To appear in the *International Journal of General Systems*
Vol. 00, No. 00, month 2013, 1–10

Local and global models of physics and computation

Susan Stepney*

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

(*received: xxx*)

Classical computation is essentially local in time, yet some formulations of physics are global in time. Here I examine these differences, and suggest that certain forms of unconventional computation are needed to model physical processes and complex systems. These include certain forms of analogue computing, massively parallel field computing, and self-modifying computations.

Keywords: unconventional computation; Newtonian schema; principle of least action; field computing; complex systems; self-modifying code

1. Introduction

Lloyd (2005) argues that the universe is a giant quantum computer, computing itself. He arrives at this conclusion by focussing on information processing by physical reality. Wharton (2012) argues against this view. His argument takes a different form, and is based on the way certain laws of quantum physics are formulated, and the relationship of this formulation to computation.

Without addressing the argument of whether the universe is or is not a computer (or whether that question even makes sense), here I use Wharton's argument as a starting point for examining certain aspects of various unconventional computational systems.

First, I provide the background of Wharton's argument, as it relates to models of computation, in §2. Then I discuss whether this computational conclusion is valid in the case of conventional (§3) and unconventional (§4) computational models. I then discuss the role of time in unconventional computations, in §5. Finally I discuss the implications for computational systems in §6.

2. Wharton's argument: Newtonian versus Lagrangian mechanics

Here I briefly summarise Wharton (2012)'s argument, focussing on the computational aspects. (The full argument is more to do with models and interpretations of quantum mechanics.) The argument centres around the merits of Newtonian versus Lagrangian mechanics as models of physics. Both these are formulations of

*email: susan.stepney@york.ac.uk

Newton's "System of the World" (Newton, 1728), of differential equation-based laws of motion.

2.1 *The differential formulation*

The first of these formulations uses laws that "specify the rules for how the point that describes the initial conditions in configuration space evolves in time" (Smolin, 2009). Wharton follows Smolin in calling this "the *Newtonian schema*". This Newtonian schema formulation[1] commonly defines the time evolution of the system in question in terms of one or more first order ordinary differential equations (ODEs). This differential formulation is "local" in time, an "instantaneous" formulation: given a state at a particular time t, the equations define the state at the next infinitesimal instant $t + dt$.

Given such a formulation, a natural way to calculate the system path is to follow the time evolution directly: integrate the equations forward in time from the initial conditions (such as the position and velocity at time $t = 0$), in order to determine the final conditions (such as position and velocity as time $t = T$). The previous sentence is cast in mathematical terminology; it can equally well be cast in computational terminology as: iteratively compute from the inputs, to determine the outputs.

2.2 *The integral formulation*

Lagrangian mechanics is an alternative formulation. Its key property for Wharton's argument is its basis on constraints, specifically "the principle of least action"[2], rather than explicitly Newton's laws of motion (which can be derived from the principle). The action $\mathcal{S}$ of a system is the integral of the Lagrangian $\mathcal{L}$ over the path the system takes over time: $\mathcal{S} = \int_0^T \mathcal{L}\, dt$. The action is defined for any potential path. On the path the system actually takes, the action is stationary: $\delta\mathcal{S} = 0$. This integral formulation is "global" in time: it is a property of the entire path, relative to nearby paths.

In this formulation, it is straightforward to put constraints on the system at different times (for example, initial position at $t = 0$ and final position at $t = T$), and then determine the other parameters of the system (for example, initial velocity at $t = 0$). Computationally, the inputs are the *boundary* conditions, which are not necessarily *initial* conditions.

Given such a formulation there is no similar natural way to calculate the system path. We need the entire path in order to calculate the action, and we need a set of paths in order to find the one with minimum (or stationary) action.

2.3 *Equivalent laws?*

These local and global formulations are alternative ways of expressing the same laws of motion. In principle, one formulation can be recast in terms of the other. However, there is more to solving a set of equations than the equations themselves:

[1]Here I give only a thumbnail sketch of the approaches, extracting the essence that affects the computational argument. For a more rigorous discussion, see a good textbook on classical mechanics, such as Goldstein (1980).

[2]More correctly, of *stationary* action.

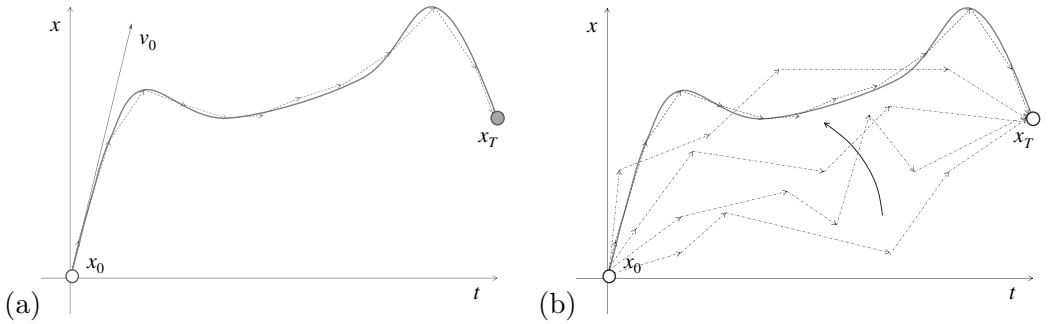

Figure 1. Conceptual view of Newtonian schema computation versus least action computation. (a) The Newtonian schema formulation. The physical path (solid line) is determined by the initial conditions x_0 and v_0, and the equations of motion. It starts at the initial position x_0, evolves over time, and ends at the final position x_T. The path can be computed (dashed arrows) by numerically integrating the ODE-style equations of motion, iteratively progressing along the path to the calculated final position, x_T. (b) The least action formulation. The physical path (solid line) is determined by the boundary conditions x_0 and x_T, and the principle of least action. The path can be computed by calculating various possible paths (dot-dashed arrows), and iteratively minimising the action (represented by the solid curved arrow).

in each specific case we also require the specific boundary conditions. In the Newtonian schema, these are initial conditions. In the least action formulation, however, these conditions can be imposed at different times, and the given initial conditions, considered alone, may thereby under-constrain the problem, leaving them insufficient for a Newtonian schema style solution.

2.4 *Initial conditions may not be the appropriate formulation*

Wharton's argument is that much of the present day problems with quantum mechanics is that it is modelled in Newtonian schema terms, requiring a complete knowledge of initial conditions, from which the final conditions are then deduced. He claims that quantum systems are more naturally modelled in terms of the alternative constraint-formulation style, where the input knowledge of the system can be spread across different times.

2.5 *Computational formulation*

The part of Wharton's argument of interest in this paper is his identification of computation with a Newtonian schema style of calculation: computation comprises iterating a calculation from the initial state and inputs to the final state and outputs (figure 1a). This computational model struggles to simulate systems defined in terms of the constraint of least action: multiple potential paths have to be calculated, and the minimum action one sought (figure 1b).

Wharton then concludes that, since the (quantum) universe is at heart most naturally described in a least action formulation, whereas computation is essentially embedded in the Newtonian schema, that the universe is not a computer. The universe does not determine its least action computationally. (He does not say how it does determine it.)

If computation *is* essentially Newtonian, this is a more immediate issue than philosophical arguments about whether or not the universe is not a computer. There are many least action-like problems that we would like to be able to compute efficiently. For example, consider the optical fourier transform (Goodman, 2005, fig.5.7), where the light from an image placed in the focal plane of a convex lens is passed through the lens, and the image's fourier transform appears on a screen placed in the other focal plane. While the fast fourier transform algorithm (Cooley

& Tukey, 1965) calculates the discrete fourier transform of N points in time $O(N \log N)$ operations, the optical fourier transform (apparently) calculates it in time $O(1)$.

2.6 *Terminology*

In the following discussion, I will for brevity use *Newtonian* and *Lagrangian* to refer to physics and/or computation expressed in the local differential Newtonian schema, and expressed in the global integral Lagrangian least action formulation, respectively.

3. Is computation essentially Newtonian?

Here I discuss Wharton's claim that computation is essentially Newtonian, and the light that investigation of this claim can shed on conventional and unconventional computation.

3.1 *Imperative languages are Newtonian*

The programming languages typically used for solving scientific problems are those such as Fortran, C, C++, and Matlab. These belong to the class of *imperative languages*, that are "based on commands that update variables held in storage" (Watt, 2004, ch.11). A program defines a specific sequence of commands that incrementally change this stored state from the initial state to the final state, and "an imperative ... program may be viewed abstractly as implementing a *mapping* from inputs to outputs" (Watt, 2004, ch.15).

When solving a physical problem cast in the Newtonian schema formulation, this sequence of commands and the associated state changes are direct analogues of the progression of physical time and its associated physical state changes in the problem (figure 1a). Watt (2004, ch.11) comments on the naturalness of this paradigm:

> Many programs are written to model real-world processes affecting real-world entities, and a real-world entity often possesses a state that varies with time. So real-world entities can be modeled naturally by variables, and real-world processes by commands that inspect and update these variables.

Specifically, consider a set of equations of motion specified as first order differential equations in normal form[1] as

$$\frac{dx_i}{dt} = f_i(x_1, \ldots, x_n) \tag{1}$$

where $i = 1 \ldots n$, and with initial conditions given for the x_i at time $t = 0$. We can straightforwardly compute[2] the values of the x_i at time $t = T$ in an imperative language:

[1] Higher order differential equations can be expressed as first order equations by introducing more variables. For example, $F = ma = m\ddot{x}$ can be expressed in normal form with two equations: $\dot{x} = v; \dot{v} = F/m$.

[2] A robust numerical integration would use a more sophisticated numerical method than this. However, this simplistic algorithm illustrates the basic underlying computational principle.

$t := 0$
$\Delta := T * 0.001$
for each $i, x_i := x_i(0)$
while $t < T$ **do**
  for each $i, x_i := x_i + f_i(x_1, \ldots, x_n) * \Delta$
  $t := t + \Delta$
end while

3.2 *Logic languages are more Lagrangian than Newtonian*

When computing the solution of a physical problem cast in the least action formulation, however, there is a "disconnect between algorithmic time and actual time" (Wharton, 2012), and a similar disconnect between program state iterating through multiple potential physical paths, and the single actual physical state of least action (figure 1b).

Not all programming languages are imperative, however. Another class is that of *logic* languages, such as Prolog. Watt (2004, ch.15) describes the different flavour of the logic paradigm:

> A logic program ... implements a *relation*. Since relations are more general than mappings, logic programming is potentially higher-level than imperative ... programming.

In such a language, the program is a set of statements, and the result is the consequence that can be inferred from the statements.

For example, we can define a set of rules such as[1]:

$has\_legs(X, 8)$: - $spider(X)$.
$has\_legs(X, 6)$: - $insect(X)$.
$has\_legs(X, 4)$: - $mammal(X)$.
$insect(archy)$.

From this, we can infer that $has\_legs(archy, 6)$.

In order to define a least action-style calculation, we could include a rule like:

$is\_least\_action(Path)$.

On the face of it then, logic programming languages would seem to be Lagrangian. However, in conventional computation, that is not actually the case.

3.3 *Denotational versus operational semantics*

What I have been talking about so far is different *high-level* programming paradigms: imperative versus logic languages.

Whatever paradigm is used, a program has a *meaning*, such as a mapping from input to output, or a relation between sets of entities. That meaning can be defined in higher- or lower-level ways. See, for example, (Schmidt, 1986, p.3), (Meyer, 1990, ch.4).

The high level, mathematical meaning of the programs, is defined, for example, in a denotational or axiomatic form. *Denotational semantics* can be used to define the meaning of an imperative programming language as the mathematical function that maps its input to its output. *Axiomatic semantics* can be used to define the

[1] A rule such as $has\_legs(X, 8)$: - $spider(X)$ can be read as "X has 8 legs if X is a spider", and $insect(archy)$ can be read as "archy is an insect".

meaning of a logic programming language in terms of axioms and inference rules, which are used to infer the meaning of a program text.

But when one comes to *calculate* an output from an input, or to *infer* the consequences of the program statements, or in other words, to *compute* the meaning, an *operational semantics* is needed. In this case the meaning of the language is defined in terms of the actions of an (abstract) machine interpreting the language statements: it defines an *algorithm* that computes the meaning. In other words, "In an operational semantics we are concerned with *how* to execute programs and not merely with what the results of the execution are." (Nielson & Nielson, 1992, ch.2)

Fisher and Henzinger (2007) describe the difference as:

> A computational model is a formal model whose primary semantics is operational; that is, the model prescribes a sequence of steps or instructions that can be executed by an abstract machine, which can be implemented on a real computer. A mathematical model is a formal model whose primary semantics is denotational; that is, the model describes by equations a relationship between quantities and how they change over time. The equations do not determine an algorithm for solving them ...

In conventional computation at least, operational semantics is "Newtonian": it defines the specific state changes in imperative languages, and it defines a specific algorithm to perform inferencing in logic languages. For the least action computation, the algorithm to infer the path that satisfies $is\_least\_action(Path)$ will be something akin to "calculate the action of all possible $Path$s, and chose the one with least action".

Not all logic language programs execute this inefficiently! This is analogous to an operational semantics for $is\_sorted(List)$ being "calculate all possible permutations of $List$, and choose the sorted one". In the case of sorting, there are *much* more efficient algorithms, which can be provided by the programmer or by a clever optimising compiler. In the case of calculating the least action, there may not be a much more efficient algorithm, however.

So we seem to be back to an underlying computation akin to figure 1b.

4. Can computation have a Lagrangian operational semantics?

4.1 *The reason for Newtonian operational semantics*

Wharton's argument seems to boil down to: the operational semantics of any computer is necessarily Newtonian. But just why is operational semantics Newtonian? If (some) physics may be better formulated as Lagrangian, and given that the execution of the computation occurs in a physical device, why could that execution not also be Lagrangian? Is the Newtonian schema a fundamental property of computation *per se*, or only of our current implementations?

Standard processors are built to the so-called von Neumann architecture, with its "fetch-execute-store" cycle and stored program paradigm. This architecture is essentially iterative and state based, and so languages with imperative operational semantics map naturally onto it.

The Turing model of computation as implemented by the von Neumann architecture is itself essentially algorithmic, since it was originally designed to formalise the real world procedures followed by human "computers": clerks carrying out calculations (Copeland, 2008).

4.2 *Alternative operational semantics*

The von Neumann architecture is not the only computational architecture, merely the predominant one. Other unconventional computational architectures exist: for example, neural networks, cellular automata (the other von Neumann architecture!), microfluidic processors, reaction-diffusion chemical computing, optical computing, quantum computing (with a range of associated architectures including the circuit model, and measurement based systems), and more.

The question raised by Wharton's argument is, can we have an (unconventional?) operational constraint-based computational semantics? Can we make our computations more physics-like? Yes. We can take those very physical processes that Wharton argues do not act computationally in the Newtonian schema sense, and use them as unconventional computational devices, through analogue computing. We simply need to find a physical process that is a suitable analogue of the computation we wish to do, and that we can configure as (part of) a computational device. To use a physical system as a computational device in this manner requires that the underlying physical model is sufficiently well-characterised that we know what computation it is performing (Horsman, Stepney, Wagner, & Kendon, 2014).

5. The role of time

5.1 *Atemporal models*

One question that arises from the Lagrangian formulation, with its constraints at times other than the initial time (the "now"), is: how does light "know" which is the shortest path that it should take? how does the system "know" to achieve the boundary conditions at other times? how does it arrange itself to achieve stationary action, a quantity that requires "knowledge" of future times?

To avoid the lingering teleological odour of such questions, some scientists have suggested atemporal models of physics. From the atemporal space-time of Einstein's relativity, to more subtle arguments (Barbour, 1999), these resolve the issue by removing time as a dynamic process from the picture.

Not all authors are in favour of eliminating time as something real. For example, when Smolin (2009) talks of "the misapplication of the Newtonian schema to the universe as a whole", he does not mean that he is against applying it to *parts* of the universe. Indeed, he lays out various principles for the reality of time, one of which is:

> Everything that is real in a moment is a process of change leading to the next or future moments. ... Things that persist must be thought of as processes leading to newly changed processes. An atom in a moment is a process leading to a different or a changed atom in the next moment.

In this view, time is real, and things really do progress through time.

If time is real, are we forced back to teleological-sounding explanations? One way of examining how light travels along the shortest path is to think of it in terms of waves, reinforcing and cancelling at different points in space. In this approach, the physical system is not a single localised particle, but a spatially extended field. Computation of its behaviour does not then need to follow only a single point in space, but the behaviour of all of space. So maybe it is not the temporality of classical computation that is the issue, but its spatial *zero-dimensionality.* All classical computation is pushed through the single point that is the von Neumann bottleneck.

Generalisations of these kind of ideas, of thinking of fields rather than particles, lead to such unconventional computational architectures as cellular automata, field computing (MacLennan, 1999), and environment orientation (Hoverd & Stepney, 2009, 2011).

5.2 *Energy minimisation*

Another process we see in physical systems, akin to action minimisation, is (free) energy minimisation. For example: a ball rolls downhill to find its lowest potential energy; the protein folding problem can be cast as a free energy minimisation problem (Anfinsen, 1973). Here, however, the minimisation is definitely a temporal process: the system moves from a high energy to a lower energy state, "seeking" the minimum energy state: the ball *rolls downhill* to its minimum potential energy state, the protein *folds* from an unfolded form. Contrast this with the atemporal view of the bent light crossing between two media: there was never some unbent beam that over time bent itself to the correct angle, minimising its travel time.

There is no guarantee of success, either: the system may get caught in a local minimum, where its (free) energy is merely stationary, not minimal. Adding a thermal component may allow the system to "jiggle" out of a local minimum. This is the basis of the simulated annealing algorithm (Kirkpatrick, Gelatt Jr., & Vecchi, 1983), an optimisation algorithm that equates some cost function to be minimised with a physical energy, and has an artificial temperature that slowly cools, by analogy to the physical annealing process.

However, even when we formulate algorithms that can follow the temporal process, such as the slow cooling in simulated annealing, or the gradual folding of a protein, it still seems that we are having to do a lot of computational work for our result, where the physics just "happens" without explicit computation. For example, we have to explicitly calculate the energy analogues and their differences, and the related Boltzmann probabilities. The "protein folding problem" is so called because we cannot yet compute the relevant physical processes efficiently.

It appears that fields provide a more natural model of some of these processes, such as the electric fields around amino acids in proteins. Alternatively, the Boltzmann distribution at the heart of simulated annealing arises in physical systems as a statistical distribution of the energies a large number of particles. Physicists use similar statistical notions, such as a density matrix approach, to connect the deterministic macroscopic world of observation with the possibly non-deterministic underlying physical system. In the case of both the field approach, and the statistical distribution approach, it is the zero-dimensionality of classical computation where the problem lies.

5.3 *Complex Systems and Growth*

Complex systems are fundamentally temporal. They can depend critically on their detailed history (Bak & Paczuski, 1995), and are intrinsically irreversible. Because of this essential contingency, there may be no sensible formulation of their dynamics in terms of boundary conditions at the "end point" of their evolution. So computations that are temporal are a natural fit to these systems. But again, since an essential quality of complex systems is that they contain a large number of agents and processes leading to emergent properties (Anderson, 1972), the zero-dimensionality of classical computation is a severe bottleneck to their computation.

One important sub-class of complex systems is one where the number of compo-

nents changes and grows as the system develops, for example the continual novelty expressed by open-ended evolution. Here the very dimensionality of the system is changing, and mathematical models struggle to keep up with the dynamic introduction of new dimensions, and novel kinds of dimensions, in the overall state space (Stepney, 2012a, §4). The system's growing, changing components do not behave as a statistical ensemble. This time computation comes to our rescue (Stepney, 2012b). Coping with extra dimensions is straightforward: malloc(n) and new Obj(p) allocate new memory, increasing the dimensionality of the computational state space. Coping with novel kinds of dimensions, not anticipated at coding time, requires self-modifying code (Stepney & Hoverd, 2011). This is yet another move away from the underlying constraints of classical computation, with its careful separation of code and data, to an unconventional formulation.

6. Conclusion

Wharton (2012) admonishes us:

> Now there's one last anthropocentric attitude that needs to go, the idea that the computations we perform are the same computations performed by the universe ...

I would suggest that there is another attitude that needs to go: that the only computational model is the classical, imperative, "Newtonian" one. Instead, we can look to the physical and biological worlds, see where they perform better, or just differently, from classical computation, and use that as inspiration for developing novel forms of unconventional computation.

Another anthropocentric attitude that needs to go is the idea that the computations our physical computers can perform are restricted to being the same as the calculations we can perform with pen-and-paper.

Acknowledgements

My thanks to Angelika Sebald for highlighting the important distinction between statistical ensembles and growing systems.

This work was partly funded by the EU FP7 FET Coordination Activity TRUCE (Training and Research in Unconventional Computation in Europe), project reference number 318235.

References

Anderson, P. W. (1972). More is different. *Science*, *177*, 393-396.

Anfinsen, C. B. (1973). Principles that govern the folding of protein chains. *Science*, *181*(4096), 223-230.

Bak, P., & Paczuski, M. (1995). Complexity, contingency, and criticality. *Proc. Natl. Acad. Sci. U. S. A.*, *92*, 6689-6696.

Barbour, J. B. (1999). *The end of time: The next revolution in physics*. Oxford University Press.

Cooley, J. W., & Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, *19*, 297–297.

Copeland, B. J. (2008). The modern history of computing. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy* (Fall 2008 ed.). (Archived online:

http://plato.stanford.edu/archives/fall2008/entries/computing-history/, accessed 29 July 2013)

Fisher, J., & Henzinger, T. A. (2007). Executable cell biology. *Nature Biotechnology*, *25*(11), 1239–49.

Goldstein, H. (1980). *Classical mechanics* (2nd ed.). Addison-Wesley.

Goodman, J. W. (2005). *Introduction to Fourier optics* (3rd ed.). Roberts & Company.

Horsman, C., Stepney, S., Wagner, R. C., & Kendon, V. (2014). When does a physical system compute? *(forthcoming)*.

Hoverd, T., & Stepney, S. (2009). Environment orientation: an architecture for simulating complex systems. In *CoSMoS 2009, York, UK* (p. 67-82). Luniver Press.

Hoverd, T., & Stepney, S. (2011). Energy as a driver of diversity in open-ended evolution. In *ECAL 2011, Paris, France* (p. 356-363). MIT Press.

Kirkpatrick, S., Gelatt Jr., C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671-680.

Lloyd, S. (2005). *Programming the universe*. Vintage.

MacLennan, B. J. (1999). *Field computation in natural and artificial intelligence, extended version* (Tech. Rep. No. UT-CS-99-422). Computer Science Department, University of Tennessee, Knoxville.

Meyer, B. (1990). *Introduction to the theory of programming languages*. Prentice Hall.

Newton, I. (1728). *Principia book 3: The system of the world*.

Nielson, H. R., & Nielson, F. (1992). *Semantics with applications*. Wiley.

Schmidt, D. A. (1986). *Denotational semantics*. Wm. C. Brown.

Smolin, L. (2009, June). The unique universe: Against the timeless multiverse. *Physics World*, 21-26.

Stepney, S. (2012a). Non-Classical computation : a dynamical systems perspective. In G. Rozenberg, T. Bäck, & J. N. Kok (Eds.), *Handbook of natural computing* (pp. 1979–2025). Springer.

Stepney, S. (2012b). Programming unconventional computers: Dynamics, development, self-reference. *Entropy*, *14*, 1939–1952.

Stepney, S., & Hoverd, T. (2011). Reflecting on open-ended evolution. In *ECAL 2011, Paris, France* (pp. 781–788). MIT Press.

Watt, D. A. (2004). *Programming language design concepts*. Wiley.

Wharton, K. (2012). The universe is not a computer. arXiv:1211.7081v1.