PHILOSOPHICAL TRANSACTIONS A

rsta.royalsocietypublishing.org

Review

Cite this article: Kendon V, Sebald A, Stepney S. 2015 Heterotic computing: past, present and future. *Phil. Trans. R. Soc. A* **373**: 20140225. http://dx.doi.org/10.1098/rsta.2014.0225

Accepted: 10 April 2015

One contribution of 13 to a Theo Murphy meeting issue 'Heterotic computing: exploiting hybrid computational devices'.

Subject Areas:
hybrid computing

Keywords:
heterotic computing, unconventional computing, hybrid computing

Author for correspondence:
Susan Stepney
e-mail: susan.stepney@york.ac.uk

Heterotic computing: past, present and future

Viv Kendon[1], Angelika Sebald[2,4] and Susan Stepney[3,4]

[1]Department of Physics, Durham University, Durham DH1 3LE, UK
[2]Department of Chemistry, [3]Department of Computer Science, and [4]York Centre for Complex Systems Analysis, University of York, York YO10 5DD, UK

VK, 0000-0002-6551-3056; SS, 0000-0003-3146-5401

We introduce and define 'heterotic computing' as a combination of two or more computational systems such that they provide an advantage over either substrate used separately. This first requires a definition of physical computation. We take the framework in Horsman *et al.* (Horsman *et al.* 2014 *Proc. R. Soc. A* **470**, 20140182. (doi:10.1098/rspa.2014.0182)), now known as abstract-representation theory, then outline how to compose such computational systems. We use examples to illustrate the ubiquity of heterotic computing, and to discuss the issues raised when one or more of the substrates is not a conventional silicon-based computer. We briefly outline the requirements for a proper theoretical treatment of heterotic computational systems, and the advantages such a theory would provide.

1. Introduction

Hybrid computational systems are ubiquitous. From graphics co-processors in desktop computers to GPS chips in mobile phones, commercial computational devices have long included specialized hardware to perform time-critical tasks more efficiently. Less conventional computational devices almost always include a classical computer as a control system somewhere in the set-up. Theoretical computer science, on the other hand, is based around elegant single-paradigm models for which computability and complexity results can be derived and comparisons made. However, the relevance of this theory to real-world devices is often unclear, and certainly does not incorporate the motivations for combining diverse computational systems when designing practical computers.

THE ROYAL SOCIETY PUBLISHING

Observing that this hybridization is not merely for convenience, but can achieve substantial advantage in the performance of the hybrid systems, we have introduced the concept of 'heterotic computing' [1,2]. 'Heterotic', from the Greek *heterosis*, is a term used in genetics to mean 'hybrid vigour'. We use the term to mean the composition of two or more potentially widely differing kinds of physical computational substrates to produce a computer that has certain computational advantages over an individual system alone. For example, two non-universal systems might be composed into a universal one, or two universal but not efficiently inter-simulable systems might be composed with an increase in computational power. These differing substrates may exploit physical, chemical and biological properties, and may include classical silicon chips.

This computational advantage may be computational power, in the case where the substrates by themselves are non-universal. More commonly, it may be due to supporting a more natural mapping of the computational problems of interest, such as where representation in an analogue component can exhibit a smaller 'semantic gap' to the problem. In this case, it may be that no one component can naturally map to the entire problem, but each heterotic component can naturally map to a component of the overall problem.

Given the existence of such heterotic devices, either the currently existing ones, or future ones to be designed, we need a theoretical analysis and design framework:

> Given that we have different basic types of computers, not necessarily Turing universal, it is natural to ask how to compose them into hybrid – 'heterotic' – computers, and to ask about the computational power of the composition. Thus, we need a framework that not only allows different models of computation to be compared and contrasted, but also allows us to compose different models and determine the resulting computational power.
>
> Kendon *et al.* [1, p. 113]

Moreover, when combining computational systems there immediately arises the question of how the components are connected, and the *communication* required between the components, which may require *transduction* if the components and/or communication channels use different encodings of the data. This is particularly important because these communication steps may involve non-trivial computation, for example translation of encodings, and hence contribute to the overall resource requirements and computational power of the system. A full accounting of all the computation in a hybrid device is crucial for fair comparisons to be made between disparate computational systems.

We begin, in §2, by defining what we mean by computation in potentially unconventional substrates, in order to distinguish computation from the substrate just 'doing its thing'. In §3, we extend this definition to heterotic systems, with two separate physical substrates. In §4, we give an overview of the current somewhat *ad hoc* state of the art in heterotic computing. Then in §5 we discuss the requirements for an appropriate theoretical framework to support the principled design and analysis of heterotic devices encompassing a range of substrate types.

2. Definition of physical computing

Computation as an abstract concept can be described and defined mathematically. Although there is debate over precise definitions, these do not impede our discussion here of the process of actually carrying out a specific computation using a physical computational device. While we can all agree that our smart phones, laptops and desktop computers are carrying out computations for us, there are more exotic examples where the presence of an actual computation is less clear cut. Consider, for example, leech neurons on a silicon substrate [3]: Is it computing? What is computing? How can we tell?

The question of when a physical system is computing was considered in detail by Horsman *et al.* [4]. The basic framework is shown in figure 1. The key step is the use of *representation relations* (vertical double arrows in the figure). Representation is how we connect abstract concepts and physical objects. Two types of representation relations are shown in figure 1: (i) the modelling

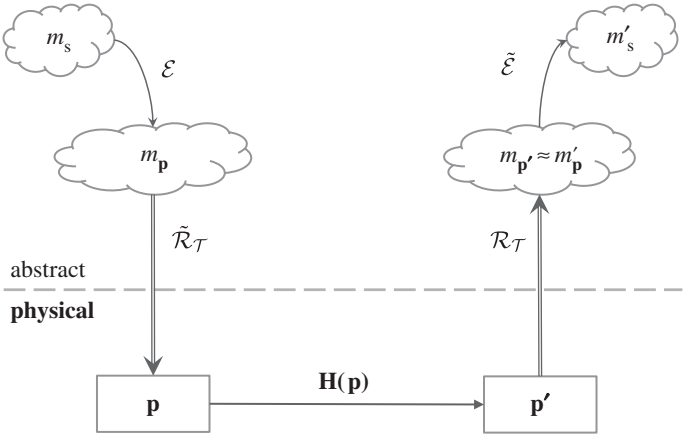

Figure 1. A physical system **p** being used to compute an abstract computation m.

representation relation $\mathcal{R}_{\mathcal{T}}$ indicates creation of an abstract model of a physical system; (ii) the instantiation representation relation $\tilde{\mathcal{R}}_{\mathcal{T}}$ indicates an abstract model is instantiated as a physical system. Despite the apparent symmetry in the figure, instantiation $\tilde{\mathcal{R}}_{\mathcal{T}}$ requires much more work than modelling $\mathcal{R}_{\mathcal{T}}$. It requires us to understand the theory $\mathcal{T}$ sufficiently well that we can engineer a corresponding physical system.

Given these relations, physical computing can be explained as follows. We start with an abstract mathematical problem to be solved, $m_{\mathbf{s}}$. This is encoded into the mathematical model of the computer $m_{\mathbf{p}}$ using the encoding $\mathcal{E}$. The desired abstract computation would result in $m'_{\mathbf{p}'}$, from which the result of the computation $m'_{\mathbf{s}}$ is extracted using the decoding process $\tilde{\mathcal{E}}$. The physical computer **p** is then programmed using the description $m_{\mathbf{p}}$ and the instantiation relation, and the program allowed to run, $\mathbf{H}(\mathbf{p})$. The final state $\mathbf{p}'$ is inspected through the modelling relation to obtain the mathematical description of the outcome $m_{\mathbf{p}'}$. The physical system, along with the relevant representation relations, has performed the desired computation when the observed $m_{\mathbf{p}'}$ is a sufficiently close prediction of the desired $m'_{\mathbf{p}}$. The outcome of the computation is then decoded using $\tilde{\mathcal{E}}$ to obtain $m'_{\mathbf{s}}$. Note that most of these steps are highly non-trivial, and may themselves require significant computation, e.g. $\mathcal{E}$ and $\tilde{\mathcal{E}}$, or involve prior work to engineer a suitable physical system and establish the instantiation and representation relations.

From this, we can state several necessary conditions for physical computation to be actually taking place [4], most crucial being the presence of the representation of the abstract computation in the physical system. This in turn implies there must be something capable of supporting such a representation, termed a *computational entity* by Horsman *et al.* [4]. A computational entity does not need to be human, or even biological; it simply has to be capable of supporting an abstract model of the computation at the appropriate level of complexity. A key characteristic of the abstract representation is that there is a degree of arbitrariness about it. For example, representing the abstract binary values zero or one in a physical system could use 'spin up' to represent zero and 'spin down' for one, or equally well use 'spin down' to represent zero and 'spin up' to represent one. Without representation, there is no computation, there is only the physical system doing its thing.

This framework allows a clear description of the case of a physical system simulating another physical system, or even simulating itself (see [4, fig. 10]). The representation, and hence the computational entity, must be present for the process to be a simulation, rather than just the physical system evolving as it usually does. What is being simulated is the abstract model of the physical system: computations are abstract mathematical concepts; the only way for one system to simulate another is through a logical or mathematical theory of both systems.

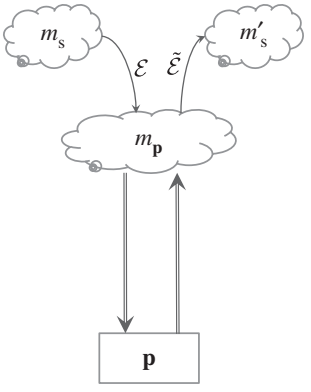

Figure 2. Abbreviated version of figure 1: a single physical computational system.

In figure 2, the computational cycle in figure 1 has been abbreviated as a single two-way representation in which the evolution $\mathbf{H}(\mathbf{p})$ is not explicitly shown. This enables us to more compactly describe the heterotic case.

3. Heterotic physical computing

In §2, we defined physical computation as a single process using a single physical system. Heterotic computation requires us to examine the physical system and abstract model more closely, as subdivisions of the abstract $m_\mathbf{p}$ and physical $\mathbf{p}$ systems shown in figure 2. Or, equivalently, we can consider how to *compose* two or more such computational systems. An important question is how the composition of physical systems affects the composition of computational systems, reflected across the abstract/physical divide via the representation relation.

A simple example of composition is a trivial composition of two identical systems to make one larger one, a non-trivial computational advantage in the sense that one now has a larger computer capable of larger computations than the two systems side by side with no connections between them.

There are essentially two ways to compose computational systems: serial and parallel. In serial composition, figure 3*a*, information flows from one system to the next, uni-directionally. Input is provided to the first system, and output from the second. In parallel composition, figure 3*b*, information can flow both ways between the two systems. Input and output are most naturally handled via one of the systems, but can be from both.

When the composition is of two different types of system, transduction (conversion of information and/or energy) of input and output formats may be required, if information is represented differently in the two systems. This is shown as a transduction 'cloud' at the abstract level, and as a box at the physical level, in figure 3. The similarity between transduction and computation is not accidental: conversion of data formats usually involves non-trivial computation, and clearly there is a representation arrow that could be placed between the abstract transduction cloud and the physical transduction box. One can thus require the additional constraint that composition can only be applied between compatible inputs and outputs; where there is a mismatch, a third computational element—the transducer—must be used to convert the signals. This restriction ensures that all computation associated with signal processing will be counted. An example of a transduction step where a classical computer is employed in an experiment involving unconventional substrates can be found in [5]. There, gate operations performed on an nuclear magnetic resonance (NMR) computer are linked via classical control systems.

Clearly, one can compose more than two systems in a similar manner, but in what follows for simplicity we consider just two different systems composed. Since we are focusing on the physical

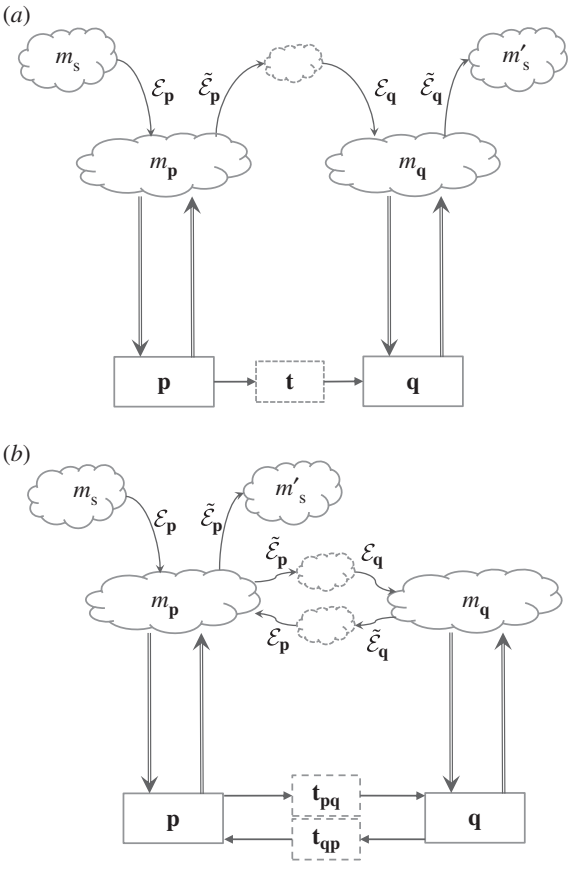

Figure 3. Generic ways to compose computational systems (using the abbreviated figure style of figure 2): (*a*) serial composition with transduction of communication, **t**; (*b*) parallel composition with two-way transduction.

systems, we generally omit the abstract model of the system from the following figures. It is to be understood that for these systems to be actually used for computation, the abstract component must also be present, and the representation relation between the abstract model and the physical system established by the computational entity that is using the computer [4].

4. Current examples of heterotic computing

We now flesh out these definitions by giving examples of a range of existing computers that can be described in heterotic terms. In all these cases, there are two distinct physical systems of computational substrate; in most cases, one system is a classical computer, and one is a somewhat more or less unconventional substrate. Future heterotic devices may well include multiple diverse unconventional substrates, and may omit a classical system. This section allows us to bring out by example some of the many issues that arise in hybrid and heterotic computation.

(a) Classical computing co-processors

Classical desktop computer architectures have for years included specialized co-processors to do common tasks more efficiently. While the basic CPU (central processing unit) is computationally complete (for its size and available memory), one of the most crucial aspects of desktop computers is their speed relative to their human users. This is particularly apparent for the display screens, for which graphics co-processors (GPUs) have been standard for a couple of decades. The

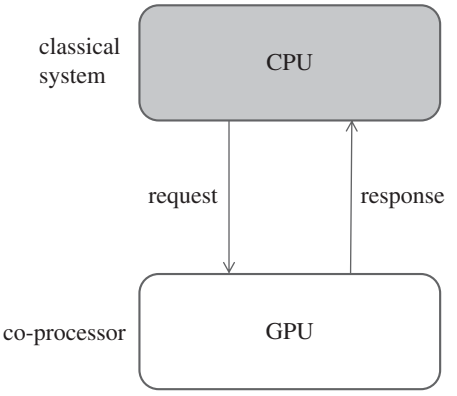

Figure 4. Co-processor computing.

GPU is also computationally complete, but has a different architecture from the main CPU to facilitate parallel updating of the pixels in the display. While the main CPU is optimized for serial processing of data, with a few processing cores running in parallel (typically four to 16 today), the GPU has hundreds or thousands of cores each dedicated to a few pixels. This facilitates the real time updating of the display screen faster than the reading and reaction time of a human user. A modern computer thus has two complementary processing units, one optimized for serial processing where the next step in the computation typically depends on any or all of the results of the current step, the other optimized for parallel processing where the processing steps are typically similar for all data and dependent only on local values of the data. Using the CPU for screen rendering would be far slower, suitable only for the text display that the early computers had.

Figure 4 shows a (highly simplified) structure of this situation as a heterotic system. There are two physical systems, a classical computer and a co-processor, connected in parallel, corresponding to the **p** and **q** in the lower portion of figure 3*b*. The transduction in this case is trivial: both systems represent information using the same digital encoding and physical memory. Of course, the detailed architecture is more complicated than this (e.g. [6]), but those details are not necessary for this particular description.

The power of GPUs has been harnessed for other data processing tasks that lend themselves to a high degree of parallel processing, and chips with the GPU architecture are now produced specifically for these computational tasks as well as for graphics cards. Both uses require the interplay between the GPU as a fast processor and the CPU as the controller, providing the instructions to the GPU and taking the higher level decisions about what to compute and when. These are tasks that are naturally serial, depending on the outcome of previous computations and instructions from the user.

There are many other silicon-based but non-CPU style co-processors, including floating point co-processors, non-universal ASICs (application-specific integrated circuits), field-programmable gate arrays (FPGAs) and field-programmable analog arrays. They all share the co-processor architecture of the GPU (figure 4) and are all motivated by the efficiency gained through optimizing the architecture for a specific type of computation.

(b) Measurement-based quantum computing

Quantum computation is in a different computational complexity class from classical computation. By using the intrinsic parallel nature of quantum physics in the form of interference effects (wave phenomena), superposition (being in more than one state at the same time) and entanglement (physical correlations between states), quantum computers can solve some problems fundamentally faster than can classical computers. The canonical examples are database

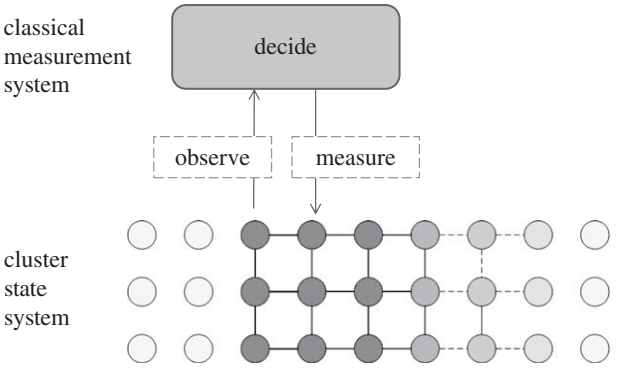

Figure 5. Cluster state computing (adapted from Stepney *et al.* [2, fig. 1a]).

searching [7], polynomially faster than the fastest possible classical algorithm, and factoring large semi-primes [8,9], exponentially faster than the fastest known classical algorithm. In a quantum computer, classical bits are replaced by quantum bits or qubits, which can be in a superposition of zero and one at the same time, thus representing several numbers in the memory required for a single number in a classical computer register. There are several distinct architectures for quantum computers, though none have been built beyond the proof-of-concept stage yet.

Of particular interest here is the cluster state or *one-way* quantum computer [10], also known as a measurement-based quantum computer (MBQC). In MBQCs, an entangled resource of many qubits is prepared, then the computation proceeds by measuring the qubits in turn. The observed outcomes from the measurements feed forward to determine the type of measurement performed on the next qubits (figure 5).

The role of the classical controlling system in MBQCs was first noted by Jozsa [11], while demonstrating the equivalence of measurement-based and teleportation-based quantum computing schemes. Anders & Browne [12] realized that the classical computation required to control and feed-forward information in MBQC is a crucial part of the computational power. Applying measurements without feed-forward is efficiently classically simulable, as is (trivially) the classical part of the computation. However, the combination of the two is equivalent to the quantum circuit model with unbounded fan-out [13], which is not (efficiently) classically simulable. Thus, the combination of two or more systems, to form a new computational system composed of several distinct physical systems, can be in a more powerful computational class than the systems acting separately.

This is to date the only known example where a heterotic computer is actually in a different complexity class from its components, though this is largely because this is an unexplored area, and we expect further examples to be forthcoming.

(c) NMR computing

Liquid-state NMR has been used to perform simple classical gate logic, such as NAND, and to compute small circuits of gates [5]. Figure 6 shows the two physical systems involved, a classical control system and an NMR system, composed in parallel. The NMR system is configured to implement logic gates. The specific input bit values are transduced into the relevant frequency and phase of the NMR operation. The radio frequency output is transduced by integrating and thresholding, to result in the value of the output bit. The classical system sequences the outputs from one gate to the inputs of the next in the circuit.

Using NMR as a platform to perform classical computing involves choosing from the very rich set of available parameters some subset that is suitable for representing a particular logic operation. Additionally, the physical operations chosen must be compatible with the requirements of having an ensemble of nuclear spins that behaves classically: the ensemble must

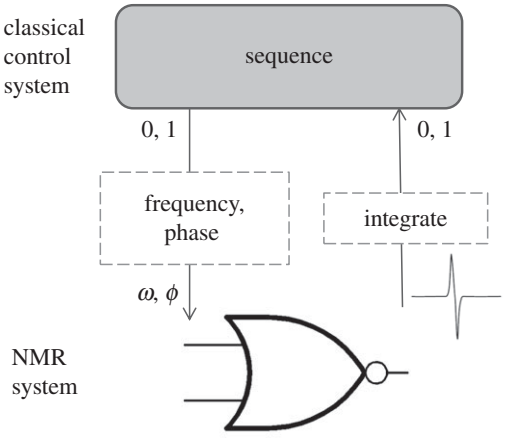

Figure 6. NMR classical computing (adapted from Stepney *et al.* [2, fig. 1b]).

be kept in fully determined classical states (in NMR spectroscopy terminology: the spin dynamics must be such that they can be represented by a magnetization vector).

Prior work on computation using NMR mostly deals with implementations of quantum computations, predominantly based on solution-state NMR experiments (for a review, see [14]), with some examples exploiting solid-state NMR [15]. As a test-bed for techniques, NMR has been very useful in the development of quantum computing, but liquid-state NMR has intrinsic shortcomings related to the purity of the quantum states and lacks scalability for building production-size hardware. Where NMR is used for the implementation of (pseudo)quantum computations, the choice of experimental parameters for encoding is slightly more restricted than for the classical case, but the limitations on the physical properties are reduced: for these purposes more complicated, much rich spin dynamics are wanted (and necessary).

As a step towards characterizing the computational power of the NMR system, Bechmann *et al.* [16] have produced a preliminary classification of the experimental NMR parameters for implementing classical logic gates. This work has been extended to take advantage of the inherently continuous nature of the NMR parameter space of non-coupled spin species [17] by implementing continuous gates, so the combined system performs an analogue computation.

As with MBQC, the classical control system plays an essential role in the computation, but by itself it does not perform the individual gate logic. The extent to which such control system contributes to the overall computational power of quantum or classical NMR computing has yet to be formally analysed.

(d) Ancilla-based quantum computing

Most gate-based architectures for quantum computers use some sort of ancillary system to enact the two-qubit gates. The dilemma is this: quantum systems are fragile and susceptible to disturbance by their environment, which corrupts the quantum information they represent. In order to minimize this disturbance, they need to be isolated as far as possible. However, enacting quantum gates requires controlled disturbance, which is hard to do to well-protected systems. There are many ways to optimize the storage of quantum information (quantum memories) as well as the efficiency of quantum gates, but all involve extra operations or trade-off between optimal properties. One of the most useful techniques in practice is to use ancillary quantum systems to enact the quantum gates (e.g. [18]).

The theory of ancilla-mediated quantum computation has been abstracted into a framework where a quantum system (ancilla) controls another quantum system (the qubits), with [19] or without [20] measurement of the ancilla system during the computation. This framework is capable of modelling many types of hybrid quantum computing architectures. For some common

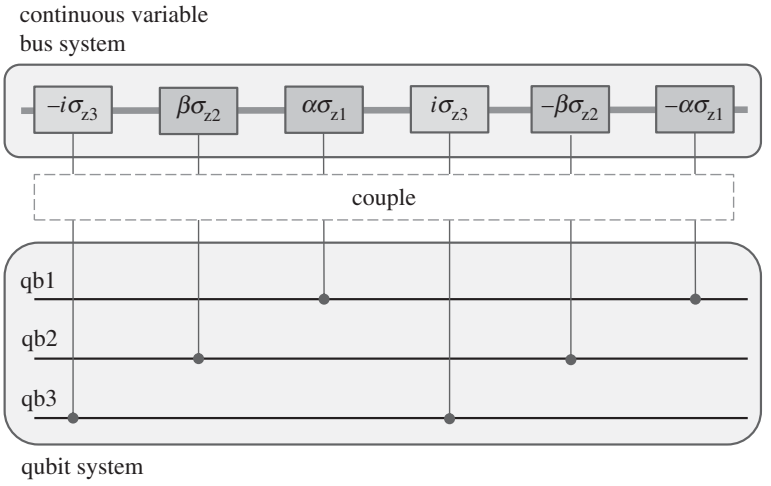

Figure 7. Ancilla-based computing (adapted from Stepney *et al.* [2, fig. 1c]).

types of gate sequences, extra efficiency savings become available [21,22] due to the extra degrees of freedom in the ancilla.

The qubus quantum computer uses a quantum state known as a *coherent state* as the ancilla, which has two quadratures, which act as two coupled continuous variable quantum systems. This type of ancilla can interact with many qubits at the same time, allowing savings in the number of basic operations required for gate operations [21] and for building cluster states [23,24] for MBQC. Figure 7 shows a sequence of six operations that performs four gates, one between each possible pair of the three qubits. Each gate performed separately would require two operations, thus this sequence saves at least two operations over standard methods, more if the qubits have to be swapped to adjacent positions for direct gates. Typically, this provides polynomial reductions in the number of elementary operations required for a computation, when compared with interacting the qubits directly.

However, compiling these gate sequences requires classical computation, which is usually regarded as 'free' when calculating the resources required for the computation. Hence the savings are achieved in part because of the differential cost of quantum and classical computation, where unloading everything possible onto a classical computer is cost effective. In general, all quantum computers run a program that is specified as classical data, and for many architectures classical controls are also required to implement the gate sequences. One architecture that does not need classical controls during execution of the algorithm is computation by continuous-time quantum walk [25]. The computation proceeds by allowing a quantum process to travel ballistically through a graph that represents the gate sequence for the computation. Even so, the graph structure on which the quantum walk takes place still has to be computed or compiled.

The classical component in quantum computation is an understudied area, and it is not known in general what the minimal quantum component is that can achieve universal quantum computation. Nor, for that matter, is the minimum 'classical' component in quantum computing known: all experimental proof-of-concept quantum computers contain extensive classical controls.

(e) Chemical and biological co-processors

There are many uses of unconventional chemical and biological substrates that are given as examples of unconventional computers. When examined, many of these are acting as unconventional co-processors, with a classical computational component performing a non-trivial part of the computation. In many cases, this classical computation comprises a substantial

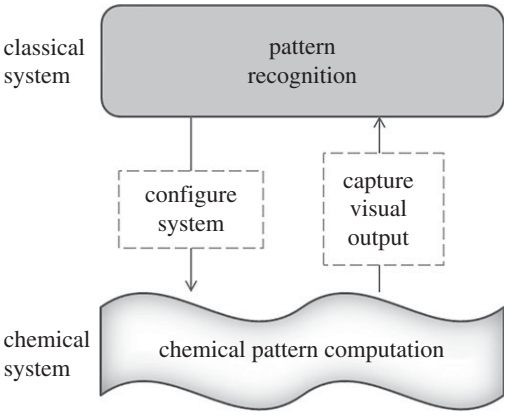

Figure 8. Generic chemical computing.

image processing stage to extract the computational result residing as a spatial pattern in the unconventional substrate.

Figure 8 illustrates the generic arrangement. A classical system defines the configuration required of the chemical system (for example, the placement of chemical spots). The chemical system evolves for a time, and then the spatial configuration is captured, typically photographically. The resultant image is post-processed classically to extract the result of the computation.

Reaction–diffusion (RD) computers [26] use interacting chemical substrates to perform a variety of computations. For example, an RD computer can be used to calculate a two-dimensional Voronoi diagram from a set of points [27,28]. The input set of abstract mathematical two-dimensional points is instantiated in a set of chemical droplets in two-dimensional space. The chemical substrate is designed so that reaction waves propagate from these points at a constant speed. Where waves intersect, their reactions result in a different colour, and form the visible boundaries between the respective regions. The output is this visible pattern of chemicals. To convert this output into its mathematical representation of edges and regions requires image processing on a classical computer. Similarly, RD computers can be used to find shortest paths through a maze [29], but classical post-processing of the visible output is needed to extract the actual path.

Slime moulds are another substrate used to compute various shortest-path problems [30]. Here planar graph nodes are instantiated as oat flakes in the corresponding positions. The slime mould grows between these oat flake food sources forming its body into a shortest-length configuration: the resulting slime mould body instantiates the graph edges. Again, post-processing is necessary to convert this output into the relevant mathematical representation of a graph.

(f) Intrinsic evolution

Evolutionary algorithms (e.g. [31]) are a class of meta-heuristic search algorithms, inspired by the process of Darwinian evolution. A *population* of candidates have their *fitness* (closeness to the desired solution) evaluated; fitter candidates are preferentially selected to produce the next generation, using a variety of operators such as mutation and crossover.

When the problem is purely computational, the fitness can be calculated *in silico*. When the problem involves the design of a physical artefact, there are two alternatives: *simulation* of the physical solution *in silico* and *intrinsic* evaluation, using either the actual physical system, or a physical simulation (for example, a scale model) of the actual physical system.

Where the evaluation is intrinsic, we can see this as a form of heterotic computing. The selection and breeding is done in a classical software 'evolutionary harness', as is the definition

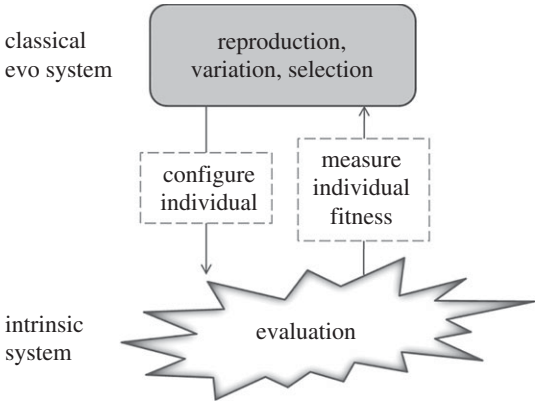

Figure 9. Intrinsic evolutionary computing.

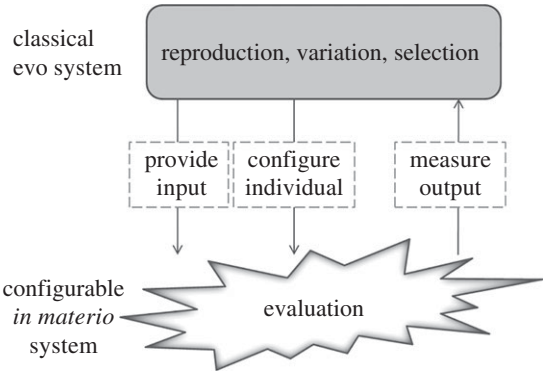

Figure 10. *In materio* computing, evolutionary search stage.

of the configuration of the individuals to be constructed and assessed. The individuals are constructed, and their resulting fitness is measured, to feed back into the evolutionary harness (figure 9). The computational gain comes from not having to provide a simulation, from not having to know the physical theory of the material, and from having an essentially nil semantic gap.

(g) *In materio* computation

Intrinsic evolution can be used to design computational artefacts. In such a case, where the different individuals can each be configured in the same piece of hardware, rather than each created from scratch, intrinsic evolution is called 'evolvable hardware' (for silicon devices including FPGAs) or 'evolution *in materio*' (for non-silicon devices such as liquid crystals and carbon nanotubes). Lohn & Hornby [32] provide a survey of evolvable hardware, and Miller *et al.* [33] a survey of *in materio* computing. Specific examples include Thompson's evolved FPGA tone discriminator [34], Harding and Miller's liquid crystal systems [35] and more general *in materio* computing [36], and evolutionary robotics [37].

Figure 10 shows the evolution phase of the process. There are two inputs to the configurable system: configuration instructions, and computational input. The system performs its function, and the output is observed, and used as a fitness measure in the evolutionary process. This is computation, rather than mere experiment [4]. In an experiment, we are trying to discover the underlying dynamics of a system; here, we are trying to discover which system in an ensemble

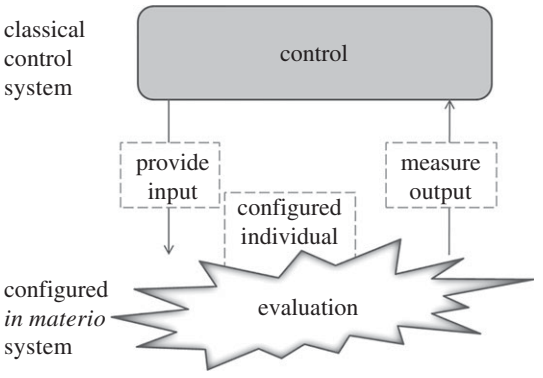

Figure 11. *In materio* computing, use stage.

has the desired computational dynamics (alternatively: we are trying to engineer the configurable system to have the desired computational dynamics).

Figure 11 shows the use of the resulting evolved system. The desired configuration of the system as discovered by the evolutionary process is fixed into the system, configuring it to be the desired physical system. The computational input is provided to the system, which performs its function. The output is observed, and interpreted as the result of the computation.

5. Summary and conclusion

(a) Need for heterotic computing

As we have demonstrated through our examples, single paradigm computation is the exception in practical devices. The era of faster general purpose computers through smaller faster components (via Moore's law) is already over. The need to reduce power consumption has required multi-core CPU designs, and the demand for fast and realistic graphics spawned GPU co-processors two decades ago. Future developments are focused on achieving greater speed and efficiency through better matching of the device to the task. The diversity of computing applications from space probes to nanobots is driving innovation in many directions including the development of a much wider selection of physical computational systems, each with advantages for specific applications.

Heterotic computing is essential for effective development of unconventional computational systems, which are rarely viable without additional computational components. Single substrate computation has many issues, as laid out in Stepney *et al.* [38], such as non-universality, wiring problems in communicating between parts of the substrate, and encoding and decoding the input and output. All of these can be—and are being—solved by combining complementary substrates, and often by including more conventional control systems. As we have demonstrated in the examples in §4, the importance of these control systems to the overall computational power of the devices is largely unrecognized, which hinders the design and development of applications.

Also crucial to applying these new computational substrates in practical devices is a proper and fair evaluation of their computational abilities. Some unconventional substrates where super-computational properties are claimed, are hiding their computation in unfamiliar resources [39] and in the transduction stages. Applying a heterotic analysis will help to expose the hidden assumptions and understand these systems fully.

(b) Requirements for heterotic computing

If we want to use an unconventional physical system for computation, the first question to answer is whether this system is actually computing in the situation we want to use it [4].

Without a sufficiently well-tested theory of how the system works, we cannot rely on it for computation, although we can of course design experiments that will improve our confidence. From a theoretical point of view, single-substrate computation is certainly easier to analyse and abstractly model. However, such models may be 'unphysical' in the sense that they ignore the practical requirements for transduction, control and scalability of real physical devices. This leaves us unable to apply the theoretical analysis to the design of actual computers, unless we have a framework in which all the physical requirements can be included.

A framework for heterotic computing will be highly non-trivial. Many of the issues it will need to handle involve the encodings used to represent the data, and the signal transduction required to convert the data formats when information is passed between different substrates. The information input to and output from diverse substrates will often be in different materials or energy types. For example, optical computation is very fast but conversion to electronic signals to interface with classical computers can be a bottleneck that negates the optical gains. The current trend to mitigate this is for all optical computation, but this cannot be a general solution, especially for substrates where communication is slow or difficult to achieve effectively at all.

When the substrate has been chosen to be a good match to the data types for the computation, explicit accounting of the data encoding and decoding is also required: we cannot just convert everything to a common standard without losing some of the computational advantages. Different substrates may also compute on very different timescales, resulting in an impedance mismatch when connected. Compare nanoseconds for optical, seconds for chemical and hours or days for biological computation [38], and the need for compatible memories to store the signals until required becomes clear. The correct programming model for such diverse systems in combination is also largely unexplored.

Given a hybrid system we would like to analyse, there may be several valid decompositions into constituent systems, depending on the focus of our study. This is especially true when highly engineered substrates are being used, such as various engineered nanoparticle structures, or electrical–chemical composites, where the engineered physical system is not as well characterized as is the 'pure' system. Further research is required to understand the relationship between different possible decompositions.

(c) Future of heterotic computing

All of our examples in §4 consist of a substrate with interesting computational potential combined with a classical control layer provided by a conventional digital computer. One potential exception is in quantum computing with ancilla-mediated quantum gates (§7), although there are classical controls of one form or another required for most versions of this model. In fact, it is not clear that quantum computing without any classical component is possible if 'whole system' analysis is performed.

Stepney *et al.* [38] have proposed a heterotic system combining optical, bacterial and chemical computing as a proof-of-concept exploration of the very different time scales in these systems, but with direct interfaces that do not require a classical computer to mediate.

As we have sought to demonstrate with our examples and discussion, most of our current computing is in fact heterotic. Acknowledging this, and developing the theory to go along with the practice, will enable future computation to be better understood and hence more reliable, more efficient in both computational power and resource requirements, and better adapted to its purpose in a myriad of settings.

Authors' contributions. All authors contributed equally to developing the ideas; S.S. and V.K. drafted the manuscript; A.S. helped to draft the manuscript; all authors gave final approval for publication.

Competing interests. The authors have no competing interests.

Funding. S.S. acknowledges partial funding by the EU FP7 FET Coordination Activity TRUCE (Training and Research in Unconventional Computation in Europe), project reference number 318235. V.K. is funded by EPSRC (UK Engineering and Physical Sciences Research Council) fellowship EP/L022303/1.

References

1. Kendon V, Sebald A, Stepney S, Bechmann M, Hines P, Wagner RC. 2011 Heterotic computing. In *Unconventional computation 2011, Turku, Finland, June 2011*. LNCS, no. 6714, pp. 113–124. Berlin, Germany: Springer.
2. Stepney S, Kendon V, Hines P, Sebald A. 2012 A framework for heterotic computing. In *8th Workshop on Quantum Physics and Logic* (*QPL 2011*), *Nijmegen, The Netherlands*. EPTCS, no. 95, pp. 263–273.
3. Fromherz P, Offenhausser A, Vetter T, Weis J. 1991 A neuron-silicon junction: a Retzius cell of the leech on an insulated-gate field-effect transistor. *Science* **252**, 1290–1293. (doi:10.1126/science.1925540)
4. Horsman C, Stepney S, Wagner RC, Kendon V. 2014 When does a physical system compute? *Proc. R. Soc. A* **470**, 20140182. (doi:10.1098/rspa.2014.0182)
5. Roselló-Merino M, Bechmann M, Sebald A, Stepney S. 2010 Classical computing in nuclear magnetic resonance. *Int. J. Unconv. Comput.* **6**, 163–195.
6. Wilt N. 2013 *The CUDA handbook: a comprehensive guide to GPU programming*. Reading, MA: Addison Wesley.
7. Grover LK. 1996 A fast quantum mechanical algorithm for database search. In *Proc. 28th Annual ACM Symposium on the Theory of Computing* (*STOC*), pp. 212–219. New York, NY: ACM.
8. Shor P. 1994 Algorithms for quantum computation: discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pp. 124–134. Piscataway, NJ: IEEE Computer Society Press.
9. Shor PW. 1997 Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Sci. Statist. Comput.* **26**, 1484. (doi:10.1137/S0097539795293172)
10. Raussendorf R, Briegel HJ. 2001 A one-way quantum computer. *Phys. Rev. Lett.* **86**, 5188–5191. (doi:10.1103/PhysRevLett.86.5188)
11. Jozsa R. 2005 An introduction to measurement based quantum computation. (http://arxiv.org/abs/quant-ph/0508124)
12. Anders J, Browne D. 2009 Computational power of correlations. *Phys. Rev. Lett.* **102**, 050502. (doi:10.1103/PhysRevLett.102.050502)
13. Browne D, Kashefi E, Perdrix S. 2010 Computational depth complexity of measurement-based quantum computation. In *TQC 2010* (eds W van Dam, VM Kendon, S Severini). LNCS, no. 6519, pp. 35–46. Berlin, Germany: Springer.
14. Jones JA. 2011 Quantum computing with NMR. *Progr. Nuclear Magn. Reson. Spectroscopy* **59**, 91–120. (doi:10.1016/j.pnmrs.2010.11.001)
15. Collins D. 2000 NMR quantum computation with indirectly coupled gates. *Phys. Rev. A* **62**, 022304. (doi:10.1103/PhysRevA.62.022304)
16. Bechmann M, Sebald A, Stepney S. 2012 Boolean logic-gate design principles in unconventional computers: an NMR case study. *Int. J. Unconv. Comput.* **8**, 139–159.
17. Bechmann M, Sebald A, Stepney S. 2010 From binary to continuous gates—and back again. In *ICES 2010, York, UK, September 2010*. LNCS, no. 6274, pp. 335–347. Berlin, Germany: Springer.
18. Cirac JI, Zoller P. 1995 Quantum computations with cold trapped ions. *Phys. Rev. Lett.* **74**, 4091. (doi:10.1103/PhysRevLett.74.4091)
19. Anders J, Oi DKL, Kashefi E, Browne DE, Andersson E. 2010 Ancilla-driven universal quantum computation. *Phys. Rev. A* **82**, 020301. (doi:10.1103/PhysRevA.82.020301)
20. Proctor TJ, Andersson E, Kendon V. 2013 Universal quantum computation by the unitary control of ancilla qubits and using a fixed ancilla-register interaction. *Phys. Rev. A* **88**, 042330. (doi:10.1103/PhysRevA.88.042330)
21. Brown KL, De S, Kendon V, Munro WJ. 2011 Ancilla-based quantum simulation. *New J. Phys.* **13**, 095007. (doi:10.1088/1367-2630/13/9/095007)
22. Proctor TJ, Dooley S, Kendon V. 2015 Quantum computation mediated by ancillary qudits and spin coherent states. *Phys. Rev. A* **91**, 012308. (doi:10.1103/PhysRevA.91.012308)
23. Brown KL, Horsman C, Kendon V, Munro WJ. 2012 Layer by layer generation of cluster states. *Phys. Rev. A* **85**, 052305. (doi:10.1103/PhysRevA.85.052305)
24. Horsman C, Brown KL, Munro WJ, Kendon VM. 2011 Reduce, reuse, recycle for robust cluster-state generation. *Phys. Rev. A* **83**, 042327. (doi:10.1103/PhysRevA.83.042327)
25. Childs AM. 2009 Universal computation by quantum walk. *Phys. Rev. Lett.* **102**, 180501. (doi:10.1103/PhysRevLett.102.180501)

26. Adamatzky A, de Lacy Costello B, Asai T. 2005 *Reaction–diffusion computers*. Elsevier.
27. Adamatzky A. 1994 Constructing a discrete generalized Voronoi diagram in reaction-diffusion media. *Neural Network World* **4**, 635–644.
28. Tolmachev D, Adamatzky A. 1996 Chemical processor for computation of Voronoi diagram. *Adv. Mater. Opt. Elect.* **6**, 191–196. (doi:10.1002/(SICI)1099-0712(199607)6:4<191::AID-AMO238>3.0.CO;2-G)
29. Steinbock O, Tóth Á, Showalter K. 1995 Navigating complex labyrinths: optimal paths from chemical waves. *Science* **267**, 868–871. (doi:10.1126/science.267.5199.868)
30. Adamatzky A. 2010 *Physarum machines: computers from slime mould*. Singapore: World Scientific.
31. Mitchell M. 1996 *An introduction to genetic algorithms*. Cambridge, MA: MIT Press.
32. Lohn JD, Hornby GS. 2006 Evolvable hardware: using evolutionary computation to design and optimize hardware systems. *IEEE Comput. Intell. Mag.* **1**, 19–27. (doi:10.1109/MCI.2006.1597058)
33. Miller JF, Harding SL, Tufte G. 2014 Evolution-in-materio: evolving computation in materials. *Evol. Intell.* **7**, 49–67. (doi:10.1007/s12065-014-0106-6)
34. Thompson A. 1997 An evolved circuit, intrinsic in silicon, entwined with physics. In *Proceedings of the First International Conference on Evolvable Systems: From Biology to Hardware, ICES'97*. LNCS, no. 1259, pp. 390–405. Berlin, Germany: Springer.
35. Harding SL, Miller JF, Rietman EA. 2008 Evolution in materio: exploiting the physics of materials for computation. *Int. J. Unconv. Comput.* **4**, 155–194.
36. Harding S, Miller JF. 2012 Evolution in materio. In *Computational complexity* (ed. RA Meyers), pp. 1030–1042. Berlin, Germany: Springer.
37. Pollack JB, Lipson H, Ficci S, Funes P, Hornby G, Watson RA. 2000 Evolutionary techniques in physical robotics. In *Proceedings of the Third International Conference on Evolvable Systems: From Biology to Hardware, ICES'00*. LNCS, no. 1801, pp. 175–186. Berlin, Germany: Springer.
38. Stepney S, Abramsky S, Bechmann M, Gorecki J, Kendon V, Naughton TJ, Perez-Jimenez MJ, Romero-Campero FJ, Sebald A. 2012 Heterotic computing examples with optics, bacteria, and chemicals. In *Unconventional Computation and Natural Computation 2012, Orleans, France, September 2012*. LNCS, no. 7445, pp. 198–209. Berlin, Germany: Springer.
39. Blakey E. 2011 Unconventional complexity measures for unconventional computers. *Nat. Comput.* **10**, 1245–1259. (doi:10.1007/s11047-010-9226-9)