devices and processes. First, at the nanoscale, error and noise are major, unavoidable factors. Even at low temperatures, thermal noise looms large, and as sizes are decreased quantum indeterminacy becomes significant. As a consequence error, noise, and indeterminacy must be a part of our models of computation; it is a bad idealization to assume it can be eliminated. Rather, we must design under the assumption they are present and even learn to exploit them as positive effects. Second, many nanoscale processes, such as molecular ones, will be to some degree reversible, so we cannot assume that computation proceeds monotonically forward. Therefore we will be most interested in computations that progress on a statistical or macroscopic scale even in the face of microscopic reversals. Finally, many nanocomputations, especially molecular computations, will not have well-defined computational steps, for they will proceed asynchronously in continuous-time parallelism.

IV. Computation in General

A. Traditional notions of digital and analog computation

Historically, there have been many kinds of computation. Digital computation, effective calculability, formal axiomatics, and the Church-Turing model of computation have their roots in arithmetical algorithms, using written numerals as well as devices, such as abaci, that have been used since ancient times. However, there are equally important examples of continuous computation, including devices, such as the slide-rule, and computational procedures, such as the traditional, compass-and-straightedge constructions of Euclidean geometry. In more modern times, we have had both analog and digital electronic and mechanical computers. The contemporary overwhelming visibility of digital computers should not blind us to this fact. Therefore both history and the existence of alternative frames of relevance show us the importance of non-Turing models of computation.

B. Definition of computation

How, then, can we define "computation" in sufficiently broad terms? Historically, computation has involved the manipulation of mathematical objects by means of physical operations. The familiar examples are arithmetic operations on numbers, but we have also seen geometric operations on spatial objects and logical operations on formal propositions. Modern computers manipulate a much wider variety of objects, including character strings, images, sounds, and much else. Therefore, when I say that computation uses physical operations to accomplish the mathematical manipulation of mathematical objects, I mean it in the broadest sense, that is, the abstract manipulation of abstract objects. In terms of the traditional separation of *form* and *matter*, we may say that computation uses material processes to accomplish formal manipulation of abstract forms.

The forms manipulated by a computation must be materially realized in some way, but the characteristic of computation that distinguishes it from other physical processes is that it is independent of specific material realization. That is, although a computation must be materially realized in *some* way, it can be realized in any physical system having the required formal structure. (Of course, there will be practical differences between different physical realizations, but I will defer consideration of them until later.) So, when we consider computation *qua computation*, we must, on the one hand, restrict our attention to formal structures that are physically realizable, but, on the other, consider the processes independently of any particular physical realization.

These observations provide a basis for determining whether or not a particular physical system (in the brain, for example) is computational. If the system could, in principle at least, be replaced by another having the same formal properties and still accomplish its purpose, then it is reasonable to consider the system computational. For example, it seems likely that primary visual cortex performs a Gabor wavelet transform on the image from the lateral geniculate nucleus. If this is so, then it could be replaced, in principle, by an artificial device performing the same computation but with a completely different physical realization. (We have the beginnings of such substitutions in cochlear implants and similar devices.) On the other hand, if a system can fulfill its purpose only by control of particular substances or particular forms of energy, then it cannot be purely computational.

For example, a feedback control system regulating the quantity of a certain hormone in the blood would not accomplish that purpose if the quantity were represented in a different medium, such as electrical charge density; therefore this control system is not purely computational. Similarly, the immune system, although it processes information, is not purely computational, since its chemical realization is essential to its function. To give a technological example, a radio transmitter cannot be purely computational, because its purpose is to produce electromagnetic radiation of a certain frequency; it could not accomplish its purpose by manipulating information represented in a different medium, such as light. These examples illustrate that while some systems are purely computational, many, especially in biology, accomplish non-computational purposes as well. Also, a computational system will not be able to accomplish its purpose unless it can interface properly with its environment; this is a topic I will consider later.

Based on the foregoing considerations, I would like to propose the following definition of computation:

> **Definition:** *Computation* is a physical process, the purpose of which is the abstract manipulation of abstract objects.

Alternately, we may say that computation accomplishes the formal manipulation of formal objects by means of their material embodiment. Next I will define the relation between the physical and abstract processes:

> **Definition:** A physical system *realizes* a computation if, at the level of abstraction appropriate to its purpose, the abstract manipulation of the abstract objects is a sufficiently accurate model of the physical process. Such a physical system is called a *realization* of the computation.

That is, the physical system realizes the computation if we can see the material process as a sufficiently accurate embodiment of the formal structure, where the sufficiency of the accuracy must be evaluated in the context of the system's purpose. Next I will suggest a definition by which we can classify various systems, both natural and artificial, as computational:

> **Definition:** A physical system is *computational* if its purpose is to realize a computation.

Finally, for completeness:

> **Definition:** A *computer* is an artificial computational system.

Thus I restrict the term "computer" to intentionally constructed computational devices; to call the brain a computer is a metaphor. These definitions raise a number of issues, which I will discuss briefly. No doubt they can be improved.

These definitions, which I have proposed, make reference to the *purpose* of a system, but philosophers and scientists are justifiably wary of appeals to purpose, especially in a biological context. However, I claim that the use of purpose in the definition of computation is unproblematic, for in most cases of practical interest, purpose is easy to establish.

First consider computers, which I have defined as artificial computational systems, that is, artificial systems whose *purpose* is to realize the abstract manipulation of abstract objects. Since computers are designed by human beings, we can simply ask the designers what their purpose was in constructing the device. Here I am making the commonsense assumption that people know why they have undertaken a project, which is unproblematic in most circumstances.

For biological systems the appeal to purpose is more problematic. Nevertheless, biologists routinely investigate the purpose of biological systems, such as the digestive system and immune system, and make empirically testable hypotheses about their purposes. Ultimately such claims of purpose are reduced to the selective advantage of a particular species in that species' environment of evolutionary adaptedness, that is, the environment in which it has historically evolved, but in most cases we can appeal to more familiar ideas of purpose. When we are dealing with organ systems, such as the nervous system, establishing purpose is relatively unproblematic.

However, I should mention one problem that does arise in biology and can be expected to arise in our biologically-inspired robots. That is, while the distinction between computational and non-computational systems is significant for us, it does not seem to be especially significant to biology. The reason may be that we are concerned with the *multiple realizability* of computations, that is, with the fact that they have alternative realizations. For this property allows us to consider the implementation of a computation in a different technology, for example in electronics rather than neurons. In nature, typically, the realization is given, since natural life is built upon a limited range of substances and processes. On the other hand, there is typically selective pressure in favor of exploiting a biological system for as many purposes as possible. Therefore, in a biological context, we expect physical systems to serve multiple purposes, and therefore many such systems will not be purely computational; they will fulfill other functions besides computation. From this perspective, it is remarkable how free the nervous systems of all animals are from non-computational functions.

C. Autonomy

It is important to realize that in natural computation, as well as in many other applications of computers, the computation of a mathematical function is not the most appropriate standard for evaluating models of computation. Certainly, the idea of a computation that is given an input and eventually produces an output was an appropriate idea for understanding effective calculability and decidability in formal systems and as a model of batch computation, but it is not appropriate for systems, such as autonomous agents, in continuous realtime interaction with their environments. Nor is it especially suited for dealing with situations, such as those found in natural computation, in which there is large and unavoidable error and uncertainty in sensor inputs and in actuator effects. Therefore, computation needs to be interpreted in a broader sense that the computation of mathematical functions. Realtime control systems and soft constraint maintenance systems are more appropriate models for many applications.

D. Transduction

I have emphasized that the purpose of computation is the abstract manipulation of abstract objects, but obviously this manipulation will be pointless unless the computational system interfaces with its environment in some way. Certainly our computers need input and output interfaces in order to be useful. So also computational systems in the brain must interface with sensory receptors, muscles, and many other noncomputational systems in order to be useful. In addition to these practical issues, the computational interface to the physical world is also relevant to the *symbol grounding problem*, the philosophical question of how abstract symbols can have real-world content. Therefore we need to consider the interface between a computational system and its environment, which comprises *input* and *output transducers*.

Consider first input transducers, for which we can take a photo sensor as a concrete example. Its job is to take a light intensity and translate it into a computational representation. For example, if we are dealing with electronic analog computation, then the job of this input transducer is to translate a light intensity into a voltage, for example. On the other hand, if we were dealing with a fluidic analog computer, the job of the input transducer might be to translate light intensity into water pressure. In principal, we can use any physical system with the appropriate formal properties to implement this analog computation, and so the input transducer must re-represent input in the corresponding medium of computation (for example, voltage or water pressure). The output of an input transducer is computational, and so it is multiply realizable, just like the rest of the computation. On the other hand, the input to the input transducer is physically determined by the purpose of the computation; for example if it responds to air pressure rather than light intensity, the computational system will not be able to serve its function.

The same of course applies on the output side, for the task of an output transducer is to translate from the generic computational medium to the specific physical output required to serve the computational system's purpose.

The relation of transduction to computation is easiest to see in the case of analog computers. The inputs and outputs of the computational system have some physical dimensions (light intensity, air pressure, mechanical force, etc.), because they must have a specific physical realization for the system to accomplish its purpose. On the other hand, the computation itself is essentially dimensionless, since it manipulates pure numbers. Of course, these internal numbers must be represented by *some* physical quantities, but they can be represented in *any* appropriate physical medium. In other words, computation is *generically* realized, that is, realized by any physical system with an appropriate formal structure, whereas the inputs and outputs are *specifically* realized, that is, constrained by the environment with which they interface to accomplish the computational system's purpose.

We can also describe transduction in terms of form and matter, where by matter I mean any physical instantiation. In effect, computation is purely formal, for although it must be physically realized, the material substrate of the computation is irrelevant so long as it supports the formal structure. A pure input transduction, therefore, translates a physical configuration into the computational medium, that is, it preserves the form while changing its material embodiment. If we neglect the physical realization of the computation, we can think of a pure input transduction as acquiring the form of the input and leaving the matter behind. Similarly, a pure output transduction takes a form produced by the computation and imposes it on the specific material substrate required for output.

So we can think of pure transduction as changing matter while leaving form unchanged, and computation as transforming form independently of matter. In fact, most transduction is not pure, for it modifies the form as well as the material substrate, for example, by filtering. Likewise, transductions between digital and analog representations transform the signal between discrete and continuous spaces.

E. Classification of computational processes

I have tried to frame this definition of computation quite broadly, to make it *topology-neutral*, so that it encompasses all the forms of computation found in natural and artificial systems. It includes, of course, the familiar computational processes operating in discrete steps and on discrete state spaces, such as in ordinary digital computers. It also includes continuous-time processes operating on continuous state spaces, such as found in conventional analog computers. However, it also includes hybrid processes, incorporating both discrete and continuous computation, so long as they are mathematically consistent. As we expand our computational technologies outside of the binary electronic realm, we will have to consider these other topologies of computation. This is not so much a problem as an opportunity, for they are better matched to many important applications, especially in natural computation.

F. Approximate realization

In connection with the classification of computational processes in terms of their topologies, it is necessary to say a few words about the relation between computations and their realizations. A little thought will show that a computation and one of its realizations do not have to be of the same type, for example, discrete or continuous. For example, the discrete computations performed on our digital computers are in fact realized by continuous physical systems obeying Maxwell's equations. The realization is approximate, but exact enough for practical purposes. Conversely a discrete system can approximately realize a continuous system, much like numerical integration on a digital computer. In comparing the topologies of the computation and its realization, we must describe the physical process at the relevant level of analysis, for a physical system that is discrete on one level may be continuous on another.

V. Expanding the Range of Physical Computation

A. General guidelines: matching natural processes and computations

Next I would like to discuss the prospects for expanding the range of physical processes that can be applied to computation. I will consider some general guidelines, and then look at applications in natural computation, nanocomputation, and quantum computation.

A powerful feedback loop has amplified the success of digital VLSI technology to the exclusion of all other computational technologies. The success of digital VLSI encourages and finances investment in improved tools and technologies, which increase the success of digital VLSI. Unfortunately this powerful feedback loop is rapidly becoming a vicious cycle. We know that there are limits to digital VLSI technology, and, although estimates differ, they are not too far off. We have assumed there will always be more bits and more MIPS, but that is false. Unfortunately, alternative technologies and models of computation remain undeveloped and largely uninvestigated, because the rapid advance of digital VLSI has passed them by before they could get out of the starting gate. Investigation of alternative computational technologies is further constrained by the assumption that they must support binary logic, because that is the only way we know how to compute, or because our investment in this model of computation is so large. Nevertheless, we must break out of this vicious cycle or we will be technologically unprepared when digital VLSI finally, and inevitably, hits the brick wall.

Therefore, as means of breaking out of this vicious cycle, I would like to step back and look at computation and computational technologies in the broadest sense. What sorts of physical processes can we reasonably expect to use for computation? Based on my preceding remarks, we can see that *any* mathematical process, that is, any abstract manipulation of abstract objects, is a potential computation. Of course, not all of these are useful, but mathematical models in science and engineering offer many possibilities. Aside from *de novo* applications of mathematical techniques to practical problems, mathematical models of computation in natural systems may be applied to our computational needs. Of course, a computation must be physically realizable, which means that we need to find at least one physical process for which the desired computation is a good model. Therefore, in principle, *any reasonably controllable, mathematically described, physical process can be used for computation*.

In some sense, this approach is a return to old notion of analog computation, according to which we find some appropriate physical analog to the system of interest. However, in this sense, all computation is analog computation, for even digital computers are physical realizations of the abstract structure of some system of interest. (Note that we must distinguish "analog" in the sense of physical realization from "analog" in the sense of continuous computation, which is opposed to "digital" in the sense of discrete computation. The two senses of "analog" are independent.)

Of course, there are practical limitations on the physical processes usable for computation, but the range of possible computational technologies is much broader than might be suggested by a narrow definition of computation. Considering some of the requirements for computational technologies will reveal some of the possibilities as well as the limitations.

One obvious issue is speed. The rate of the physical process may be either too slow or too fast for a particular computational application. That it might be too slow is obvious, for our world is obsessed by speed. Nevertheless, there are many applications that have limited speed requirements, for example, if they are interacting with an environment with its own limited rates. Conversely, these applications may benefit from other characteristics of a slower technology, such as energy efficiency, insensitivity to uncertainty, error, and damage, and the ability to adapt or repair itself. Another consideration that may supersede speed is whether the material substrate is suited to the application: is it organic or inorganic? Living or nonliving? Chemical, optical, or electrical?

It might be less obvious that a physical realization might be too fast for an application, but that is because we are stuck in our batch processing mentality, according to which we want the output from our job as quickly as possible. However, many applications of natural computation are more like control processes, in which it may be more important that the speed of computation is matched to the system's environment. A system may be too responsive, leading to instability and other problems, although such an overly responsive system may be dampened.

A second requirement is the ability to implement the transducers required for the application. Although the computation is theoretically independent of its physical embodiment, its inputs and outputs are not, and some conversions to and from a computational medium may be easier than others. For example, if the inputs and outputs to a computation are chemical, then chemical or molecular computation may demand simpler transducers than electronic computation. Also, if the system to be controlled is biological, then some form of biological computation may suit it best.

Finally I should mention that a physical realization should have the accuracy, stability, controllability, and so forth required for the application. Fortunately, natural computation gives us many examples of useful computations that are accomplished by realizations that are not very accurate, for example, neuronal signals have at most about one digit of precision. Also, nature shows us how systems that are subject to many sources of noise and error may be stabilized and accomplish their purposes.

A key component of the vicious cycle is that we know so well how to design digital computers and program them. We are naturally reluctant to abandon this investment, which pays off so well, but so long as we restrict our attention to it, we will be blind to the opportunities of other technologies. But no one is going to invest much time or money in technologies that we don't know how to use. How to break the cycle?

I believe that natural computation provides the best opportunity. Nature provides many examples of useful computations based on different models from digital logic. When we understand these processes in computational terms, that is, as abstractions independent of their physical realizations in nature, we can begin to see how to apply them to our own computational needs and how to implement them in alternative physical processes. As examples we may take information processing and control in the brain, and emergent self-organization in animal societies, both of which already have been applied to a variety of computational problems. (I am thinking of artificial neural networks, genetic algorithms, particle swarm optimization, and so forth.) But there is much more that we can learn from these and other natural computation systems, and we have not made much progress in developing computers better suited to them. More generally we need to increase our understanding of computation in nature and keep our eyes open for physical processes with useful mathematical structure.

Therefore, one important step toward a more broadly based computer technology will be a library of well-matched computational methods and physical realizations. For example, for a particular model such as the continuous Hopfield network, we should have available a growing repertoire of physical processes for which the Hopfield net is a reasonably accurate model. For example, they would include an assortment of relaxation processes. Therefore, if we have a problem for which the continuous Hopfield net is a good computational solution, we can consider the available realizations according to criteria such as those already mentioned: speed, stability, implementation medium, availability or feasibility of transducers, and so forth.

An important lesson learned from digital computer technology is the value of programmable general-purpose computers, for prototyping of special-purpose computers as well as for use in production system. Therefore to make better use of an expanded range of computational methodologies and technologies, it will useful to have general-purpose computers in which computational process is controlled by easily modifiable parameters. That is, we will want generic computers capable of a wide range of specific computations under the control of an easily modifiable representation. As has been the case for digital computers, the availability of such general-purpose computers will accelerate the development and application of new computational models and technologies.

We must be careful, however, lest we fall into the "Turing Trap," which is to assume that the notion of universal computation found in Turing machine theory is the appropriate notion in all contexts. The criteria of universal computation defined by Turing and his contemporaries was appropriate for their purposes, that is, studying effective calculability and derivability in formal mathematics. For them, all that mattered was whether a result was obtainable in a finite number of atomic operations and using a finite number of discrete units of space. Two machines, for example a particular Turing machine and a programmed universal Turing machine, were considered to be of the same power if they computed the same function by these criteria. Notions of equivalence and reducibility in contemporary complexity theory are not much different.

It is obvious that there are many important uses of computers, such as realtime control applications, for which this notion of universality is irrelevant. In such applications, one computer can be said to emulate another only if it does so at the same speed. In other cases, a general-purpose computer may be required to emulate a particular computer with at most a fixed extra amount of computational resources, such as storage space. The point is, is that in the full range of computer applications, in particular in natural computation, there may be much different criteria of equivalence than computing the same mathematical function. Therefore, in any particular application area, we must consider in what ways the programmed general-purpose computer must behave the same as the computer it is emulating, and it what ways it may behave differently, and by how much. That is, each notion of universality comes with a frame of relevance, and we must uncover and explicate the frame of relevance appropriate to our application area.

Fortunately there has been some work in this area. For example, theoretical analysis of general-purpose analog computation goes back to Claude Shannon in the 19040s, with more recent work by Pour-El (1974) and Rubel (1981, 1993). In the area of neural networks we have the Kolmogorov approximation theorem, which defines one notion of universality for feed-forward neural networks, although perhaps not a very useful one. Also, I have done some work on general-purpose field computers (MacLennan 1987, 1990, 1999). In any case, much more work needs to be done, especially towards articulating the relation between notions of universality and their frames of relevance.

It is worth remarking that these new types of general-purpose computers might not be programmed with anything that looks like an ordinary program, that is, a textual description of rules of operation. Some general-purpose analog computers will be programmed by specifying a pattern of interconnections, essentially a wiring diagram, for a fixed set of analog devices; this is not too different from a conventional program. General purpose neural networks may be programmed by specifying connection strengths, that is, by means of an array of analog values. For other analog computers, including field computers, computation is described by spatiotemporal continua of information, such as visible or audible images. Such an image might be used, for example, to govern a gradient descent process. In these cases, as I have discussed in some of my earlier papers, it becomes more appropriate to speak of a *guiding image* than a program. We are, indeed, quite far from universal Turing machines and the associated notions of programs and computing, but non-Turing computation is often more relevant in natural computation and other new domains of computation.

B. Natural Computation

Computation in nature gives us many examples of the matching of physical processes to the needs of natural computation, and so we may learn valuable lessons from nature. First, we may apply the actual natural processes in our artificial systems, for example using biological neurons or populations of microorganisms for computation. Second, by understanding the formal structure of these computational systems in nature, we may realize them in alternative physical systems with the same abstract structure. For example, neural computation or insect colony self-organization might be realized in an optical system.

C. Nanocomputation

Our broadened view of computation also provides new opportunities for nanocomputation; we do not have to limit ourselves to attempting to implement binary digital electronics at the nanoscale. Rather, we can take our physical understanding of nanoscale processes and use them as direct realization of non-Turing computations. In order to make effective use of these nanoscale processes we will need models of computation that are oriented toward massive, fine-grain, asychronous, continuous-time parallelism, and that are effective in the presence of reversibility and a high probability of error and noise. By matching nanoscale processes to computational applications, many of these properties can be changed from problems into assets.

D. Quantum and Quantum-like Computation

Next I would like to say a few words about quantum computation. Current proposals combine the ability to do parallel computation in linear superposition with conventional binary computation. Quantum decoherence is a formidable problem, especially for a non-trivial number of qubits, but there is slow progress. Because it is not my specialty, I will not discuss that research here, but I will mention two somewhat different approaches to quantum computation.

First, I would like to point out that current approaches to quantum computation under-utilize the representational capabilities of the wave-function. A wave-function is a complex-valued spatially extended continuous function, an element of an infinite-dimensional Hilbert space. Therefore its information capacity is much greater than one bit! To be able to use this capacity we will need procedures to shape the wave-function, in order to imprint information onto it, and the ability to control the linear dynamics of the wave-function. Such an approach may be especially suited to image processing and other applications with image-like data. As usual, parallel computation takes place in linear superposition. Of course, such an approach to quantum computation faces the same problem of quantum decoherence as the qubit approach. On the other hand, since more efficient use is made of the wave-function's representational capacity, it may be possible to accomplish the same purposes with smaller quantum systems.

The definition of computation that I have proposed suggests quite a different way to use the idea of quantum computation, which may be called *quantum-like computation* (QLC). To understand it, observe that in a quantum system the wave-function evolves according to a certain linear differential equation. That is, this differential equation and the wave-function on which it operates are *realized* by the physical quantum system. We can accomplish the same computation by any other physical system that realizes the same mathematical structure. One's immediate reaction might be that the quantum realization is unique, but this is not the case. As we know, physicists routinely simulate the evolution of the wave equation on ordinary digital computers. The simulations are slow and approximate, because they involve the numerical integration of partial differential equations, but the fact that it is done demonstrates that *non-quantum physical systems can realize the mathematics of quantum mechanics*.

The goal of quantum-like computation, then, is to find non-quantum physical systems that obey the equations of quantum mechanics, but proceed sufficiently quickly to be useful and also satisfy the other requirements for practical realizations previously discussed. To this end, it may be convenient to separate the complex wave equation into its real and imaginary parts. The separated parts of the wave-function can be realized by various real-valued physical quantities, such as electrical charge, light intensity, and so forth. Since the system is linear, it will be capable of parallel computation in linear superposition. If nothing else, quantum-like computing may give us the ability to explore the technology of quantum computation in advance of the development of true quantum computers.

VI. Conclusions

In conclusion, let me summarize my points. The historical roots of Church-Turing computation remind us that the theory exists in a *frame of relevance*, which is not well suited to natural computation, nanocomputation, or quantum or quantum-like computation. Therefore we need to supplement it with new models based on different assumptions and suited to answering different questions. Central issues include continuity, indeterminacy, and parallelism. Finally, I argued that once we understand computation in a broader sense than the Church-Turing model, we begin to see new possibilities for using natural processes to achieve our computational goals. These possibilities will increase in importance as we approach the limits of electronic binary logic as a basis for computation. They will also help us to understand the computational processes in the heart of nature.

References

MacLennan, B. J. (1987). Technology-independent Design of Neurocomputers: The Universal Field Computer, in M. Caudill & C. Butler (eds.): *Proc. IEEE First Int'l. Conf. Neural Networks*, Vol. 3, IEEE Press, pp. 39–49.

MacLennan, B. J. (1990). Field Computation: A Theoretical Framework for Massively Parallel Analog Computation, Parts I–IV. Technical Report UT-CS-90-100, Dept. of Comp. Sci., Univ. of Tenn., Knoxville.

MacLennan, B. J. (1999). Field Computation in Natural and Artificial Intelligence, *Information Sciences* **119**: 73–89.

MacLennan, B. J. (2003). Transcending Turing Computability, *Minds and Machines* **13** (1): 3–22.

MacLennan, B. J. (2004). Natural Computation and Non-Turing Models of Computation, *Theoretical Computer Science* **317**, Issues 1–3, pp. 115–145.

Pour-El, M. B. (1974). Abstract Computability and its Relation to the General Purpose Analog Computer (Some Connections Between Logic, Differential Equations and Analog Computers), *Trans. Amer. Math. Soc.* **199**: 1–29.

Rubel, L. A. (1981). A Universal Differential Equation, *Bull. (New Ser.) Amer. Math. Soc.* **4** (3): 345–349.

Rubel, L. A. (1993). The Extended Analog Computer, *Adv. Appl. Math.* **14**: 39–50.

Shannon, C. E. (1941). Mathematical Theory of the Differential Analyzer, *Journ. Math, & Physics of Mass. Inst. Tech.* **20**: 337–354.