Embedding infinitely parallel computation in Newtonian kinematics

E.J. Beggs [a,1] J.V. Tucker [b,2]

[a]*Department of Mathematics, University of Wales Swansea, SA2 8PP, U.K.*

[b]*Department of Computer Science, University of Wales Swansea, SA2 8PP, U.K.*

Abstract

First, we reflect on computing sets and functions using measurements from experiments with a class of physical systems. We call this *experimental computation.* We outline a programme to analyse theoretically experimental computation in which a central problem is: *Given a physical theory T, explore and classify the computational models that can be embedded in, and abstracted from, the physical systems specified by the physical theory T.* We consider the embedding *arbitrary* sets, functions, programs, and computers into designs for systems that can be specified in subtheories or fragments T of Newtonian kinematics in order to explore some of the physical assumptions of T that allowed its systems to qualify as hyper-computers, i.e., physical models that compute sets and functions that cannot be computed in classical computability theory. In designing systems we work strictly within the chosen theory T and do *not* concern ourself with whether or not T is valid of the world today. We are interested in exploring the subtheory from a computational point of view and especially in restrictions on the assumptions of T that allow us to return from hyper-computation to classical computation.

Secondly, we give a construction of an infinitely parallel machine that can decide all the arithmetical sets of natural numbers. We embed this hyper-computer as system in 3-dimensions obeying the laws of a fragment of Newtonian kinematics. In particular, the example shows that communication allowable in Newtonian kinematics is especially powerful. We conclude with further reflections and open problems.

Key words: Newtonian kinematics, hypercomputation, infinite parallellism

[1] Email: e.j.beggs@swansea.ac.uk
[2] Email: j.v.tucker@swansea.ac.uk

1 Introduction

Computability theory, founded by Church, Turing and Kleene in 1936, is a deep theory about the functions computable by algorithms on discrete data, and it has been extended to continuous data via approximations. At its heart is the concept of an *algorithmic procedure* which is based on various models of systems that process discrete data. It also contains subtheories about non-computable functions such as (i) *arithmetic definability*, based on classifying functions by their definability in first-order logical languages and (ii) *degree theory* based on various notions of relative computability. These theories of the non-computable are absolutely essential because they classify a huge number of interesting computational problems. One need only ask what is the complexity of deciding the set of all true first-order statements about natural number arithmetic to be introduced to the full power and beauty of hierarchies and degrees.

Hyper-computation is about making models of systems that are capable of computing functions that are *beyond* those computable by algorithmic procedures. There are many proposals for such hyper-computers, being models based on different kinds of mathematical, physical, and biological theories. But the entry of the rough and troublesome behaviour of physical theories into the elegant ordered world of classical computability causes disruption and conflict. The physical validity of the models are disputed, and the mathematical, physical and philosophical basis of the subject is messy and controversial. In particular, there are several examples of inadequate mathematical analyses and flaws in the interpretation of mathematical results. To take an early example, the intriguing work of Pour El and Richards (31; 32) on the wave equation is directed at answering the questions of Kreisel (19) about the computability of physical systems. Although mathematically sound, the theorems cannot support Gedanken experiments that show there exists a theoretical wave machine that is a hyper-computer. In fact, the revision of Weihrauch and Zhong (44) confirms the opposite (see the discussion in (1)). To take another example, the critical analyses Davis (9; 10) dismantles the claims of Siegelmann (35) and goes on to discredit the search for a theory of hypercomputation. Choosing from among several surveys, speculations and philosophical discussions, see Cooper and Odifreddi (4), Cotogno (5), noting Welch (45), and Ord (27).

We do not know how to construct a theoretically plausible hyper-computer. Moreover, we do not know how to conduct an investigation into the subject. Despite these technical and methodological problems, the novelty and diversity of new computational ideas coming from computer science, physics and biology are breathing new life into the foundations of computability theory and giving it a wider vision and new prospects for

(i) models, suggesting new methods and paradigms for computation;
(ii) mathematics about the undecidable and non-computable; and
(iii) hard questions about the physical and biological foundations for computational models.

We believe that hyper-computation is a useful and necessary concept that can help enrich and integrate physical theories with classical computability theory - which contains a wealth of beautiful results about non-computable sets and functions already.

In this paper, we consider aspect (iii) of hypercomputation and propose a methodology that may help in developing the idea and role of hyper-computation. Specifically, we consider hyper-computation that is dependent on the idea of computing a function using an experimental procedure on physical system. We propose to examine this general idea, which we call *experimental computation*. This is relevant to a great deal of the literature, for hyper-computers are often proposals for such systems. However, such proposals are often imprecise, especially in stating the physical assumptions that underly them and the methods needed for their use in computation. It seems to us that it is necessary to establish much greater precision in defining models, and rigour in reasoning about their behaviour. It also seems to us that it is necessary to clarify some principles and make explicit a methodology for struggling with these difficult and yet exciting ideas at the interface between physical theories and classical computability theory. There is a need for a neutral intellectual space, free of misunderstanding and prejudice, where the mathematical results can simply flourish and their interpretations can be debated with care and discipline.

In summary, our approach is simply to organise the mathematical study of problems, such as the following:
Classifying computers and hyper-computers in physical theories: *To define precisely a subtheory T of a physical theory and examine the systems that are models of T that can be said to be computers and hyper-computers.*
Classifying the border between computer and hyper-computer in physical theories: *To analyse what properties of the subtheory T are the source of non-computable behaviour and seek necessary and sufficient conditions in terms of T for the systems to implement precisely the algorithmically computable functions.*
Some of the literature fits this programme, or can be made to fit following more precise analyses of the proposals. The work of Smith (36) on the N-body problem fits the program particularly well. The work of Hogarth (16; 17) and Nemeti and David (25) on hyper-computers embedded in General Relativity, and Kieu (18) on hyper-computers embedded in quantum physics, also comes to mind.

We started developing the methodology in some earlier papers (1; 2) in which we constructed *simple* examples of systems that are valid models of Newtonian kinematics and also hyper-computers. Specifically, we showed how to embed arbitrary sets and functions into equipment valid in subtheories and develop necessary and sufficient conditions for the systems to be computers: see Section 2.3. In each case, the aim is to explore the computational nature of a fragment of the physical theory.

Secondly, after discussing methodology, we embed a model of a hyper-computer into a fragment of Newtonian mechanics. The model is an infinitely parallel computer. By an *infinitely parallel computer* we mean an infinite network of processors computing and communicating via some channels, busses or other media. Such machine models can become hyper-computers. In our case, we use its infinite architecture to decide quantifiers and hence arithmetic sets. In particular, we prove:

Theorem 1.1. *There exists an infinitely parallel machine M in 3-dimensional space, with a Newtonian communication system and single global clock, whose observable behaviour decides any arithmetically definable set A of natural numbers. The system has infinitely many identical component machines all running the same program. The system can decide any arithmetical sentence*

$$\phi = (Q_1 x_1) \ldots (Q_n x_n)\ f(x_1, \ldots, x_n),$$

where the $Q_1, \ldots, Q_n$ are quantifiers and f is quantifier-free, in $T(f) + n.r + t$ steps, where $T(f)$ is the number of time steps taken to evaluate f in a component and r, t are constants.

By way of illustration of the architecture we show how the infinite machine could compute the Twin Primes problem and solve Hilbert's Tenth Problem.

The structure of the paper is this. In Section 2 we discuss the nature of experimental computation and the aims of a programme to analyse theoretically the physical foundations of computation and hypercomputation. We also discuss examples of embedding computational notions in physical theories. In Section 3 we describe the fragment of mechanics we will use to embed an infinitely parallel machine. In Section 4 we describe the architecture, processing units, behaviour, and construction of the infinite machine. In Section 5 we look at the Twin Primes Problem and prove the main theorem, and give an application to Hilbert's Tenth Problem. In Section 6 we summarise the physical features of the machine. Finally, in Section 7 we discuss some open problems.

This paper is a companion to Beggs and Tucker (1; 2), which contains complementary results and a literature survey. Technically, only elementary Newtonian mechanics is needed to follow our arguments. However, some knowledge of the different approaches to the formulation of the problem and its "solution" is necessary. It is helpful if the reader is familiar with (1) and with the theory of the functions computable by algorithms on discrete data, and its extension to continuous data (see: (32; 38; 40; 41; 43)).

2 Experimental computation and physical theories

Here we discuss a programme for investigating computation and hypercomputation by physical systems.

2.1 Experimental computation

Consider the idea of computing a function using a physical system. Suppose each computation by a physical system is based on an experimental procedure with three stages:

(i) input data x are used to determine initial conditions of the physical system;

(ii) the system operates for a finite time; and

(iii) output data y are obtained by measuring the observable behaviour of a system. The function f computed by a series of such experiments is the relation $y = f(x)$. We can expect to compute functions on continuous data, such as the set $\mathbb{R}$ of real numbers, as well as functions on discrete data, such as on the set $\mathbb{N}$ of natural numbers.

This idea of *experimental computation* is general and old. It depends upon a choice of

(a) physical system, which we may think of as a part of nature, a machine, or a combination of both, that performs the computation, and of

(b) a choice of physical theory, which we use to reason about the system's behaviour and the computation.

The concept can be unearthed in modelling physical and biological systems, e.g., in classical wave mechanics (31; 32; 44; 39) or in the emergent behaviour of cellular lattices and neural networks (15; 35; 46). It can also be found in technologies for designing machines, e.g., analogue computers (29; 20; 14; 42) or quantum computers (26). We have classified and surveyed the literature in our (1). The questions arise:

What are the functions computable by experiments with a class K of physical systems? How do they compare with the functions computable by algorithms?

Do there exist physical systems in the class K that qualify as hyper-computers?

We argue that, in contrast to algorithmic computation, the idea of experimental computation is not understood. At its heart are the concepts of *experimental procedure*, which is closely related to that of algorithmic procedure, and *experimental equipment* both of which are dependent upon a physical theories. We have discussed this in (1; 2).

2.2 Role of Physical Theories

Physical theories play a complicated role in experimental computation. Ideally, in formulating answers to the questions in subsection 2.1, we should use a physical theory to

(i) define the class K of physical systems under investigation;

this means that the concepts and laws of the theory will be used to

(ii) design and construct the systems in K;

(iii) explain and prove properties of their operation and behaviour; and, hence,

(iv) validate experimental computations by the systems of K.

For example, a physical theory is needed to design, explain and validate a Newtonian system capable of simulating a Turing machine. Furthermore, if a hyper-computer has been found in K then we should use the theory to

(v) evaluate the physical credibility of the system, or even

(vi) reveal weaknesses in the theory.

Finally, in each case, we should use the theory to

(vii) make clear, precise and detailed statements about the whole process of experimental computation, especially the experimental procedure and equipment.
It is clear that these simple desiderata are not always all present in the literature on hypercomputation.

We believe it is the physical theory itself that is the central object of theoretical study in investigations of experimental computation and the questions above. Here we propose four aspects or stages to an investigation of a particular type of experimental computation.

1. Defining a physical subtheory: *To define precisely a subtheory T of a physical theory and examine experimental computation by the systems that are valid models of the subtheory T.*

All physical theories are large and evolving, though the classical theories usually have an elegant mathematical form and can be organised incrementally. In some cases they have been axiomatised informally and, sometimes, even formally. Clearly, it is not easy to define exactly such a theory. Therefore, one might begin with some subtheory that is quite large, for example the Newtonian kinematics of point particles, which is a subtheory of Newtonian Mechanics. Conceptual precision and mathematical rigor will surely reduce the subtheory into something *much* smaller. For suppose one finds an interesting system C that is a computer or hyper-computer and a model of the physical theory. Then the problem arises:

Minimal Subtheories *In computing with a physical system C, specify the smallest subtheory or fragment T of the physical theory that is needed to construct, operate, reason about, and validate C.*

The subtheory T is the source of the definition of experimental computation as follows. What is contained in T determines the set of possible experimental procedures and what form the systems have in terms of their components and architecture. In (1) we discussed how to derive from a theory T languages to specify formally both procedures and equipment.

For the purposes of a theoretical programme the idea is to look at theories that are small and rather precisely defined. *It does not matter whether we think of them as true, or roughly applicable, or know them to be false.* For example, in our work on Newtonian systems it is known that assumptions about space and time are false of our universe. In our view, that in no way diminishes the theoretical interest and value of seeking Newtonian hyper-computers. Success and failure in the search can provide insight into the nature of computability in a physical context and in terms of Newtonian principles.

2. Classifying computers in a physical theory: *To find systems that are models of T that can through experimental computation implement specific algorithms, calculators, computers, universal computers and hyper-computers.*

Subtheories do not contain notions of computation, though the term 'information' is common in some theories. The computation of functions and sets must be abstracted from the operation of physical systems along the lines mentioned in 2.1. For the purposes of comparison with algorithmic procedures and, especially for the search for hypercomputation, there is a natural course of action, namely study the mathematical problem:

Embedding computations in a physical theory: *To seek ways of embedding functions, sets, logical formulae, algorithms, programs, models of computers and hyper-computers, etc. into some physical systems that satisfy a subtheory T.*

Clearly, as with any embedding problem, we are interested in embedding *complex computations* into *simple systems* specified by *simple subtheories.* For example, to answer the question do there exist Newtonian systems that compute everything, one can embed *arbitrary sets* into valid 2-dimensional single particle systems obeying a few simple Newtonian or Relativistic laws (see 2.3) In this paper, we will embed an infinitely parallel computer into Newtonian mechanics.

The idea is to play with computational ideas and their emdeddings into physical theories to explore the interface between algorithms and experiments and to see what our trickery can get away with and what it can uncover. In our experience, and that of others, the embedding of non-computable sets, functions, etc., leads to systems where the hyper-computations are courtesy of *deus ex machina.* For this stage we should not care about the validity of the physical theory but we should care about making theoretically complete models and precise mathematical analyses.

3. Mapping the border between computer and hyper-computer in physical theory: *To analyse what properties of the subtheory T are the source of computable and non-computable behaviour and seek necessary and sufficient conditions for the systems to implement precisely the algorithmically computable functions.*

It is an intriguing and complicated problem to isolate what properties of a physical theory permit computers and hyper-computers. In the case of systems built from embeddings it may be clear where some infinite computational resource is to be found. Thus, if we constrain that resource we may be able to recover computability. In our Newtonian work we have isolated some necessary and sufficient conditions on the subtheories that can control the embedding of sets of natural numbers, but they amount to algorithmic conditions on mechanical equipment that are new and external to classical mechanics. This is a tricky point for the conditions ought to be framed in terms of the physical theory. For example, for certain arguments it is not acceptable to assume that standard models of calculators and computers are simply available for use in experimental computation. Strictly speaking they are external or foreign to the physical theory T.

Calculations: *Which of the standard models of calculators can be implemented in the T?*

It is not clear that a Turing machine can be faithfully embedded in many physical theories: see Davies (8).

4. Reviewing and Refining the Physical Theory: *To determine the physical relevance of the systems of interest by reviewing the valid scope or truth of the subtheory. Criticism of the system might require strengthening the subtheory T in different ways.*

We have emphasised the autonomy and independence of the physical theory for we take it and its subtheories to be the focus of the mathematical work. In addition to providing intellectual pleasure, a physical theory is supposed to be true of some aspect of the physical world. Of course, a physical theory cannot be proved, but it can be validated by experimental facts. For any theory T there are restrictions that defines that part of the theory that has been validated experimentally. These restrictions can be seen as a refinement theory $E(T)$, called the *experimental domain* of T. For any valid model C of a theory T, we may ask:

Experimental validity: *Is C a valid model of the experimental domain $E(T)$ of T?*

The process of reviewing examples can lead to the refinement of the subtheory in different ways, perhaps adding more laws in the search of more realism - e.g., what happens if we add friction and/or elasticity to the Newtonian system. And it can lead to the rejection of the subtheory and its systems as a basis for making a hyper-computer - e.g., arbitrary velocities may be needed, which are not possible. Often, as one moves out of $E(T)$ into T one contradicts other theories. In Newtonian kinematics, using arbitrary velocities contradicts Relativity Theory, and making arbitrary small components contradicts Atomic Theory. In our work, we have worked with the full, unrestricted, kinematic theories, ignoring the experimental domain, and have concentrated on mathematical rigour, consistency and completeness.

2.3 Example: Experimental Hypercomputation with Newtonian Kinematics

Newtonian Kinematics is about systems of particles in n-dimensional space ($n = 1, 2, 3$) satisfying Newton's Laws of Motion and laws such as Gravitation and Conservation of Potential and Kinetic Energy. Typically, subtheories describe systems that allow the

(a) projection of particles in space with arbitrary velocity, and

(b) observation of the position of particles at certain times.

A subtheory of Newtonian kinematics may further assume that the space may be structured in different ways, influenced by gravitational fields, populated by special bodies and obstacles, and shaped by geometries. It may also suppose that materials have different physical properties such as friction and elasticity. Newtonian kinematics has been used in many discussions of computation and hypercomputation such as Moore (23), da Costa and Doria (6; 7), Yao (47).

The work of Smith (36) is particularly close in spirit to the framework described above: his portfolio of theories in which he studies the N-body systems addresses the issues of Problem 3 above. Here the infinite computational resource comes from the arbitrarily high velocities attained in non-collision singularities in the n-body system, and the study of 'topologically' different trajectories which can be used to achieve this.

We will illustrate the four stages of exploring hypercomputation in Newtonian Kinematics by some of our earlier results in (1; 2). These results pursue the idea of embedding arbitrary sets and functions into systems satisfying very simple Newtonian subtheories. In this process we seek:

(i) simple kinematic subtheories to isolate physical assumptions;

(ii) transparent embeddings so we can take apart and inspect the systems;

(iii) systems that are hyper-computers;

(iv) necessary and sufficient *physical* conditions for the sets and functions to become computable and the systems to become computers.

In our (1), we showed there exist kinematic systems, which operate under the theory of Newtonian kinematics - and, in fact, Relativistic kinematics - that can decide the membership of *any* subset A of the set $\mathbb{N} = \{0, 1, 2, \ldots\}$ of natural numbers. The systems were 2 dimensional bagatelles with single particles. However, the systems needed *unbounded* space, time and energies for their hypercomputations.

In our (2), we "improved" on these bagatelle machines by constructing new 3 dimensional Newtonian machines that are marble runs that each require *bounded* space, time, mass and energy to decide $n \in A$ for all n. These complementary results involve the assumption that space can be infinitely divided into smaller and smaller units, a valid assumption in Euclidean geometry and Newtonian mechanics not used on the bagatelles.

The two types of system are designed by means of embeddings. The proofs embed *any* set A into the static structure of a simple piece of Newtonian kinematic equipment such that a simple experimental procedure, based on the projection and observation of a particle, can recover the set A from the given equipment.

Clearly, the theorems show Newtonian hyper-computers "exist" in a dramatic way: every set and function on $\mathbb{N}$ is computable by *simple* Newtonian kinematic systems, indeed, can be represented by *the rolling of a ball in a straight line*. Given the Church-Turing Thesis, this raises the question, *Is the elementary theory of Newtonian kinematics is undesirably strong?*

Now, the purpose of the embedding theorems is to explore concepts in physical theory relevant for computation. In our results, we find that the idea of experimental computation involves much more than an experimental procedure applied to a black box containing an arbitrary given physical system. Crucially, it involves the

(i) application of a sequence of *experimental actions* to the marble run determined by a subtheory of Newtonian kinematics, and

(ii) *process* of *assembly* or *construction* of the equipment.
Our results on hypercomputation demonstrate that the idea of equipment is problematic in Newtonian mechanics. Physical conditions on the architecture of the system must be found to control the class of Newtonian equipment.

In (2) we propose a new approach of using languages like programming languages to analyse both experimental procedures and construction procedures. Our results suggest that a refinement of mechanics is needed in which we can define classes of *constuctible equipment*, and which would allow us to restrict the subsets A of $\mathbb{N}$ in the marble run. The design of such languages is complicated. We argue that using such languages to control procedures and equipment, the functions computed by experimental computation are equivalent to those computed by algorithms, i.e. the partial computable functions.

Our idea of analysing experimental computation using experimental procedures and constructible equipment and programming languages for defining them seems to be new. It advances theoretically the informal reflections of Geroch and Hartle (13) on the computability of measurements in physical experiments.

2.4 Example: Experimental Hypercomputation with General Relativity

General relativity uses a curved space time manifold (with three space and one time dimensions) as a model for the universe; we call such a manifold a *space-time* for short. The Einstein equation relates the curvature of the manifold to the amount of matter and energy in the universe via the energy-momentum tensor. We can either try to specify the energy-momentum tensor (for example by taking the vacuum) and then solve the Einstein equation, or specify the space-time, and then work out what the energy-momentum tensor would have to be in retrospect. Time is measured along the space-time path (world line) of each observer, and is thus not a universal absolute, in distinction to Newtonian mechanics.

In Special and General Relativity, moving clocks appear to run slow, and someone in a spaceship could start a computation, take off and travel for a year round Alpha Centauri at a velocity $v < c$, and return to see the printout of 40 years of computation (the 'twin paradox'). So, to attain hypercomputation, it is "only" necessary to extend the 40 years to an eternity, and return to see the printout of an *entire* infinite computation.

To implement this idea, people have resorted to some examples in General Relativity, where the geometry of the space-time allows such effects. The basic idea is that an observer at a certain point in space-time can see the entire infinite future of a certain path in space-time, and those geometries which have this property are called *Malament-Hogarth space-times.*

The idea of a theory General Relativistic (GR) computers is recent and can be found in two independent schools. The possibility of hyper-computation in General Relativity is raised in Hogarth (16), based on earlier observations of Pitowsky (28), and Malament. GR-computation was also considered in a lecture of Németi in 1987 and was developed in work culminating in the (Los Alamos version) of Etesi and Németi (12) and recently Németi and David (25) (see the references for further information). For further discussion, see Shagrir and Pitowsky (34).

In (16) there is a great deal of technical material about how this can be realised in general relativity, and the interested reader should consult this, but we will try to explain some of it using the anology given there of the Tardis from the Dr. Who science fiction series, a machine which looks like an ordinary box on the outside, but its interior volume is huge. It is possible to have *space-times with bounded regions in both space and time (as their surface is seen from the outside), which contain infinitely long world-lines.* In other words, the Tardis could exist for only a day in external time, but someone entering it could wander forever around its passages. An observer who stayed outside the Tardis could see the results of an eternity of calculation done by the person who entered the Tardis, just by waiting for a day. The analysis of different space-times is an exploration of subtheories of General Relativity.

The article Németi and David (25) illustrates the progressive refinement of ideas of physical theories towards more pleasing, real or practical theories. They begin with using a 'usual' black hole to give a rough argument about how to achieve hyper-computation. Then they explain why it will not work, in that case that the observer will not survive the observation of the result. This then motivates a refinement of the theory, by including the idea of a rotating black hole, and they show that this is a much more promising candidate for realistic hypercomputation.

These physical speculations on the universe will doubtless effect theoretical computer science for the ideas are too fascinating to be neglected. For example, one notes the geometric model in (11) uses an analogy with Black holes.

3 Parallelism and a Newtonian Kinematic Subtheory

Shortly, we will embed or realise an infinitely parallel machine - a hyper-computer - in a Newtonian system. The scope of the term “Newtonian mechanics” is huge and includes a lot of concepts and results, indeed it is open ended. We shall begin by highlighting those few pieces of the theory we need for our embedding. This amounts to an informal specification of a subtheory or fragment T within which we will work.

To realise an infinitely parallel computer we will require the following:

1. A *communication system* which transfers information at arbitrarily large, but finite, speeds. To do this we will use *cannons and projectiles.*

2. A *scheduling system* for the operations that uses arbitrarily accurate *clocks*, though no individual clock needs to be infinitely accurate. To do this we will use Newton’s *infinitely divisible universal time.*

3. A method to simultaneously *transfer information* to infinitely many processors to program the machine. To do this we will use *infinitely long rigid rods.*

Let us look at these mechanical ideas in turn.

Cannons & projectiles. The relevant part of Newtonian mechanics is *force F equals mass m times acceleration a*, $F = ma$. Given a projectile of given mass, we can project it at arbitrarly high speeds if we use enough energy. This means that arbitrarily large distances can be covered in arbitrarily small time. In the simplest version of the infinitely parallel machine, where each cannon has a pre-assigned target, we know before hand just how much gunpowder (energy) each gun will require to complete a communication.

Gravity does not pose a problem, in fact it is easier to arrange for unobstructed projectile paths if we can fire above the plane in which the cannons and targets lie. In the case of a uniform gravitational field, the flight paths will be parabolic, and we always choose the lower of the two possible paths for given speed and distance to target to get a fast flight time.

Projectiles & targets. We need a target which will determine if it has been fired at, when infinitely many cannons may fire at it in a given time. To avoid having an infinitely sensitive trigger on the target, we take all projectiles to have the same mass. An infinitely sensitive trigger mechanism could easily spontaneously trigger due to brownian motion of its component atoms. This will, of course, mean that guns firing further away will take greater amounts of gunpowder. In the simplest scenario where we only need to determine whether a target has been hit, we will favour the famous "wave a hat on a stick test" - if the hat is damaged, then it was hit by a projectile. One problem to avoid is what was known in Cold War parlance as *fratricide*, that is one projectile might destroy another. To avoid this we can interleave the travel times so that at most one projectile is in flight at a time This tight regulation of travel times uses another aspect of Newtonian dynamics, the arbitrarily small divisibility of time, as we require an infinite number of disjoint time periods, within finite total time.

Consider the more complicated scenario where a *message* has to be sent with the projectile. This can be catered for by a trapdoor arrangement where the first projectile to hit enters the trapdoor and closes it behind it. The contents of the projectile can then be examined. The later projectiles hit the trapdoor and are ignored. This is similar to the machines used by mail trains to recover mail bags while in motion. (They were withdrawn amid safety concerns, but since the infinite machine already involves arbitrarily large amounts of gunpowder, safety is not high on our priority list!)

Clocks & timings. We assume that every component machine has its own clock, which can be read to a given accuracy (the accuracy needed may vary from machine to machine). To synchronise the clocks on all the machines we can use the same rigid rod which is used to program them.

Rigid bodies. We will use the Newtonian ideal of a completely rigid body, that is one whose shape and size is unaltered despite the application of forces. This is an unattainable ideal in the real world, as all real objects can be deformed by the application of a sufficient amount of force.

Notice that, like our use of arbitrary velocities, arbitrary long rigid rods also are inconsistent with Special Relativity, as a perfectly rigid body would transmit the results of a hammer blow at infinite speed. To see this, note that a perfectly rigid body has its dimensions constrained to be constant, so a movement of (for example) one end of a rod means that the other end also moves by the same amount *at the same instant*, even if the rod were a light year long.

Infinite rods. We further require the idea of a rigid rod that is *infinite*. Does this fall outside the scope of Newtonian Mechanics, for we may ask how could such a thing be moved? If the rod had a positive constant mass per unit length, then its total mass would be infinite, and it would require an infinite amount of force to accelerate it.

However, there are at least three possible solutions to this, the first of which is probably the nicest to consider:

(i) Make the rod non-uniform and decreasing in diameter. Specifically, suppose that the first metre of the rod had diameter d, the next metre diameter $d/2$, the next $d/4$, and the n-th metre of diameter $d/2^{n-1}$. Then *the infinitely long rod would have finite volume, and thus finite mass if made of a uniform material.*

Two cheaper theoretical assumptions are these:

(ii) Declare the rod to be massless. Although possible, this somewhat strains the idea of even idealised Newtonian mechanics.

(iii) Provide booster motors along the way. This is a rather complicated solution. The motors would be provided for each one metre length of the rod, and would provide enough force to move that one meter length. One problem with this is synchronisation, how to get the motors to work at the right time. If the rod actually consisted of separate one metre long sections, then each motor could watch for the movement of the previous section to see if its section had to be moved. If it could not react instantaneously, then a progressive speeding up of the reaction time for each motor would be required to allow the whole infinite length to be moved in finite time.

4 Embedding an infinitely parallel machine in Newtonian Kinematics

Our infinitely parallel machine is rather complicated. First, we will describe its architecture using infinitely many dimensions. In contrast, the components of the machine are just conventional digital computers or, depending on the application, calculators. Then we describe the encoding of its layout in $\mathbb{R}^3$.

Consider the set L of sequences of integers $(n_1, n_2, \dots)$, which consist of a finite sequence of natural numbers followed by infinitely many -1s. For example $(0, 3, 2, -1, \dots)$, $(-1, \dots)$ and $(7, -1, \dots)$ are in L, but $(0, -1, 5, -1, \dots)$ and $(6, 7, 0, \dots)$ are not. We write

$$\underline{n} = (n_1, n_2, \dots, n_{max}, -1, \dots)$$

where $n_1, n_2, \dots, n_{max} \in \mathbb{N}$. For each element $\underline{n}$ of L we have a component machine or node $C_{\underline{n}}$, consisting of a computer and an input/output communications channel or bus. The vector $(n_1, n_2, \dots, n_{max})$ we term the *address* of the component. The processors can be thought of as organised into ranks, where the *rank* is the number *max* of natural numbers preceeding the first -1 in $\underline{n}$, i.e. largest value of i for which $n_i \neq -1$. A picture of ranks 0-2 of the infinite machine is given in Fig. 1.

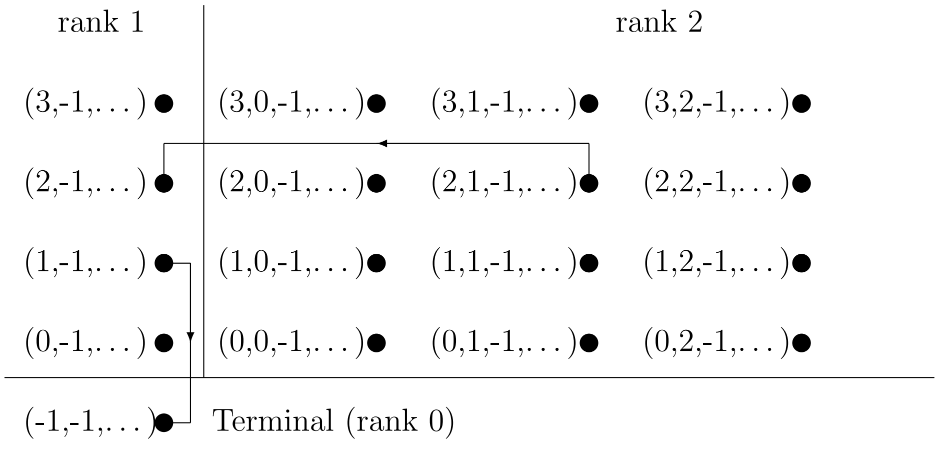

Fig. 1 The beginning of the infinite machine

The third rank can be visualised as sitting above the paper, and the higher ranks require countably infinite dimensions to define. (Remember that this is just a convenient way to "visualise" the architecture; the physical layout will only require three spatial dimensions.) Fig. 1 also shows a couple of typical communications in the machine.

Notice the self-similarity in the architecture: the repeating unit consists of the rank 0 (the terminal) and 1 nodes. To add on the rank 2 nodes, we can consider each rank 1 machine as a terminal and add on a new row of 'rank 1' machines to it. Similarly we can add to each rank 2 to get rank 3, and so on.

4.2 Component Processors

The machine $C_{(-1,\ldots)}$ of rank 0 is the terminal machine, the only one which communicates output to the outside world.

The component machines or nodes (except for the terminal machine $C_{(-1,\ldots)}$) are identical in all respects except in the communication system and the number of registers required to hold their addresses:

Definition The component machine $C_{\underline{n}}$ is a register machine, with:

(a) A register *max* containing the rank of $\underline{n}$.
(b) Registers containing the machine's address, i.e. the numbers $n_1, n_2, \ldots, n_{\max}$.
(c) Basic operations to carry out addition $+$, subtraction $-$, multiplication . and tests of equality $=$ and order $<$ on arbitrary length integers in one time step.
(d) A stored program P, identical in all machines.
(e) A channel or communication bus output, which can output a single integer in one time step to its *supervisor*, the processor $C_{\underline{m}}$, where $\underline{m} = (n_1, n_2, \ldots, n_{\max-1}, -1, \ldots)$.
(f) A channel or communication bus input, which can receive a single number in one time step from its *underlings*, the processors $C_{\underline{m}}$, where $\underline{m} = (n_1, n_2, \ldots, n_{\max}, r, -1, \ldots)$ for all $r \geq 0$. In the event of simultaneous communication from several of these underlings, the input from the machine with the least value of r is taken, and the rest are ignored. A physical mechanism for this is discussed above involving the arbitrary divisibility of time.

Using the nodes and addresses we can state more clearly the following important *self similarity property* of the architecture: Every processor in the machine is a supervisor to a set of underlings, and can be considered as the terminal component machine of its own infinite machine whose rank 1 processors are its underlings; its printout as a terminal is in the form of a message to its own supervisor.

4.3 Physical Layout of 3-Dimensional Machine

The component $C_{\underline{n}}$ is placed along the x-axis at the position

$$2^{n_1+1}3^{n_2+1}5^{n_3+1}\ldots {p_{max}}^{n_{max}+1}$$

determined by the usual prime power product.

We can assume that a component machine fits in a box of length one along the x-axis and of arbitrarily large extent in the $y \in \mathbb{R}$ and $z \leq 0$ directions.

The space where $z > 0$ is reserved for the inter-component communications (i.e. the trajectories of the cannon balls). Here is a view of the device in $\mathbb{R}^3$

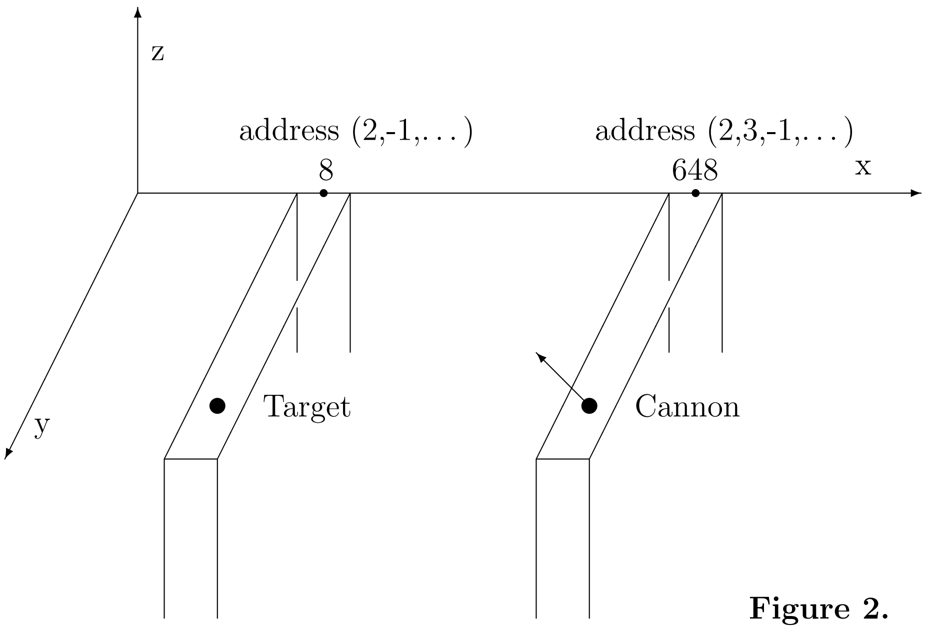

Figure 2.

With this layout, we need one infinite rigid rod along the x-axis to program the machine.

4.4 Communications and Programming

For communications we use the fact that arbitarily large velocities are allowed in Newtonian mechanics. Suppose each machine has a gun on top, which is aimed at a collecting basket on its target machine. In this paper we shall only be concerned with whether a message is sent or not, so we may as well use solid cannon balls. (However the reader can consider a variant infinite machine where the cannon ball is hollow and contains the contents of a register on the source node.)

We have claimed that the machine is fully programmable, and that it has time steps which are of equal length for all machines. The infinite rigid rods are needed if the machine is to be programmed. The synchronisation pulses are transmitted to all the nodes similtaneously using a rigid rod.

The program is initially distributed to each machine using a rigid rod linking all the machines. One way of coding this would be to use a sort of Morse code, with a short push for '0' and a long push for '1'. We would need every machine to have an (identical) protocol for interpreting motions of the rods - it would contain the codes for 'begin program', 'end program' and 'synchronisation pulse next'.

5 Examples and Theorems

Why can this fully programmable machine calculate more than a Turing machine? To begin with, it is not that obvious that it can calculate more than a Turing machine! To better understand the machine we first describe a calculation. Then we show that the machine decides the sets of the arithmetic hierarchy and so is a hyper-computer. We also give an application.

5.1 Twin Primes

A twin prime is a pair $(p, p+2)$ where p and $p+2$ are both prime. For example $(3,5)$, $(5,7)$, $(11,13)$ and $(17,19)$ are the first few twin primes. Despite extensive investigation, it is not known whether there are infinitely many twin primes or not.

Suppose that each component machine is equipped with an *oracle of primality* $\mathcal{P}$. This is a device which can determine if any input natural number is prime or not in a fixed number clock cycles - say, it takes 100 cycles. This oracle assumption is not really needed, as we can design an oracle of primality within the unaugmented infinite machine, but it is convenient for this illustration.

We only need the component machines of rank 0, 1 and 2, with addresses $(-1,\dots)$, $(n,-1,\dots)$ and $(n,m,-1,\dots)$ for $n,m \in \mathbb{N}$; all other machines do nothing and terminate.

The idea is to search all the numbers for twin primes in an enumeration of the form $n+m$.

For each $n,m \in \mathbb{N}$, the component machine $C_{(n,m,-1,\dots)}$ asks its oracle of primality whether $n+m$ is prime. If not, it terminates. If $n+m$ is prime, it asks its oracle of primality whether $n+m+2$ is prime. If not, it terminates. If $n+m+2$ is prime, it sends a message (content not important) to its supervisor $C_{(n,-1,\dots)}$, and then terminates. Suppose the total time taken for these two oracle calls, processing and communication is say $100 + 100 + 50 = 250$ cycles.

The supervisor component machine $C_{(n,-1,\dots)}$ does nothing except wait for any received messages for the first, say, 255 cycles. If it receives any messages, it terminates. If not, it sends a message to its supervisor the terminal $C_{-1,\dots}$, and then terminates. Suppose the total time taken is, say, 260 cycles.

The terminal $C_{(-1,\dots)}$ does nothing except wait for received messages for the first 265 cycles. After that time, if it received no messages, it prints "There are infinitely many twin primes", and terminates. If it received any messages, it prints "There are only finitely many twin primes", and terminates. Suppose the total time taken is, say, 300 cycles.

The reason this works is the following: each rank 1 component machine $C_{(n,-1,\dots)}$ receives a message only in the case where there is a twin prime $(p, p+2)$ for $p \geq n$, and so sends a message to the terminal $C_{(-1,\dots)}$ only in the case where there are *no* twin primes $(p, p+2)$ for $p \geq n$. Therefore, if the terminal $C_{(-1,\dots)}$ receives any message at all within the 265 cycles, it must be the case that there is an n with no twin primes $(p, p+2)$ for $p \geq n$, i.e. there are only finitely many twin primes.

5.2 Arithmetic Sets

Now we consider a general case of computing with the infinite machine. Recall the arithmetically definable sets of natural numbers.

Definition 5.1. *Let $\Sigma = \{1, 0, +, -, \times, =, <\}$ be a signature for the natural numbers. Then the class of arithmetic formulae $\mathcal{F}$ is defined inductively by*

Equality *If t and t' are terms over Σ, then $t = t'$ is a formula.*

Inequality *If t and t' are terms over Σ, then $t < t'$ is a formula.*

Not *If $f \in \mathcal{F}$, then $\neg f \in \mathcal{F}$.*

And *If $f_1, f_2 \in \mathcal{F}$, then $f_1 \wedge f_2 \in \mathcal{F}$.*

For all *If $f \in \mathcal{F}$ and x is a variable, then $\forall x\ f(x)$ is a formula.*

Exists *If $f \in \mathcal{F}$ and x is a variable, then $\exists x\ f(x)$ is a formula.*

In what follows we make use of timing and the self-similarity property of our machine.

Time plays a key role in the operation of our infinitely parallel machine model and, in particular, we must be careful about the time taken by a component processor C to compute a quantifier-free formula f, given values of all its variables.

Suppose that we want to calculate $f(x_1, \ldots, x_n)$, where the value of the variable x_i is contained in register r_i in the processor C. We will require that the time taken is $\leq T(f)$ independently of the values of the variables. This property can be proved by induction on the structure of the formula using clause (c) of the definition of component machines in Section 4.2.

Of course, this is not true of, say, an n bit binary machine. If very large integers were to be added together, they would have to be added in n bit lengths, and the carry numbers taken to the next n bit addition.

We also make use of the self-similarity property of the infinite machine: Every processor in the machine can be considered as the terminal component machine of its own infinite machine; its printout is in the form of a messsage to its own supervisor.

Theorem 5.2. *All arithmetic statements (i.e. formulae with no free variables) in 5.1 are computable by the infinite parallel Newtonian machine running the same program in all components.*

Suppose $\phi = (Q_1 x_1) \ldots (Q_n x_n)\ f(x_1, \ldots, x_n)$*, where each* Q *is either* $\forall$ *or* $\exists$*, and where* $f(x_1, \ldots, x_n)$ *is a quantifier free formula in* n *variables. Let* $f(x_1, \ldots, x_n)$ *be calculated in* $T(f)$ *time steps, independently of the values of the* n *variables. Let each processor check whether it received an input, and send an output based only on this information, in constant* r *time steps. Let the terminal check whether it received an input, and print an output based only on this information, in constant* t *time steps. Then an upper bound for the time taken for this computation is* $T(f)+n.r+t$ *steps. Further the machine uses only components of rank* $\leq n$

Proof. It is known that all arithmetical statements can be put in the above form of ϕ by the Tarski-Knaster algorithm (see, e.g., Rogers (33)).

For clarity, we describe a particular case of arithmetical formula (a Σ_3 sentence). Suppose that we have

$$\exists x_1\ \forall x_2\ \exists x_3\ f(x_1, x_2, x_3)$$

Then:

1. The component processor at node $(x_1, x_2, x_3, -1, \ldots)$ calculates $f(x_1, x_2, x_3)$ while the other machines wait - this needs $T(f)$ time steps.

2. The component processor at node $(x_1, x_2, x_3, -1, \ldots)$ sends a message to is

supervisor $(x_1, x_2, -1, \dots)$ and stops if its calculation had value *true*, otherwise it just stops - this needs r time steps.

3. The component processor at node $(x_1, x_2, -1, \dots)$ determines if it received a message or not. If it did *not*, it sends a message to its supervisor $(x_1, -1, \dots)$ and stops, otherwise it just stops - this needs r time steps.

4. The component processor at node $(x_1, -1, \dots)$ determines if it received a message or not. If it did not, it sends a message to $(-1, \dots)$ and stops, otherwise it just stops - this needs r time steps.

5. The terminal processor at node $(-1, \dots)$ determines if it received a message or not. If it did, it prints "the statement is true", otherwise it prints "the statement is false" - this needs t time steps.

The total running time is $T(f) + 3\,r + t$ time steps.

Now we analyse the result of these operations stage by stage, starting with the output of the terminal:

The terminal prints "the statement is true" $\Leftrightarrow \exists x_1 \in \mathbb{N}$ such that the node $(x_1, -1, \dots)$ sends a message.

For the rank 1 processors we have:

the node $(x_1, -1, \dots)$ sends a message $\Leftrightarrow$ it is not the case that $\exists x_2 \in \mathbb{N}$ node $(x_1, x_2, -1, \dots)$ which sends a message

For the rank 2 processors we have:

the node $(x_1, x_2, -1, \dots)$ sends a message $\Leftrightarrow$ it is not the case that $\exists x_3 \in \mathbb{N}$ node $(x_1, x_2, x_3, -1, \dots)$ which sends a message

For the rank 3 processors we have:

the node $(x_1, x_2, x_3, -1, \dots)$ sends a message $\Leftrightarrow f(x_1, x_2, x_3)$ is *true*.

The net result is:

the terminal prints 'the statement is true' $\Leftrightarrow \exists x_1 \in \mathbb{N}$ not $\exists x_2 \in \mathbb{N}$ not $\exists x_3 \in \mathbb{N}\ f(x_1, x_2, x_3)$

This is the same as our original statement by the usual negation of quantifiers. The proof for the general case follows this pattern by induction on the number of quantifiers and quantified variables. □

5.3 Diophantine equations, Diophantine sets and semi-decidable sets

Here is a simple application. Consider a polynomial D in several variables with integer coefficients. A *Diophantine equation* is of the form $D(a_1, \dots, a_n, x_1, \dots, x_m) = 0$ where the a_i and x_j are natural numbers. We have separated the variables for the polynomial into two collections, the parameters a_i and the unknowns x_j.

A subset $A \subset \mathbb{N}^n$ is *semi-decidable* if it is exactly the set of inputs on which a given Turing machine eventually halts.

A *Diophantine set* $A \subset \mathbb{N}^n$ is a set specified in the form

$$\underline{a} \in A \iff \exists x_1, \dots, x_m \in \mathbb{N} : D(\underline{a}, \underline{x}) = 0 \ .$$

From the Matiyasevich-Robinson-Davis theorem (22) we can show that the semi-decidable subsets in $\mathbb{N}^n$ are precisely the Diophantine sets. Since each Diophantine set is Σ_1 in the arithmetic hierachy we have the following:

Corollary 5.3. *The infinite parallel Newtonian machine can decide all Diophantine sets.*

As the prime numbers are a Diophantine set, we can construct an oracle of primality within the machine.

If we consider the program for deciding a given Diophantine set, we see that it is simplest to consider the specification above as an m quantifier problem. The alternative would be to code $\mathbb{N}^m$ into $\mathbb{N}$, and then use a single quantifier, but we would then have to deal with decoding the number. We load $\underline{a}$ into each component machine, and then use the rank m machines, with the machine at position $(\underline{x}, -1, \dots)$ computing $D(\underline{a}, \underline{x})$. As all the computations of $D(\underline{a}, \underline{x})$ involve the same number of operations, these machines will all finish in a given time, and then the existence of a zero value (if there is one) is reported back to the terminal via the machines of lesser rank.

6 Critique of Machine

There is no shortage of critical remarks to be made about the infinitely parallel Newtonian machine. Some points ask for technical improvements of the example. However, the main purpose of the discussion is explore computation in physical terms. We *know* that any Newtonian machine is open to criticism since we *know* that Newtonian mechanics has assumptions that are not generally true of the world. This is the reason we emphasise speculation and analysis of impossible machines in the programme of Section 2. The criticism is an exploration of the subtheory not a judgement on its irrelevance or on the inadequacies of design.

6.1 Background

As pointed out in our previous papers (1; 2), the real problem in computability of mechanical systems is this:*In some nice subtheory, a system or machine may be shown to exist that can compute by experimental computation any given function from* $\mathbb{N}$ *to* $\mathbb{N}$*, or decide the arithmetical sets, but what has the theory to say on how was it made in the first place?*

In our previous examples, knowledge of the non-computable sets would be required in any manufacturing process. In addition, the machines in (1; 2) were created to solve a specific problem, they are obtained from the *deus ex machina* shop with a label 'machine for computing the set ..., and only that set'. Whereas by standard coding tricks it is possible to calculate various universal sets for recursive functions from a single such machine, nonetheless the user cannot convert the machine to another purpose, i.e., it is not programmable.

Another trick in Newtonian mechanics to make machine models for beating Turing machines is to have arbitarily fast clock speeds or to violate the atomic hypothesis by having arbitrarily small parts.

The machine given here is an attempt to answer these criticisms. It is

(i) fully programmable;

(ii) its component parts are essentially identical and the same size, and the program is the same for all processors; and

(iii) there is a universal clock speed for all processors.

As far as construction is concerned, the construction process is carried out by an infinite number of gangs working at each site $2^{n_1+1}3^{n_2+1}5^{n_3+1}\ldots$ along the x-axis. They work to identical blueprints, except that there are formulae for the number and contents of the address registers, the aim point of the gun and the amount of gunpowder in each charge for the gun. The gangs then complete their finite stretch of the rigid control rod which allows similtaneous communication to all of the machines, to load the program and carry the synchronisation pulses.

Clearly, suppose the mechanical subtheory specifies the making of such a machine by means *basic construction steps* then if it permits infinitely many gangs the machine is constructable in finite time. However, if the subtheory allows only finitely many gangs then the machine cannot be completed in finitely many steps. Our example, therefore also has to rely on a *deus ex machina* principle.

6.2 Criticisms

Now we summarise the concrete criticisms of the Newtonian hyper-computer:

1. Size and complexity It is of infinite spatial extent and has infinitely many component processors.

2. Energy and mass It requires arbitarily large velocities and energies for the communications. It has an infinitely long rigid rod to program.

3. Timing The firing sequence for the guns requires some very delicate timing. The reader may have noted that to avoid confusion the communication is fitted into time slots which are of progressively smaller duration for larger addressed component machines (recall our comments on projectiles and targets). Whereas we might just plead guilty again, we might have two other defences.

(i) The delicate timing is unnecessary for the Diophantine problem, we only need to know if a gun has been fired at a component machine or not. If we raise a hat, and then look for bullet holes, it doesn't matter how many bullets hit, or where they came from.

(ii) For more general programming applications, we wanted to be able to transfer the contents of a register from one machine to another. However there was no way to know whether a large (or infinite number) of component machines might be trying to input data to the same component machine in a given time step, thus the rule that only the least numbered address would succeed. The easiest method chosen to do this was to use discrete time slots. However we might alternatively have used the infinite spatial extent in the y direction of a given component machine to present one target for each possible input machine, and then use a system of rigid rods to do the sorting to ensure that only the least address gets through. The reader is welcome to look into this possibility and see if it really works!

4. Uniform cost of operations The component machines are required to be able to do the addition, subtraction and multiplication of integers of arbitrary length in one time step.

(i) We claim that a Newtonian machine of infinite spatial extent that uses rigid rods can be constructed to perform the given operations in one time step. The registers could consist of an infinite row of holes labelled by $\mathbb{N}$. A stored number n would consist of a ball in the hole indexed by n. The operations would be implemented by rigid rods linking specific registers to an accumulator. Similarly, copying could be implemented by rigid rods and a store of balls. The reader is welcome to look into this and seek other solutions!

(ii) If we were to use component machines which could only do fixed length arithmetic in a single step it would no longer be obvious that we could program the infinite machine to find Diophantine sets. However, it would be very suprising if the resulting modified infinite machine could not still compute things impossible for a single Turing machine. The reader may consider this as another challenge!

7 Concluding remarks

Experimental computation is not well understood even in the case of kinematics, possibly the simplest physical theory. We have proposed a four stage programme for investigating theoretically experimental computation and hypercomputation. One key idea is embedding hypercomputation into simple subtheories of physical theories. We have embedded an infinitely parallel computer into Newtonian kinematics. Simple examples and subtheories are important because we do not understand the interface between computation and physics and we need them as samples to put under a mathematical microscope.

At present hypercomputation is beyond our technological ability, but if this was ever to change, it would likely be done by successive refinement of physical subtheories towards a more practical model. For this to happen, we must be able to propose limited models of physics which allow hypercomputation, and then analyse their strengths and weaknesses. To believe that a practical idea of hypercomputation will suddenly spring fully formed into the world, as Athena did from the head of Zeus (Apollodorus, Bibliotheca I.III.5-6), is rather optimistic.

7.1 Infinite parallelism

Let us define a *synchronous concurrent algorithm (SCA)* to be a network of processing units distributed in space, computing and communicating in parallel, and synchronised by a family of clocks. An SCA captures the structure of many types of computational system, including neural networks, cellular automata, coupled map lattices, systolic algorithms etc. Commonly, the synchronisation is with respect to a single global clock. An *infinite* synchronous concurrent algorithm is one with infinitely many processing units. Such infinite SCAs have been considered (21; 37). Some theorems on the computability of finite SCAs over real numbers were presented in Holden *et al* (15). Our infinitely parallel machine is an example of an SCA with simple uniform conditions on processing units (e.g., natural number data and a common program) and complicated synchronous communication. One can separate the work in Sections 4-6 into two tasks. First, there is devising and specifying infinite architectures:
Problem 7.1. *Develop a portfolio of infinite synchronous concurrent algorithms that are hyper-computers capable of deciding precisely the sets of the arithmetical hierarchy.*

Second, there is implementing the architectures in a physical theory:
Problem 7.2. *Embed a general class of infinite synchronous concurrent algorithms into Newtonian subtheories.*

The notion of synchronous concurrent algorithm reminds us of computation by *analogue networks* in which the component machines compute with real numbers. The early analogue computers were often designed to solve mechanical problems by mechanical means. In Tucker and Zucker (42) we prove theorems that show that certain classes of analogue networks computing with continuous time streams of data from metric algebras are *not* hyper-computers. Analogue computation is a rich idea: as Pour El first showed in (29), many models are *sub*-recursive and fall within the computable see (14); for analogue hyper-computation, see Bournez (3) and Mycka (24).

7.2 Newtonian computers and hyper-computers

In the light of our own programme and other work, two open technical problems about Newtonian systems are these:
Problem 7.3. *For all valid Newtonian kinematic systems that possess both lower and upper bounds on space, time, mass, velocity and energy, are the sets and functions computable by experiment also computable by algorithms?*

We conjecture that the answer is “Yes”. To prove this, one needs to study general classes of experimental procedures and equipment.

Our examples here and elsewhere show that the notion of constructible equipment in Newtonian mechanics must be analysed in great detail and that it is complicated. Our reading of the literature has proved to be a hunt for *deus ex machina* principles that allow hyper-computation which, in our own work, have been chased into the unanalysed concept of experimental hardware.

Problem 7.4. *Extend Newtonian mechanics by a theory of experimental computation including experimental procedures for operating equipment and the construction procedures for making equipment, and show that the sets and functions computable by experiment are precisely the same as, or equivalent to, those computable by algorithms.*

One goal of the direction of the research programme, from physical theory to computablity, is, roughly speaking, explore the problem:

Problem 7.5. *To derive forms of the Church-Turing Thesis as laws in physical theories.*

We thank Istvan Nemeti, Warren Smith, E B Davies and a referee for comments useful for this paper. We thank the editors for their encouragement and assitance.

References

[1] E J Beggs and J V Tucker, Computations via experiments with kinematic systems, Research Report 4.04, Department of Mathematics, University of Wales Swansea, March 2004 or Technical Report 5-2004, Department of Computer Science, University of Wales Swansea, March 2004.

[2] E J Beggs and J V Tucker, Newtonian systems, bounded in space, time, mass and energy can compute all functions, Research Report 4/05, Department of Mathematics, University of Wales Swansea, March 2004 or Technical Report ?-2004, Department of Computer Science, University of Wales Swansea, March 2004.

[3] O Bournez, How much can analog and hybrid systems be proved (super-)Turing?, in *Applied Mathematics and Computation*, this issue.

[4] S B Cooper and P Odifreddi, Incomputability in nature, in S. B. Cooper and S. S. Goncharov, (eds.), *Computability and Models: Perspectives East and West*, Kluwer Academic/Plenum Publishers, Dordrecht, 2003, pp. 137-160.

[5] P Cotogno, Hypercomputation and the physical Church-Turing Thesis, *British Journal for the Philosophy of Science* 54 (2003), 181-223.

[6] N C A da Costa and F A Doria, Undecidability and incompleteness in classical mechanics, *International Journal of Theoretical Physics* 30 (1991), 1041-1073.

[7] N C A da Costa and F A Doria, Classical physics and Penrose's thesis, *Foundations of Physics Letters* 4 (1991), 363-373.

[8] E B Davies, Building infinite machines, *British Journal for the Philosophy of Science* 52 (2001), 671-682.

[9] M Davis, The myth of hypercomputation, in C Teuscher (ed), *Alan Turing: Life and Legacy of a Great Thinker*, Springer, 2004, 195-211.

[10] M Davis, Why there is no such discipline as hypercomputation, in *Applied Mathematics and Computation*, this issue.

[11] J Durand-Lose, Abstract geometric computation for Black hole computation, in M Margenstern (ed.) *Universal Turing Machines and Computations*, Springer Lecture Notes in Computer Science 3354, 175-186, 2005.

[12] G Etesi and I Nemeti, Turing computability and Malament-Hogarth spacetimes, Los Alamos arXiv.org:gr-qc/0104023, and *International Journal of Theoretical Physics* 41 (2) (2002), 342-370.

[13] R Geroch and J B Hartle, Computability and physical theories, *Foundations of Physics* 16 (1986), 533-550.

[14] D S Graça and J F Costa, Analog computers and recursive functions over the reals, *Journal of Complexity* 19 (2003) 644-664.

[15] A V Holden, J V Tucker, H Zhang and M Poole, Coupled map lattices as computational systems, *American Institute of Physics - Chaos* 2 (1992), 367-376.

[16] M Hogarth, *Predictability, computability and spacetime*, PhD Thesis, Cambridge University, 2002.

[17] M Hogarth, Deciding arithmetic using SAD computers, *British Journal for the Philosophy of Science* 55 (2004), 681-691.

[18] T Kieu, Hypercomputability of quantum adiabatic processes: prejudices versus facts, in *Applied Mathematics and Computation*, this issue.

[19] G Kreisel, A notion of mechanistic theory, *Synthese* 29 (1974), 9-24.

[20] L Lipshitz and L A Rubel, A differentially algebraic replacement theorem, *Proceedings of the American Mathematical Society* 99(2) (1987), 367 – 372.

[21] B McConnell and J V Tucker, Infinite synchronous concurrent algorithms: the specification and verification of a hardware stack, in F L Bauer, W Brauer, H Schwichtenberg (eds), *Proceedings of NATO Summer School 1991 at Marktoberdorf, in Logic and algebra of specification*, Springer, 1993, 321-375.

[22] Y Matiyasevich, *Hilbert's Tenth Problem*, MIT Press, 1993.

[23] C Moore, Unpredictability and undecidability in dynamical systems, *Physical Review Letters* 64 (1990), 2354 – 2357.

[24] J Mycka, Analog computation beyond the Turing limit, in *Applied Mathematics and Computation*, this issue.

[25] I Nemeti and Gy David, Relativistic computers and the Turing barrier, *Applied Mathematics and Computation*, this issue.

[26] M A Nielsen and I L Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 2000.

[27] T Ord, Hypercomputation: computing more than the Turing machine, in *Applied Mathematics and Computation*, this issue.

[28] I Pitowsky, The physical Church-Turing Thesis and physical computational complexity, *Iyyun* 39 (1990), 81–99.

[29] M B Pour-El, Abstract computability and its relations to the general purpose analog computer, *Transactions of the American Mathematical Society* 199

(1974), 1 – 28.

[30] M B Pour-El and J. I. Richards, A computable ordinary differential equation which possesses no computable solution, *Annals of Mathematical Logic* 17 (1979), 61 – 90.

[31] M B Pour-El and J. I. Richards, The wave equation with computable initial data such that its unique solution is not computable, *Advances in Mathematics* 39 (1981), 215 – 239.

[32] M B Pour-El and J. I. Richards, *Computability in Analysis and Physics*, Perspectives in Mathematical Logic, Springer-Verlag, Berlin, 1989.

[33] H Rogers, *Theory of recursive functions and effective computability*, McGraw Hill, 1967.

[34] O Shagrir and I Pitowsky, Physical hypercomputation and the Church-Turing Thesis, *Minds and machines* 13 (2003) 87-101

[35] H T Siegelmann, *Neural networks and analog computation: Beyond the Turing limit*, Birkhäuser, Boston, 1999.

[36] W D Smith, Church's thesis meets the N-body problem, in *Applied Mathematics and Computation*, this issue.

[37] V Stoltenberg-Hansen and J V Tucker, Infinite systems of equations over inverse limits and infinite synchronous concurrent algorithms, in J W de Bakker, G Rozenberg, and W P de Roever (eds.) *Semantics - Foundations and applications*, Springer Lecture Notes in Computer Science 666, Springer Verlag, 1993, 531-562.

[38] V Stoltenberg-Hansen and J V Tucker, Concrete models of computation for topological algebras, *Theoretical Computer Science* 219 (1999), 347 – 378.

[39] V Stoltenberg-Hansen and J V Tucker, Computable and continuous partial homomorphisms on metric partial algebras, *Bulletin for Symbolic Logic* 9 (2003), 299 – 334.

[40] J V Tucker and J I Zucker, Computable functions and semicomputable sets on many sorted algebras, in S. Abramsky, D. Gabbay and T Maibaum (eds.), *Handbook of Logic for Computer Science* volume V, Oxford University Press, 2000, 317 – 523.

[41] J V Tucker and J I Zucker, Abstract versus concrete computation on metric partial algebras, *ACM Transactions on Computational Logic*, 5 (4) (2004) 611-668.

[42] J V Tucker and J I Zucker, A network model of analogue computation over metric algebras, in S B Cooper, B Loewe and L Torenvliet *Computability in Europe, 2005*, Springer Lecture Notes in Computer Science, 2005.

[43] K Weihrauch, *Computable Analysis, An introduction*, Springer-Verlag, Heidelberg, 2000.

[44] K Weihrauch and N Zhong, Is wave propogation computable or can wave computers beat the Turing machine?, *Proceedings of London Mathematical Society* 85 (2002), 312-332.

[45] P D Welch, On the possibility, or otherwise, of hypercomputation, *British Journal for the Philosophy of Science* 55 (2004), 739-746.

[46] S Wolfram, *A New Kind of Science*, Wolfram Media, 2002.

[47] A Yao, Classical physics and the Church Turing thesis, *Journal ACM* 50

(2003), 100-105.