Non-Classical Hypercomputation

SUSAN STEPNEY

Dept. Computer Science, University of York, YO10 5DD, UK

Received 30 Jan 2007; in final form 16 Apr 2007

Hypercomputation that seeks to solve the Halting Problem, or to compute Turing-uncomputable numbers, might be called "classical" hypercomputation, as it moves beyond the classical Turing computational paradigm. There are further computational paradigms that we might seek to move beyond, forming the basis for a wider "non-classical" hypercomputation. This paper surveys those paradigms, and poses various non-classical hypercomputation questions.

1 INTRODUCTION

Today's computing, *classical* computing, is an extraordinary success story. However, there is a growing appreciation that it encompasses an extremely small subset of all computational possibilities. A variety of paradigms encompass classical computing, and their assumptions need to be carefully scrutinised. The UKCRC's Grand Challenge exercise [1] includes the Grand Challenge of Non-Classical Computation (GC-7) [1, 2, 30, 31], whose task it is to challenge and move beyond the various classical computational paradigms, thereby broadening and enriching the subject area.

GC-7 identifies and challenges the classical paradigms [30], of which the Turing paradigm is arguably the most prominent one. The Turing paradigm speaks in terms of Turing machines, Turing-computability, Halting, and so on. Hypercomputation traditionally seeks to solve the Halting Problem, or to compute Turing-uncomputable numbers. Let us call this "classical" hypercomputation, as it seeks to move beyond the most prominent classical computational paradigm.

In [30] we identify several further computational paradigms that may be challenged, in addition to the Turing paradigm. The rest of this paper discusses how classical hypercomputation seeks to move beyond the Turing paradigm, and the various forms of "non-classical" hypercomputation that could seek to move beyond other paradigms.

2 CLASSICAL HYPERCOMPUTATION

2.1 Physics matters

Classically, Turing computability is an abstract mathematical notion, independent of the laws of physics.

However, that notion assumes the classical Newtonian physical laws. As Deutsch [12] neatly sums it up:

> "Turing hoped that his abstracted-paper-tape model was so simple, so transparent and well defined, that it would not depend on any assumptions about physics that could conceivably be falsified, and therefore that it could become the basis of an abstract theory of computation that was independent of the underlying physics. 'He thought,' as Feynman once put it, 'that he understood paper.' But he was mistaken. Real, quantum-mechanical paper is wildly different from the abstract stuff that the Turing machine uses. The Turing machine is entirely classical"

Quantum computers have already shown that algorithmic *feasibility* depends on the laws of physics. Exploitation of quantum superposition makes a difference to computational efficiency. (Some authors refer to this as "super-Turing computation": faster, but not different.) Certain quantum random walk algorithms are exponentially faster than their classical counterparts; see [18] for an overview. Shor's quantum factorisation algorithm is polynomial time complexity [27] whereas the best known classical one is essentially exponential (but it is not known if there is a polynomial time classical algorithm). Other quantum effects can lead to results simply not possible classically. For example wavefunction "collapse" makes genuine random number generation possible [3], and quantum entanglement can be exploited to achieve untappable communication channels, dense coding, and information teleportation [6]. It is not all gain, however: quantum information cannot be cloned [37], leading to interesting problems in developing quantum error-correcting codes [8], among other things.

To emphasise further the critical importance of the underlying physical laws, it has been shown that the deep conundrum of whether **P = NP** (with definitions suitably extended to quantum computation) would be answerable, in the affirmative, *if the laws of quantum mechanics were non-linear* [4] (which, currently, they are not thought to be). The fact that the precise form of the laws of physics have an impact on what is classically thought to be a purely *mathematical* question is considerable food for thought.

We know some of the consequences on feasibility of considering computation in a world that includes quantum mechanics. Does what is *computable* similarly depend on the non-classical laws of physics? What are the consequences for computability in a world that includes special and general relativity? Certain uses of general relativistic devices appear to be able to solve the Halting Problem, by performing infinite Turing computation in the computer's rest frame, in a finite elapsed time in the observer's frame (see for example, [15]).

When we move from the mathematical to the physical domain to analyse our computations, we need to take into account more infrastructure: the same laws of physics that provide more computational power also constrain our ability to measure the outputs of our new exotic devices. So, even if we can design classical hypercomputers on paper, we may not be able to build them. But this should not worry us, for we can't build a Turing machine either. Unbounded memory is unphysical. Even if classical hypercomputers don't, or can't, exist, their study can enrich our understanding of what classical Turing computation is, what its limits are, and why.

2.2 Challenging the questions

The Turing machine is based on concepts consistent with Newtonian physical laws, and on "disembodied" mathematical abstractions that, for example, require unbounded memory resources and ignore power consumption. But the world is not Newtonian, and all computation is physically embodied in devices whose behaviour cannot be completely captured by a closed mathematical model. The mathematical model of the Turing Machine is not an adequate model for all notions of computation.

Classical hypercomputation goes some way to extending the model of computation. However, it goes only so far: it seeks different *answers*, but does not really challenge the *questions*; in some sense it still plays by the "Turing rules" [22]. The insolubility of the Halting Problem is a classical limit, and classical hypercomputation seeks to solve it (or seeks to find an uncomputable number, often to use that as an oracle to solve the Halting Problem). But what if we challenge the very basis of this problem? Is the Halting Problem of any relevance or interest to our particular non-classical domain of choice? In fact, is halting of interest? What about the other questions and assumptions underlying the Turing paradigm, and other computational paradigms?

3 NON-CLASSICAL HYPERCOMPUTATION

This section examines some further traditional computational paradigms, and hence the various forms of non-classical hypercomputation that would move beyond them.

3.1 21st century science

Classical hypercomputation exploits quantum theory, and special and general relativity. These theories are now a century old. We could say that classical hypercomputation has moved the physical basis of computation from the 17th century into the early 20th century.

Science itself has moved on in the meantime. Although the underlying theories are now a century old, some of their stranger and more powerful consequences are still being worked out[*]. Notions from Quantum Information Theory are having profound effects on the way we understand even classical information. And further scientific advances are being made. What will be the properties of, say, a string theory computer, or loop quantum gravity computer (assuming that these theories stand the test of time)?

Once the importance of the underlying laws of physics in relation to models of computation is appreciated, one can start playing with counterfactual physical systems, and examining the effect they have on computation (for example, the case of non-linear quantum mechanics and the **P = NP** question cited earlier). What is fascinating is how little "wiggle room" there seems to be in this relationship: change the laws of physics too much, and get silly computational possibilities (such as one bit seemingly encoding too much information); relax the computational constraints too much, and get acausal physical laws. See for example, [7]. Wheeler [34] speculates that computation is more fundamental than physics in his "It from bit" programme.

So, some questions for physical hypercomputation are:

> *Which are more fundamental, the laws of computation, or the laws of physics? Do the laws of computation constrain those*

[*] As Bob Coecke puts it [private communication], it wasn't until the Bell inequalities had been demonstrated in the laboratory that quantum mechanical weirdness stopped being a bug, and became a feature.

of physics, or vice versa? Or are they two sides of the same question? Or, despite the seeming mutual constraints, are they in fact independent?

3.2 Substrate as implementation detail *v.* embodiment

Classically, computation is a mathematical abstraction, independent of the underlying implementation. The substrate is mere implementation detail.

In practice, all computation is embodied. A computation, being embodied, takes time to execute, and consumes power as it executes, usually in a data-dependent manner[†] . These time and power consumption *side-channels* can be measured and analysed. Such analyses have been used to attack security mechanisms, for example [19, 20]. Side channels are outside the classical mathematical model of computation. Even if the known channels are explicitly modelled, the world is open, and further side channels always exist: no mathematical model of the world is complete, and further attacks are always possible [10]. As we broaden the base of physical laws on which we base our computations, more side-channels will become apparent. Side-channels may themselves become rich computational resources.

Classical computation is abstracted away from the implementation substrate precisely to make it substrate-independent. The computational virtual machine can be implemented on any suitable embodying substrate, no matter how 'unnatural' that implementation is for the substrate (for example, analog transistors being run saturated to make them act as digital switches). This leads to certain operations being more difficult than they need to be. For example, operations on real numbers could be implemented directly by certain substrates, rather than as in a real number virtual machine running on top of the computational virtual machine[‡] . The classical need for (at least) two levels of virtual machines results in real number computations being classically "inefficient".

Direct analog implementation of many computations, not only real number calculations, can be more efficient (in terms of speed, power consumption, and substrate area) than their classical discrete counterparts. For this reason

[†] Landauer [21] argues that it is *irreversible* computations that necessarily require energy, and generate entropy. Reversible computations might not necessarily do so, in the adiabatic limit. Today's and tomorrow's devices, however, consume noticeable power as they process.

[‡] I am not referring here to the infinite-precision real numbers of mathematics, which are sometimes employed to achieve a form of theoretical analog hypercomputation. I am referring to "physical" real numbers (*real* real numbers, if you like): the measurable values of physical quantities, the kind of number we want to manipulate in computations that relate to (possibly other) physical quantities.

they are of great interest in domains where power and weight are at a premium: embedded systems, particularly in spacecraft.

Direct implementation in a physical substrate constrains what kind of computations can be supported "naturally": ideas of universality may have to be abandoned for particular implementations. However, direct implementation may enable certain other kinds of computations, by letting the substrate "naturally" perform them [9, 17, 29].

So, some questions for substrate hypercomputation are:

> *What are the computational constraints imposed by physical embodiment? What computational tasks can be offloaded onto the physical embodiment, easing the computational burden of the program?*

Classical computation is performed by artefacts that are specifically designed to implement the relevant rules.

Biological systems are increasing being interpreted as information processing agents. Yet they do not fit into the classical framework: they are essentially embodied, a hybrid of continuous and discrete components, noisy, evolved not designed, and it is not clear what "program" they are in fact "executing" (if any).

As discussed earlier, there are computational constraints imposed by both logical information processing, and by physical embodiment. These two sets of constraints come together in embodied biological systems.

So, some questions for biological hypercomputation are:

> *What are the emergent constraints when logical and physical computation are combined in biological systems and complex embodied computational artefacts (robots, and beyond)?*

3.3 Sequential *v.* parallel

Classically, a (finitely) parallel computer is no more powerful than a sequential one: the computation it does can always be transformed into an equivalent sequential one of the same computational complexity.

In practice, all computation is embodied, and interacts with a real world in real time. Even though classical Turing computation has no notion of real time, it still appreciates this fact: exponential-time algorithms are infeasible because they *take too long to execute*.

So all computations interact with the real world, from the classical Turing level of providing an answer to a question, through an embedded real time control system in a physical machine, to ants following evaporating pheromone trails. The real world acts on it own timescales. Computation needs to take these timescales into consideration: parallelism may be the only feasible way to achieve real time response.

Classical models of computation are inherently sequential, and parallelism is considered "difficult". What we seem to have done is take the inherently massively parallel real world, forcibly sequentialise it into some computational model, then grudgingly add a little bit of thread-spaghetti back in again. Trained computer scientists tend to find the sequential model more "natural" than the parallel one. This is partly because they have forgotten how unnatural it actually is (except maybe for the few who have to teach introductory programming), and partly because the threads model of parallelism *is* unnatural, compared to the more agent-like flavour of real world parallelism. In [31], Welch suggests that CSP-based programming languages provide a more natural model of concurrency.

So, some questions for parallel hypercomputation are:

> *What are the natural computational primitives for parallel real-time embedded and embodied computation? What can we learn from the physical world to reduce, or remove, classically "difficult" parallel problems of communication, synchronisation, livelock, deadlock, etc?*

3.4 Algorithmic *v.* interactive

Classically, a computation evaluates a mathematical function, executing an algorithm to map the initial input to the final output, ignoring the external world while it executes.

Sloman [28] dubs these "ballistic" calculations: the system gets all the information it needs at the start of the calculation, then launches itself into the blue, performing the calculation with no further input or feedback from the external world. Sloman, interested in AI applications, notes that ballistic calculations do not suit adaptive, reactive intelligence. Ballistic path planning is a hard task in a static environment; it is impossible in a dynamic one.

Wegner [33] discusses interaction machines: "*Turing machines extended by addition of input and output actions that support dynamic interaction with an external environment*", and claims that they are more powerful than Turing machines. That is, he claims that they are hypercomputers¶. Since all

¶ I choose to classify this as a claim of non-classical hypercomputation, because, although the claim is about Turing machines, it is not based on one of the more conventional challenges to the assumptions underlying the Turing paradigm.

realised computers are, of course, interaction machines, this would mean that hypercomputers are already with us. Notwithstanding the details of this claim, interaction machines certainly provide a more natural model of most computational tasks.

Process algebras such as CSP [14], CCS [23], and π-calculus [24] support descriptions of interaction machines, since they include descriptions of engaging in events with the environment. Wegner notes that interaction machines are a natural interpretation of object-oriented paradigms. Combinations of process algebras and stateful systems, such as Circus [36, 26], can be used to specify object-oriented systems with communication between objects, and state within objects.

So, some questions for interactive hypercomputation are:

> *What extra possibilities do interaction machines allow? What is a natural manner to specify and describe object orientation? Is object orientation in fact a natural way to describe interactive systems? What is the relationship between communication (between objects) and state (within objects)?*

3.5 Program development

Classical program development assumes one has an abstract functional specification, and proceeds to an executable implementation by a process of refinement [13, 32].

Reality is rarely so accommodating, even classically. Functional refinement is not sufficient to maintain non-functional properties, such as safety and security [16]. It can also be overconstraining, unacceptably limiting the degree of abstraction possible [35]. Our software development process might not fit into a classical refinement framework, for example, if we are using an evolutionary or other bio-inspired algorithm to develop our system.

So, a question for development process hypercomputation is:

> *How can bio-inspired, inexact, "soft", algorithms be integrated into a program-development process?*

Emergent systems in particular do not exhibit a well defined link between implementation (the low level local agent behaviours) and the specification (the high level global system behaviour). The emergent behaviour cannot be captured by a refinement relation [25], and the relationship need not be exact.

Solid-state, or condensed matter, physics teaches us that *more is different* [5], and that new higher-level laws of physics "emerge" from large collections of particles. How might these ideas and results give new higher-level concepts of computation and information?

So, some questions for emergent hypercomputation are:

> *What theories of emergent computation can be built on "non-elementary" laws of physics, such as condensed matter? What other laws of physics can support corresponding theories of computation?*

And so we come full circle back to the relationship between natural laws and computation. Trying to solve the Halting Problem is an important part, but only a part, of a full theory of classical and non-classical hypercomputation.

ACKNOWLEDGEMENTS

I would like to thank Sam Braunstein and Fiona Polack, whose comments on an earlier draft have helped correct and clarify some of the points made here.

REFERENCES

[1] UKCRC Grand Challenge website: http://www.ukcrc.org.uk/grand_challenges/index.cfm.

[2] GC in Non-Classical Computation website: http://www.cs.york.ac.uk/nature/gc7/.

[3] Quantum random number download website: http://www.randomnumbers.info/.

[4] D. S. Abrams and S. Lloyd. (1998). Nonlinear quantum mechanics implies polynomial-time solution for NP-complete and #P problems. *Phys. Rev. Lett.*, 81:3992–3995.

[5] P. W. Anderson. (1972). More is different. *Science*, 177:293–296.

[6] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters. (1993). Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Phys. Rev. Lett.*, 70:1895–1899.

[7] A. Cabello. (2005). Communication complexity as a principle of quantum mechanics. In C. S. Calude, M. J. Dinneen, G. Păun, M. J. Pérez-Jiménez, and G. Rozenberg, editors, *Unconventional Computation 2005*, volume 3699 of *LNCS*, pages 70–81. Springer.

[8] A. R. Calderbank and P. W. Shor. (1996). Good quantum error-correcting codes exist. *Phys. Rev, A.*, 54(2):1098–1105.

[9] A. Clark. (1997). *Being There: putting brain, body and world together again*. Oxford University Press.

[10] J. A. Clark, S. Stepney, and H. Chivers. (2005). Breaking the model: finalisation and a taxonomy of security attacks. In Derrick and Boiten [11], pages 225–242.

[11] J. Derrick and E. Boiten, editors. (2005). *REFINE 2005 Workshop, Guildford, UK, April 2005*, volume 137(2) of *ENTCS*. Elsevier.

[12] D. Deutsch. (1997). *The Fabric of Reality*. Penguin.

[13] He J., C. A. R. Hoare, and J. W. Sanders. (1986). Data refinement refined (resumé). In *ESOP'89*, volume 213 of *LNCS*, pages 187–196. Springer.

[14] C. A. R. Hoare. (1985). *Communicating Sequential Processes*. Prentice Hall.

[15] M. Hogarth. (1992). Does General Relativity allow an observer to view an eternity in a finite time. *Foundations of Physics Letters*, 5:73–81.

[16] J. L. Jacob. (1992). Basic theorems about security. *J. Computer Security*, 1:385–411.

[17] J. A. S. Kelso. (1995). *Dynamic Patterns: the self-organization of brain and behavior*. MIT Press.

[18] J. Kempe. (2003). Quantum random walks: an introductory overview. *Contemporary Physics*, 44(4):307–327.

[19] P. Kocher. (1996). Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Crypto '96*, volume 1109 of *LNCS*. Springer.

[20] P. Kocher, J. Jaffe, and B. Jun. (1999). Differential power analysis. In *Crypto '96*, volume 1666 of *LNCS*. Springer.

[21] R. Landauer. (1961). Irreversibility and heat generaiton in the computing process. *IBM Journal od Research and Development*, 5(3):183–191.

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

[23] R. Milner. (1980). *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer.

[24] R. Milner, J. Parrow, and D. Walker. (1992). A calculus of mobile processes. *Information and Computation*, 100(1):1–77.

[25] F. Polack and S. Stepney. (2005). Emergent properties do not refine. In Derrick and Boiten [11], pages 163–181.

[26] T. L. V. L. Santos, A. L. C. Cavalcanti, and A. C. A. Sampaio. (2006). Object Orientation in the UTP. In S. Dunne and B. Stoddart, editors, *Unifying Theories of Programming*, volume 4010 of *LNCS*, pages 18–37. Springer.

[27] P. W. Shor. (1997). Polynomial-time algorithms for prime number factorization and discrete logarithms on a quantum computer. *SIAM J. Comp.*, 26(5):1484–1509.

[28] A. Sloman. (2002). The irrelevance of Turing Machines to AI. In M. Scheutz, editor, *Computationalism: New Directions*, pages 87–127. MIT Press.

[29] S. Stepney. (2007). Embodiment. In D. Flower and J. Timmis, editors, *In Silico Immunology*, chapter 12, pages 265–288. Springer.

[30] S. Stepney, S. L. Braunstein, J. A. Clark, A. Tyrrell, A. Adamatzky, R. E. Smith, T. Addis, C. Johnson, J. Timmis, P. Welch, R. Milner, and D. Partridge. (2005). Journeys in non-classical computation I: A grand challenge for computing research. *Int. J. Parallel, Emergent and Distributed Systems*, 20(1):5–19.

[31] S. Stepney, S. L. Braunstein, J. A. Clark, A. Tyrrell, A. Adamatzky, R. E. Smith, T. Addis, C. Johnson, J. Timmis, P. Welch, R. Milner, and D. Partridge. (2006). Journeys in non-classical computation II: Initial journeys and waypoints. *Int. J. Parallel, Emergent and Distributed Systems*, 21(2):97–125.

[32] S. Stepney, D. Cooper, and J. Woodcock. (1998). More powerful Z data refinement: pushing the state of the art in industrial refinement. In J. P. Bowen, A. Fett, and M. G. Hinchey, editors, *11th International Conference of Z Users, Berlin, Germany, September 1998*, volume 1493 of *LNCS*, pages 284–307. Springer.

[33] P. Wegner. (1997). Why interaction is more powerful than algorithms. *Comm. ACM*, 40(5):81–91.

[34] J. A. Wheeler. (1990). Information, physics, quantum: The search for links. In W. H. Zurek, editor, *Complexity, Entropy and the Physics of Information*. Addison Wesley.

[35] J. Woodcock, S. Stepney, and D. Cooper. (2007). Mondex: “real is different”. *BCS FACS*. (submitted).

[36] J. C. P. Woodcock and A. L. C. Cavalcanti. (2002). The semantics of Circus. In D. Bert, J. P. Bowen, M. C. Henson, and K. Robinson, editors, *ZB 2002*, volume 2272 of *LNCS*, pages 184–203. Springer.

[37] W. K. Wootters and W. H. Zurek. (1982). A single quantum cannot be cloned. *Nature*, 299:802–803.