Co-designing the computational model and the computing substrate

(invited paper)

Susan Stepney

Department of Computer Science, University of York, UK
York Cross-disciplinary Centre for Systems Analysis, University of York, UK

Abstract. Given a proposed unconventional computing substrate, we can ask: Does it actually compute? If so, how well does it compute? Can it be made to compute better? Given a proposed unconventional computational model we can ask: How powerful is the model? Can it be implemented in a substrate? How faithfully or efficiently can it be implemented? Given complete freedom in the choice of model and substrate, we can ask: Can we co-design a model and substrate to work well together?

Here I propose an approach to posing and answering these questions, building on an existing definition of physical computing and framework for characterising the computing properties of given substrates.

1 Introduction

There are many proposed unconventional computational models: reservoir computing, general purpose analogue computing, membrane computing, reaction-diffusion computing, quantum computing, morphogenetic computing, and more. There are just as many proposed unconventional computing substrates: slime moulds, carbon nanotubes, gene engineered bacteria, gold nanoparticle networks, memristors, optical systems, and more. But how to match model and substrate to get an effective unconventional computer?

In order to tackle this question, I work through several stages. I describe one definition of what is meant by *physical computing*, to distinguish a system that is computing from one which is just 'doing its thing' (§2). I describe a method for determining how well a given substrate performs as a physical computer implementing some computational model (§3). I then discuss how this method could be adapted to determine how well a given computational model captures the computing performed by some substrate (§4), and then how these approaches might be combined to co-design model and substrate (§5).

2 When does a physical system compute?

2.1 Abstraction/Representation theory

It is necessary to be able to determine when a substrate is specifically computing, as opposed to merely undergoing the physical processes of that substrate, before we can determine how well it is doing so.

To address the question of when a physical system is computing, we use *abstraction/representation* theory (AR theory) [14, 15, 16, 17, 35], in which science, engineering, and computing are defined as a form of *representational activity*, requiring the use of a 'representation relation' to link a physical system and an abstract model in order to define their operation. We use AR theory to distinguish scientific experimentation on a novel substrate from the performance of computation by that substrate.

The compute cycle is shown in figure 1. An abstract problem A is encoded into the computational model as $m_{\mathbf{p}}$; abstractly the computation C produces $m'_{\mathbf{p}}$; this is decoded as the solution to the problem, A'. To implement this abstract computation, the encoded model is instantiated into the physical computer state $\mathbf{p}$; the computer calculates via $\mathbf{H}(\mathbf{p})$, evolving into physical state $\mathbf{p}'$; the final state is represented as the final abstract model $m_{\mathbf{p}'}$.

For the abstract computation to have been correctly physically computed, we require $m_{\mathbf{p}'} \simeq m'_{\mathbf{p}}$ (where how close the approximate equality needs to be is device- or problem-dependent). Ensuring this equality holds is a process of debugging the physical system, including how it is instantiated (engineered, programmed and provided with input data), and how its output is represented. Then we say that the initial instantiation, physical evolution, and final representation together implement the desired abstract computation $C(m_{\mathbf{p}})$.

From this model, Horsman et al [14] define computing as *the use of a physical system to predict the outcome of an abstract evolution*.

2.2 Example: AR theory applied to Reservoir Computing

In order to demonstrate how AR theory can be applied to unconventional computing, we here outline its use applied to reservoir computing (RC) *in materio* [6]. RC is a popular choice of model for many unconventional substrates, because it treats the substrate as a black box. The stages of AR theory specialised to RC with a carbon nanotube substrate are:

- computational model: a formal model of reservoir computing, such as the Echo State Network model [18].
- instantiation:
  - the substrate is engineered to be an RC: here the physical RC is a blob of carbon nanotubes in polymer, deposited on an electrode array
  - the substrate is configured (programmed) to be an RC suitable for a particular task: here by applying voltages to a subset of the electrodes; which voltages to apply to which electrodes for a particular task are typically evolved during the programming phase

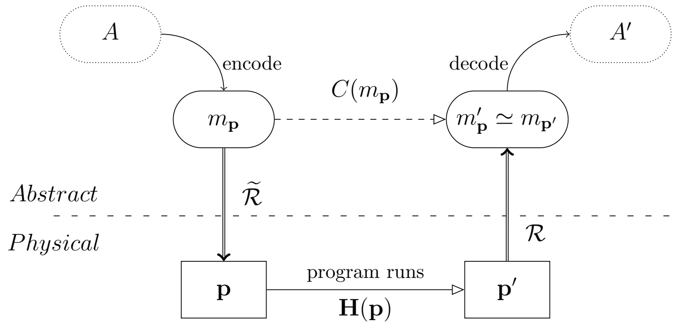

Fig. 1. Physical computing in AR theory. The full compute cycle, starting from initial abstract problem A, encoding in terms of an abstract computational model, instantiation into a physical computer $\mathbf{p}$, physical evolution of the device, followed by representation of the final physical state, then decoding to the abstract answer to the problem A'.

  - the substrate is provided input data, through another subset of electrodes
- physical substrate evolution: given the input configuration (program plus data) the physical system evolves under the laws of physics, producing an output
- representation: the output is extracted as voltages from a further subset of the electrodes, then passed through the RC output filter (trained during the programming phase) here implemented in a PC.

This description makes it clear that there is computation done in both the instantiation stage and the representation stage, in addition to the physical compute stage. Here the computing in the instantiation stage occurs during the 'programming' substage; this effort can be amortised over many runs on specific data, much as how classical computing ignores the cost of compilation and testing when analysing the computational resource costs of algorithms. The computing performed in the representation stage here, however, of processing with the output filter, occurs for each run, and so needs to be included in the overall cost of the computation.

Such instantiation and representation costs are one way to 'hide' the true cost of unconventional computation [3], and care must be taken to analyse these fully, in addition to the use of other unconventional computing resources [2].

3 How well does a physical system compute?

3.1 How well does a carbon nanotube reservoir compute?

AR theory defines *when* a substrate is computing with respect to a model. We have employed this to developed CHARC, a method to characterise *how well* some material substrate can instantiate and implement a given computational

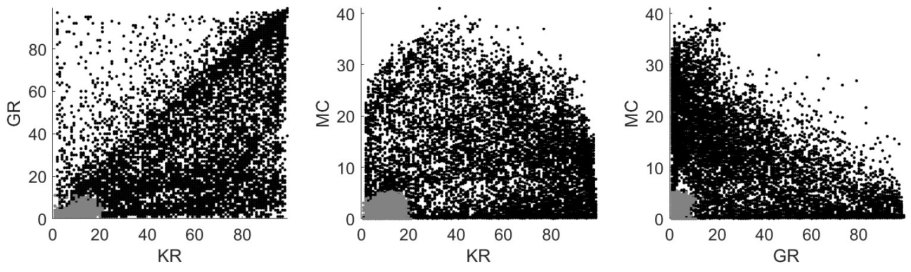

Fig. 2. Carbon nanotube in polymer behaviour space (grey) superimposed on a 100 node ESN behaviour space (black). From [7]

model [5, 7]. We have applied CHARC to a reservoir computing model instantiated in carbon nanotubes, and also in simulations.

CHARC works in the following way. First, we take a baseline system: a simulation of the computational model, here an ESN [18]. We do not want to merely examine how well a substrate implements a particular task running under a model, but rather how well it implements the model in general. So next, we decide on some 'behavioural measures' that characterise the model: for our definitional work with ESNs we use memory capacity (MC), kernel rank (KR), and generalisation rank (GR), but other measures can be used.

We then explore how well our baseline system covers the behavioural space defined by the measures, by instantiating the model with many different configurations, and evaluating the measures for each. There is a non-linear relationship between configuration space, which we use to instantiate models, and behaviour space, where we measure their properties. This means that random selection in configuration space leads to a biased sampling of behaviour space. Instead we use Novelty Search [25, 26, 27] to explore as much of behaviour space as possible.

Once we have a baseline of the amount of behaviour space covered by the baseline system (figure 2, black), we repeat the process with the candidate substrate, and measure how much of the behaviour space it covers: how many different reservoirs it can instantiate (figure 2, red). We see from the figure that our current carbon nanotube system can instantiate reservoirs, but only relatively small ones.

3.2 How well could a carbon nanotube reservoir compute?

We have demonstrated that our carbon nanotubes in polymer, deposited on a 64 electrode array, can be instantiated as small reservoir computers.

We could use the CHARC framework to engineer bigger and better *in materio* RCs, by using the amount of behaviour space covered as a fitness function in a search process, exploring carbon nanotube and polymer densities and types.

3.3 How well do other substrates compute?

The current CHARC framework can be used directly to assess other proposed RC substrates, simulated or physical, in the same manner.

The CHARC approach is not limited to reservoir computing, however. It can be adapted to evaluate how well a substrate implements other computational models. A new set of measures that provide a behaviour space specific to the new computational model need to be defined. Then the process is the same: define the behaviour space; use a simulation of the model itself as a baseline; discover what region of behaviour space the proposed substrate can be instantiated to cover, relative to the baseline.

4 How well does a computational model fit?

So far we have been focussing on evaluating, and eventually designing, substrates with respect to a given computational model: given the top abstract process of figure 1, evaluate the suitability of a given substrate to instantiate the bottom physical process. In this section I discuss the reverse problem: evaluating, and eventually designing, computational models with respect to a given substrate.

Most historical computational models were abstracted from existing substrates, for example: electronic analogue computing from the ability of circuits to implement differential equations; the Turing machine model from human 'computers' calculating by following precise instructions. Some unconventional computational models are inspired by the behaviours of specific physical, chemical and biological substrates: for example, reaction-diffusion models, and membrane models. However, there is often no explicit experimental validation of such theoretical models: how well do they capture the computing done by the substrate, or are they an abstraction too far?

Consider reversing the CHARC framework, to explore the space of computational model representations with respect to a given substrate. Performing such an exploration would need a language to express the computational models that provide the search landscape. A dynamical systems view of computation [34] could provide one such language. A dynamical system comprises a set of state variables, with equations over these variables defining the system dynamics. The relevant behaviour space would comprise dynamical properties: trajectories through state space, transients, attractors, bifurcation structure of parameterised systems, and so on.

Such a system can be visualised as graph, where nodes represent state variables, and contain the equations as state transition automata or time evolution definitions; the links show the explicit dependencies between variables. Such a visualisation is typically used for Cellular Automata, where the 'cells' correspond to the graph nodes, and the links are implicit connections to the neighbourhood nodes. Moving around the model space moves through the space of such graphs.

We are used to thinking of computational models in dynamical systems terms, even if not explicitly. Several examples include:

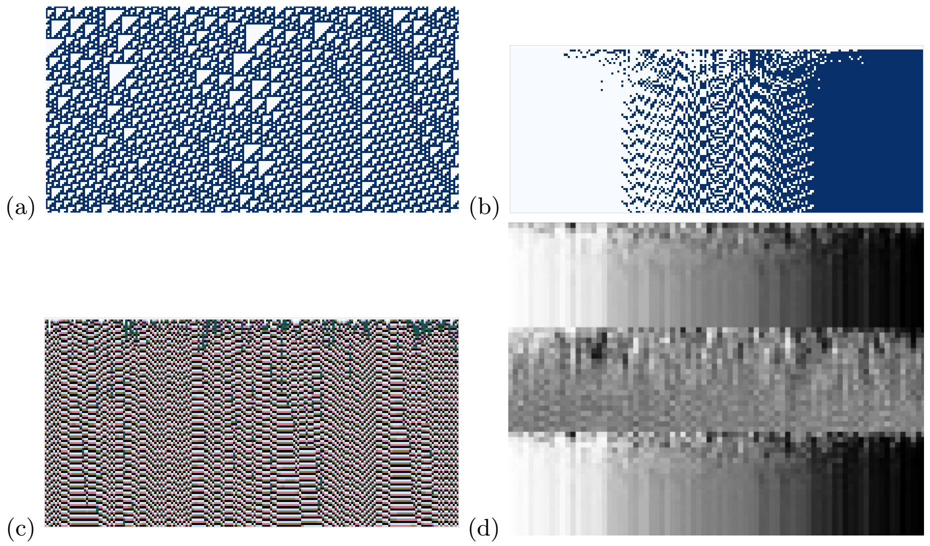

Fig. 3. The time evolution of a range of computational dynamical systems, showing their complex dynamics. The N components of the state vector run along the x axis; the T timesteps run down the y axis. (a) Elementary CA rule 110, $N = 400$, $T = 200$, random (50% 0s, 50% 1s) initial condition, periodic boundary conditions. (b) A $K = 2$ RBN with $N = 200$, $T = 80$; state components are ordered to expose the attractor and the 'frozen core'. (c) A threshold coupled lattice, threshold $x_* = 0.971$, $N = 200$, $T = 100$; each component's value $x_n \in [0, 1]$ is indicated by its shading from white $= 0$ to black $= 1$. (d) An ESN reservoir, with $N = 100$, $T = 100$; each node's value $x_n \in [-1, 1]$ is indicated by its shading from white $= -1$ to black $= 1$. Input is a square function, cycling through 1 for 20 timesteps then 0 for 20 timesteps. This ESN is in the chaotic dynamics regime.

– Cellular Automata [1, 36, 37, 38]: discrete time, discrete space, regular topology, node equations are boolean functions (figure 3a)
– Random Boolean Networks [9, 20, 21]: discrete time, discrete space, random topology, node equations are boolean functions (figure 3b)
– Coupled Map Lattices [19], threshold coupled maps [29, 30, 31]: discrete time, continuous variable, linear topology, each node equation is a (typically logistic) map with input (figure 3c)
– Reservoir computers: the ESN model is discrete time, continuous variable, random topology, node equations are some non-linear sum of inputs (figure 3d)

A computational dynamical system needs to allow inputs and outputs. Inputs may be provided on inititialisation, as the intitial state, as is typically the case with CAs and RBNs. Alternatively, the dynamical system can be *open*, allowing inputs through time as the dynamics evolves, as is typical with RCs. Outputs may comprise the final attractor state, or may be read by observing (a projection of) the system state through time.

One thing all the systems in figure 3 have in common is a fixed size state space. However, classical computational models allow the state space to change as the computation progresses: in a Turing Machine, symbols can be written to a growing tape; in higher-level programming languages, variables can come in and go out of scope. This implies that the language of dynamical systems we wish to use should support developmental, or *constructive*, dynamical systems, where the state space can change in response to the progression of the computation. There is no need for a substrate to mirror such growth explicitly: it could instead allow computing processes to move into new areas of existing material (as in classical computing memory). However, more biological substrates with actual material growth might require such a growth model. Such a dynamical system has a higher level *meta-dynamics*. Approaches such as BIOMICS [8, 28] and MGS [12, 33] provide suggestive starting points for such languages.

Dynamical systems may be coupled in flat or hierarchical architectures. For example:

– Coupled Lattice Maps and threshold coupled maps are themselves a coupling of multiple instances of some discrete-time dynamical system, typically the logistic map
– Quasi-uniform cellular automata comprise coupled patches of CAs, each with different rules [13, 32]
– Reservoirs can be coupled together, to form 'reservoirs of reservoirs' [4]
– RBN 'atoms' can be combined into 'molecular' boolean networks using an Artificial Chemistry [10, 11, 24]

One reason for considering unconventional substrates is that they may perform some tasks 'better' than conventional substrates: if not faster, then maybe with lower power, in hostile environments, in ways matched to what they are processing, or some other advantage. This leads to the idea of combining various disparate substrates, each doing what it does best, into a *heterotic* computing system [17, 22, 23]: a multi-substrate system that provides an advantage over a single-substrate system.

So, in summary, an appropriate language for capturing computational models that could be explored in a CHARC-like framework is one that supports discrete and continuous open dynamical systems, including constructive dynamical systems with a meta-dynamics, that can be combined and coupled in hierarchical architectures and in heterotic architectures.

Once such an approach is established, it could be extended to include stochastic systems and quantum systems in order to cover the full gamut of potential computing substrates.

5 Co-designing models and substrates

I have described a demonstrated approach, CHARC, that can be used to evaluate substrates for how well they can instantiate the RC model. This approach can be extended to other computational models, and could be put in a design loop to engineer improved substrates.

I have also discussed reversing this process, to design computational models appropriate for given substrates.

The final suggested step is to combine both these processes into a co-design approach: design models to fit substrates, whilst engineering those substrates to better fit the models. This would allow other features to be considered in the overall system, from ease of manufacture of the substrate to expressivity and naturalness of the model.

Additionally, the vertical lines in figure 1 (instantiation and representation; including input and output) can be addressed here in a unified manner.

6 Conclusion

In order to take unconventional computing to the next level, from small demonstrators and simple devices to large scale systems, we need a systematic engineering approach.

The discussion here is meant to point the way to one potential such process: a framework for co-designing computational models and computing substrates, to ensure the benefits of UC can be achieved without sacrificing novelty in either side of the design.

Acknowledgements

Thanks to my colleagues Matt Dale, Dom Horsman, Viv Kendon, Julian Miller, Simon O'Keefe, Angelika Sebald, and Martin Trefzer for illuminating discussions, and collaboration on the work that has led to these ideas.

This work is part-funded by the SpInspired project, UK Engineering and Physical Sciences Research Council (EPSRC) grant EP/R032823/1.

References

[1] Adamatzky, A. (ed.): Game of Life Cellular Automata. Springer (2010)
[2] Blakey, E.: Unconventional computers and unconventional complexity measures. In: Adamatzky, A. (ed.) Advances in Unconventional Computing: Volume 1: Theory. pp. 165–182. Springer (2017)
[3] Broersma, H., Stepney, S., Wendin, G.: Computability and complexity of unconventional computing devices. In: Stepney, S., Rasmussen, S., Amos, M. (eds.) Computational Matter. pp. 185–229. Springer (2018)
[4] Dale, M.: Neuroevolution of hierarchical reservoir computers. In: GECCO'18, Kyoto, Japan. pp. 410–417. ACM (2018)
[5] Dale, M., Dewhirst, J., O'Keefe, S., Sebald, A., Stepney, S., Trefzer, M.A.: The role of structure and complexity on reservoir computing quality. In: UCNC 2019, Tokyo, Japan, June 2019. LNCS, Springer (2019)

[6] Dale, M., Miller, J.F., Stepney, S.: Reservoir computing as a model for *in materio* computing. In: Adamatzky, A. (ed.) Advances in Unconventional Computing: Volume 1: Theory. pp. 533–571. Springer (2017)

[7] Dale, M., Miller, J.F., Stepney, S., Trefzer, M.A.: A substrate-independent framework to characterise reservoir computers. arXiv:1810.07135 (2018)

[8] Dini, P., Nehaniv, C.L., Rothstein, E., Schreckling, D., Horváth, G.: BIOMICS: a theory of Interaction Computing. In: Stepney, S., Rasmussen, S., Amos, M. (eds.) Computational Matter, pp. 249–268. Springer (2018)

[9] Drossel, B.: Random Boolean Networks. In: Schuster, H.G. (ed.) Reviews of Nonlinear Dynamics and Complexity, vol. 1. Wiley (2008)

[10] Faulconbridge, A., Stepney, S., Miller, J.F., Caves, L.S.D.: RBN-World: A sub-symbolic artificial chemistry. In: ECAL 2009, Budapest, Hungary, September 2009. LNCS, vol. 5777, pp. 377–384. Springer (2011)

[11] Faulkner, P., Krastev, M., Sebald, A., Stepney, S.: Sub-symbolic artificial chemistries. In: Stepney, S., Adamatzky, A. (eds.) Inspired by Nature. pp. 287–322. Springer (2018)

[12] Giavitto, J.L., Michel, O.: MGS: A rule-based programming language for complex objects and collections. ENTCS **59**(4), 286–304 (2001)

[13] Gundersen, M.S.: Reservoir Computing using Quasi-Uniform Cellular Automata. Master's thesis, NTNU, Norway (2017)

[14] Horsman, C., Stepney, S., Wagner, R.C., Kendon, V.: When does a physical system compute? Proceedings of the Royal Society A **470**(2169), 20140182 (2014)

[15] Horsman, D., Kendon, V., Stepney, S.: Abstraction/Representation Theory and the Natural Science of Computation. In: Cuffaro, M.E., Fletcher, S.C. (eds.) Physical Perspectives on Computation, Computational Perspectives on Physics, pp. 127–149. Cambridge University Press (2018)

[16] Horsman, D., Stepney, S., Kendon, V.: The Natural Science of Computation. Comms. ACM **60**(8), 31–34 (2017)

[17] Horsman, D.C.: Abstraction/representation theory for heterotic physical computing. Philosophical Transactions of the Royal Society A **373**, 20140224 (2015)

[18] Jaeger, H.: The "echo state" approach to analysing and training recurrent neural networks – with an erratum note. GMD Report 148, German National Research Center for Information Technology (2010)

[19] Kaneko, K.: Spatiotemporal intermittency in coupled map lattices. Progress of Theoretical Physics **74**(5), 1033–1044 (1985)

[20] Kauffman, S.A.: Metabolic stability and epigenesis in randomly constructed genetic nets. Journal of Theoretical Biology **22**(3), 437–467 (1969)

[21] Kauffman, S.A.: The Origins of Order. Oxford University Press (1993)

[22] Kendon, V., Sebald, A., Stepney, S.: Heterotic computing: past, present and future. Phil. Trans. R. Soc. A **373**, 20140225 (2015)

[23] Kendon, V., Sebald, A., Stepney, S., Bechmann, M., Hines, P., Wagner, R.C.: Heterotic computing. In: Unconventional Computation 2011, Turku, Finland, June 2011. LNCS, vol. 6714, pp. 113–124. Springer (2011)

[24] Krastev, M., Sebald, A., Stepney, S.: Emergent bonding properties in the Spiky RBN AChem. In: ALife 2016, Cancun, Mexico, July 2016. pp. 600–607. MIT Press (2016)

[25] Lehman, J., Stanley, K.O.: Exploiting open-endedness to solve problems through the search for novelty. In: ALIFE XI. pp. 329–336. MIT Press (2008)

[26] Lehman, J., Stanley, K.O.: Efficiently evolving programs through the search for novelty. In: GECCO 2010. pp. 837–844. ACM (2010)

[27] Lehman, J., Stanley, K.O.: Abandoning objectives: Evolution through the search for novelty alone. Evolutionary Computation **19**(2), 189–223 (2011)

[28] Nehaniv, C.L., Rhodes, J., Egri-Nagy, A., Dini, P., Morris, E.R., Horvth, G., Karimi, F., Schreckling, D., Schilstra, M.J.: Symmetry structure in discrete models of biochemical systems: natural subsystems and the weak control hierarchy in a new model of computation driven by interactions. Phil. Trans. R. Soc. A **373**, 20140223 (2015)

[29] Sinha, S., Biswas, D.: Adaptive dynamics on a chaotic lattice. Physical Review Letters **71**(13), 2010–2013 (1993)

[30] Sinha, S., Ditto, W.L.: Dynamics based computation. Physical Review Letters **81**(10), 2156–2159 (1998)

[31] Sinha, S., Ditto, W.L.: Computing with distributed chaos. Physical Review E **60**(1), 363–377 (1999)

[32] Sipper, M.: Quasi-Uniform Computation-Universal cellular automata. In: ECAL 1995. pp. 544–554. Springer (1995)

[33] Spicher, A., Michel, O., Giavitto, J.L.: A topological framework for the specification and the simulation of discrete dynamical systems. In: ACRI 2004. pp. 238–247. No. 3305 in LNCS, Springer (2004)

[34] Stepney, S.: Nonclassical computation: a dynamical systems perspective. In: Rozenberg, G., Bäck, T., Kok, J.N. (eds.) Handbook of Natural Computing, volume 4, chap. 59, pp. 1979–2025. Springer (2012)

[35] Stepney, S., Kendon, V.: The role of the representational entity in physical computing. In: UCNC 2019, Tokyo, Japan, June 2019. LNCS, Springer (2019)

[36] von Neumann, J.: Theory of Self-Reproducing Automata (edited by A.W. Burks). University of Illinois Press (1966)

[37] Wolfram, S.: Statistical mechanics of cellular automata. Reviews of Modern Physics **55**(3), 601–644 (1983)

[38] Wolfram, S.: Universality and complexity in cellular automata. Physica D. Nonlinear phenomena **10**(1), 1–35 (1984)