Partially-Ordered Event-Triggered Systems (POETS)

Steve Furber[1], Andrew Brown[2]

1: School of Computer Science, University of Manchester, M13 9PL
2: Department of Electronics & Computer Science, University of Southampton, SO17 1BJ

Abstract: Event-triggered computing systems have long formed the basis of real-time embedded systems in industrial plant control, automotive and aerospace system, to name but a few. Each of these application domains comes with its own challenges, but - generalising wildly - there are numerically few inputs and the timing constraints are such that the system can be realised with typically a few cores. Within the last decade, technology has moved on to the point where "the core" - once a central and important component in any computing system - has become as commoditised as the transistor did forty years ago. They have become negligibly cheap, and this change of value has brought with it a change of design perspective: In the past, complex data structures had to be constructed to allow a machine to operate efficiently on large numbers of "problem components", and if the resource was insufficient, multiple cores would be bought into play, their interactions choreographed explicitly by expert software architects. It is now possible to create systems where the atomic elements of a datastructure are spread evenly and thinly over a huge number of small, simple cores, and the necessary computations executed by *cores local to the data*, rather than moving the data to the cores. In this paper we discuss realisations of this idea: the SpiNNaker engine, a custom system designed to simulate the behaviour of a billion mammalian neurons in real time - a feat made possible by a bespoke communications infrastructure, asynchronously and independently transporting tiny packets of information; we then go on to generalise the concept and describe the POETS computing system, which allows a far greater range of application domains to be addressed than does SpiNNaker.

The model

Introduction

Here we begin to develop a formal system model that can be used to describe the operation of biological neural systems (such as the brain) and computational models of such systems.

This work is motivated by a desire to find useful ways to think about information processing in the brain, and by a desire to produce a formal semantics that can underpin reliable operation of event-based machines.

We introduce three models at different levels of abstraction, progressing from the biology of neural systems down to the details of the SpiNNaker machine.

A hybrid-system model

The system is a set of dynamical processes $P = \{P_i\}$ that communicate purely though event communications using a set of event channels $E = \{E_j\}$.

Each dynamical process evolves in time under the influence of received events, so $P_i = p_i(t, E)$, where p_i is a function that may have internal state and t is time. Typically P_i will depend on a subset of the event channels, not all of them.

Each event channel carries events that are either generated by a process, or come from the environment, to all the other processes that depend on them. So for an internal event channel $E_i = e_i(P_j)$ for some j. Normally an "event" is a pure asynchronous event that carries no information other than that is has occurred, so it can be thought of as a time series of identical impulses:

$$E_i = \sum_k \delta(t - t_k),$$

where t_k are the times the events occur on this channel. Sometimes it might prove useful to be able to modulate the size of the impulse.

We can consider the event channel to be instantaneous, so that events arrive at all of their destinations at the same time that they are generated by their source process, though causality allows us to view this as "after" they are generated, albeit by a vanishingly small delay. Likewise, if an incoming event causes a process to generate an outgoing event this causality is captured by the output being "after" the input.

The outputs from the model are simply a subset of the total set of events, E.

Biological neurons

Biological neurons are complex living cells that have a cell body (the *soma*), a single output (the *axon*) that carries action potentials, and a complex multi-branched input structure (*dendrites*) that collect inputs. The axon from one neuron couples to the dendrite of another through a *synapse*, which is a complex adaptive component in its own right.

Action potentials are sustained and propagated by electro-chemical processes in the axon that allow them to be viewed as pure asynchronous events.

Long axons incur significant delays, but these can be rolled into the transmitting and/or receiving process. Where there are different delays from a single source to different targets, for example a short delay to proximal targets and a long delay to distal targets, the hybrid-system model allows this to be captured either by different delays in the receiving process or by the source transmitting separate events with different source delays, or some combination of these.

We therefore claim that the hybrid-system model captures the essential features of biological neurons that exchange information principally through action potentials.

Action potentials are not the whole story, however. Some neurons produce chemical messages, for example dopamine, that modulate the activity of other neurons within a physical region. Some neurons make analogue dendritic connections with their neighbours. These phenomena are outside the hybrid-system model, but we hope that their principal effects can be captured through back-channel processes of some sort.

In addition, biological systems do not have static connectivity – they develop and grow, gaining and losing neurons and connections to their "event channels". These happen slowly relative to the real-time information flow, and such dynamic topology changes many be modelled through back-channel processes.

Biological systems are also very noisy, but we can accommodate this by using noisy processes.

An abstract computational model

We cannot compute a continuous process exactly as in the hybrid-system model, so for efficiency it is important to approximate the process in some way. Most neuron models are some form of system of differential equations, so it is common practice to compute these using a form of integration over discrete time-steps.

For *real-time* modeling, the integration can be implemented by introducing an additional "time-step" event, E_t. Now time is just another, regular (e.g. 1ms) event, from an external source, and t can be removed from the model.

We can, at least in principle if our computer is sufficiently fast, ignore the time taken for a process to handle an event. Each event is handled as it arrives, and each process is simply a set of rules defining how that process's state is changed by every possible input event. Thus:

$$P_i = [E_j \Rightarrow S_i \leftarrow p_i(S_i, j)] \forall j : E_j \in I_i$$

where S_i is the state of P_i and I_i is the set of events, now including E_t, that are inputs to P_i. This is the **event-triggered** aspect of POETS.

It is clear that a process is active only in response to an input event, and therefore any output events it generates must also occur at the same time as (though causally after) an input event. Note that this does not preclude an internal time delay between a neural input and the output it causes: the input can change the state of the process, which then progresses through several time-step events before producing an output. But the output will eventually be produced in response to, and at the same time as, a time-step event. As the only representation of time in the system is the time-step event, time is discretized.

Since the time-step event, E_t, connects to many (if not all) processes, there may be many events generated just after it. These events, from different processes, have no implicit order. This gives rise to the **partially-ordered** aspect of POETS. Each process to which some of these concurrent events are inputs will impose an arbitrary order on their reception (at notionally the same time), and as a consequence the system behaviour is non-deterministic at this point.

A SpiNNaker computational model

SpiNNaker is a massively-parallel system with an interconnect fabric designed specifically to convey events generated by a program running on one processor to all of the processors to which that event is an input. The SpiNNaker fabric must initially be configured to put the necessary connections in place, but once so configured the

hardware looks after the event connections. Processors then receive events intended for them and issue events with no knowledge of where they are destined to go.

Unfortunately the processors on SpiNNaker aren't infinitely fast, so a process takes a finite time to complete its response to an input event. While it is running another event may arrive, demanding pre-emption. The time-step event may not be synchronized across the machine (although near synchronization is possible using a technique such as fire-fly synchronization).

A further complication is that SpiNNaker processors keep some of their state in off-chip SDRAM, access to which incurs high latency costs. In general we aim to hide this latency by exploiting a DMA subsystems attached to each processor to handle SDRAM transfers while the processor gets on with other stuff.

These (and other) niceties apart, SpiNNaker aims to implement the abstract computational model as faithfully as it can, subject to all of the constraints of the physical system, delivering a reasonably efficient solution, and minimizing energy consumption.

SpiNNaker models may attempt to implement the abstract computational model faithfully, in which case they will aim to synchronize the (notional) 1ms time-step across the machine and complete all the work in every 1ms to stay in lock-step across the machine. In this case the ***peak*** process load must complete within the 1ms for correct operation. Alternatively, they may adopt an asynchronous model where there is no attempt to align a 1ms period in one process with that in another, in which case the ***average*** process load must complete within 1ms for correct operation.

Spiking neurons on SpiNNaker

Each processor on a SpiNNaker machine handles one process, where each process models a number of neurons. As incoming events from other processes are very similar they are handled by one event handler. The simplest model of a SpiNNaker process then handles two event types:

1. Incoming neuron event: locate and process synaptic data, updating local neural state accordingly.
2. Time-step event: perform integration step for all local neurons, possibly generating outgoing events.

As an implementation detail the neuron event handler will usually invoke a DMA transfer to bring the synaptic connectivity data in from SDRAM, but as this is internal to the process we hope to hide the DMA as much as possible from the application code.

This model does not handle the important aspect of synaptic plasticity, but already creates some interesting data consistency issues if a type 2 event occurs while the (fairly long) event 1 process is running and pre-empts it. These data consistency issues are avoided if no input is allowed to affect state that is used in the current time step, which amounts to imposing a minimum axonal delay of 1ms.

In general a SpiNNaker implementation uses a very simple real-time kernel of some sort, with drivers for the event communication system, DMA, etc. It includes

queue management, priority scheduling, buffer overflow procedures, and so on. This notwithstanding, it maintains a strongly event-driven nature, spending any idle time in a low-power wait-for-interrupt state.

What can a formal model offer?

A validated formal model can answer various important questions about the SpiNNaker system. At the low level:

- Is the run-time software a robust implementation of the computational model?

And at the high level:

- Is system activity at a stable level, or will it grow uncontrollably (as in epilepsy) or die away?
- How does the processing of neural information through successive layers "add value"? For example, in vision we start from pixels, which are processed (in the retina) into centre-surround signals. The primary part of the visual cortex is known as V1, (**V**isual area **one**), which processes these signals into edge/–corner/orientation, and up through further layers into "car" or "tiger". (Both hemispheres of the brain contain visual cortex, one for each visual field.) How can we quantify the benefits of each layer, preferably in information-theoretic terms?

Generalising....

Changing the narrative perspective significantly and moving it back out, Moore's Law has given us a doubling of logic density every eighteen months or so for over four decades. It has enabled microelectronics to move from a narrow professional niche into the hands and pockets of every consumer in the world. However, as process geometries continue to shrink towards the scale of the atom, we face the emergence of fundamental limits which the scaling of current methodology can no longer easily overcome; increasingly, far ranging architectural - both hardware and software - changes are required to utilise the potential of the technology. Four major challenges can be identified:

- **Power dissipation:** it is already not possible to power all parts of a chip at the same time (the *dark-silicon* problem). It has been demonstrated that multiple small CPUs are correspondingly more power efficient than fewer large ones, so the deployment of large cohorts of small CPUs is an obvious way forward.
- **Reliability:** As process geometries continue to shrink, issues of reliability and robustness inevitably emerge. In a system of millions of cores (not unreasonable today), it is unrealistic to expect 100% functionality 'out of the box'; equally, cores will inevitably fail over the lifetime of the system.
- **Communication vs computation:** A traditional argument against moving to large numbers of cores is the relative cost of computation and communication. A core can typically perform several thousand operations in the time taken to

get a single word out of memory and made available to a core. Ever deeper caches and convoluted pipelines can help alleviate the problem, but with conventional architectures, bottlenecks are still almost unavoidable.

- **Programming:** In the past, processor time (core hours) was a valuable resource, and much work went into understanding how to optimise the scheduling of a workload on parallel machines. The automatic (high-level) parallelisation of general-purpose codes remains a 'holy grail' of computer science, but fine-grain parallelisation is frequently signposted by the underlying mathematics. The problem has been in the past that solutions emerging naturally from a numerical solution technique do not map well (cheaply) onto existing architectures, partly because cores were *relatively* scarce, compared to the granularity of a discrete solution. Today, processing is effectively a free resource: cores do not have to be 'kept busy'.

These considerations form another set of constraints on a design space that is already extremely complex. However, they *also* open the way to new approaches: design space may become more convoluted, but it also gets bigger.

Whilst there is no way **through** Amdahl's Law *(The proportion of code that cannot be parallelised will ultimately limit the advantages accrued from more processors)*, the Gustafson-Barsis Law does permit a way **around** it: *(If you can have an **arbitrary** number of processors, the **total** amount of work performed by the system may be increased arbitrarily **at no extra cost**)*.

POETS technology exploits this and explicitly addresses all these points simultaneously.

What is POETS?

POETS - ***P***artial ***O***rdered ***E***vent ***T***riggered ***S***ystems - technology is based on the idea of an extremely large number of small cores, embedded in a fast, hardware, parallel communications infrastructure - the **core mesh**. Inter-core communication is effected by small, fixed size, hardware data **packets** (a few bytes) - aka **messages**.

This proposal describes research to investigate and prototype a software methodology and associated hardware platform to realise the potential of this architecture.

The physical implementation of such a system imposes a fixed and finite topology on the core graph, but a thin (hardware) layer on top of the cores allows the user to virtualise an *arbitrary* connectivity graph on top of the physical one. Once this is done, the mapping of problem domain to processor mesh follows naturally.

For example, a surprising number of industrial problems map naturally and ultimately to solution of the matrix equation [A]x = [B], and the efficient solution of this *prima facie* simple problem for large (say, rank 1000) and ill-defined systems is still the subject of current research. Using POETS technology, ***each matrix element can be mapped onto its own core***: textbook solution techniques become possible because element-element communication is truly (hardware) parallel across the entire matrix. Traditionally, calculations of this type require polynomial time; POETS can perform the calculation in linear time - a massive difference with large industrial problem sets.

Why now?

Because we can - ten years ago it was not possible.

In 1965 Gordon Moore published his famous prediction: that the number of transistors on a chip would double every 18 months or so. This is not a law, just a market prediction, yet it has become a self-fulfilling prophecy that has guided industry for decades. However, it is an exponential prediction, and no exponential is sustainable indefinitely in nature.

Moore's Law is coming to an end, gradually, not because of any one particular show-stopping physical limit, but because of a host of effects, each one in isolation probably capable of resolution, but taken together present an insuperable barrier: *it simply isn't worth it any more*.

But: if we focus on the last few years of this line, and recalibrate the axes in terms of cores/chip instead of transistors/chip, we see the beginnings of a new law: the number of cores/chip is increasing by some multiple/year. Yes, it is an exponent, and so it won't last, but while it does, we should exploit it.

POETS fits into the landscape described above in innovative ways:

- **Power dissipation:** POETS is an *event-driven* system. Cores carry out small calculations in response to the arrival of a message, based on a state subset held in local memories. These calculations may/may not result in the emission of further messages, which are immediately swept up by the communication infrastructure and delivered asynchronously, via hardware, to their target core. The target core is woken (by the hardware delivering the message), acts upon it - as above - and *returns to quiescence*, awaiting another stimulus. POETS is intrinsically energy frugal - you only power calculations when you perform calculations. The design intention is that for a significant portion of time, each core is asleep. This is a programming model of immense power and enormous potential, and is completely orthogonal to conventional architectures.
- **Reliability:** POETS architecture is intrinsically resilient in the face of hardware failure for two reasons: (1) one way of thinking about a POETS core is to view it as an asynchronous finite state machine. Like its conventional counterpart, there is no reason why its state transition graph cannot be disjoint - POETS cores can multi-task at an event level, and so can run inconspicuous system integrity checks in parallel with anything else, allowing possible recovery and/or graceful performance degradation in the face of core or communication fabric failure. (2) is rather more subtle, and not applicable to all problem domains. The dominant use intention of the system is that a fine-grained mathematical model is mapped to the core mesh for subsequent processing - usually but not always some kind of simulation. Failure of a core (or part of the communication fabric) therefore has the effect of compromising the simulation model (specifically the state subset held in the failed area), rather than the algorithm, which is distributed over the entire system. For a certain subset of problems (notably relaxation-based simulations), this perturbation is minimal, localised and does not propagate.

- **Communication vs computation:** POETS sidesteps this tension by 'embedding' the cores in a hardware communications fabric (which is truly parallel) and in which the messages are small and of fixed size (a few bytes). (This is one of the core outcomes of the SpiNNaker project.) With POETS machines, the burden of high-level message choreography is completely removed (there are none): systems trade cores against complexity in both compute and communications.
- **Programming:** This is the area where the largest research challenges lie. Our work with SpiNNaker has demonstrated the validity of the POETS concepts, but the use cases to date have all been hand-crafted. The challenge here is to find a way in which domain-specific specialists - who neither know nor care about the underpinning technology - can use the system to attack large, industrially important problems, focussing on the problem, without the distraction of the solution technique and details.

Taken together, these attributes represent a significant sea-change in the way in which large, industrially relevant problem sets may be attacked. POETS is not a general-purpose architecture, but nor is it a corner-case; it is elegantly suited to a wide variety of industrial problems:

- Finite difference and finite element problems
- Computational chemistry
- Particle & field
- Image processing
- Neural synthesis and simulation (Human Brain Project)
- Drug screening
- Discrete system simulation

In fact, anything where the underlying mathematics naturally formulates as a large graph with large numbers of small, parallel interactions, and no overarching synchronisation requirement.

For some - not all - industrial problems, POETS architectures are capable of delivering orders of magnitude speed increases.

What needs to be done?

Our work with SpiNNaker delivered the first large-scale existence proof of the power of this concept. If we are to exploit this hitherto underexplored and unconventional computing technology, there is still much research to be done. For nearly every step of the development trajectory to date, almost every tool and technique that a conventional software developer takes for granted has had to be re-engineered from scratch. Conventional support tools do not work with this system. We need standardised input formalisms, we need command, control, internal visibility and debug tools, we need to know where the limits are of an instance of the architecture, and how difficult and expensive it is to move these limits.

Why should we bother?

Despite decades of attack, the general purpose parallelisation problem remains one of the most elusive 'holy grails' of computer science. Inspired - possibly - by what we achieved by simply ignoring difficult general problems in our past neural simulation work, and focussing on the functionality we actually *needed*, our thesis here is that POETS is incredibly well-suited to an unexpectedly wide range of important engineering and physics problems, most of which are traditionally the domain of large, extremely resource hungry supercomputers. Event-driven programming, using thousands to millions of small, cheap, energy frugal cores is by far the best platform for some massive engineering problems that traditionally consume millions - tending to billions - of core-hours *and* watts, both of which translate directly into money. **Proponents of exascale computing need event-driven machines if budgets are to remain sub-exascale.**

The problem is not *creating* large cohorts of processors, but how they might be productively *used* to perform or enable the sort of analyses that users of big compute demand. The project focuses on the potential of the hardware architectural point on the scale represented by the earlier SpiNNaker work. It will look at the areas of work where the hardware architecture would be well suited, how those needs can be supported on this architecture by methods and tools, and how the architecture may be further optimised, with the objective of providing the basis of knowledge to support valuable commercial exploitation opportunities anticipated to emerge for commodity HPC.

POETS is a different type of computing architecture; no mature tools or techniques currently exist to exploit it fully, and physical implementations are not yet commonplace. However, this will change: the architecture is unusual but has the attraction of being the choice of evolution - it is the architecture of all neural cortexes, including our own brains. It is highly functional, extremely power efficient and very fault tolerant. Whilst it demonstrably can be programmed, research is needed to make it a commodity capability on a par with the architectures found in almost every electronic system today.

Aims of the research

The phrase "Technology Readiness Level (TRL)" has several definitions, not all of which are mutually consistent, but in essence POETS is currently squarely at TRL 1: the basic principles have been observed and reported. The goal at the project end is to take the concept at least to TRL 4: (Validation of the concept in laboratory conditions), preferably, with the assistance of the project partners, to TRL 5: (Validation of the concept in a relevant domain-specific environment.)

The long-term strategy is to be in a position, at the end of the project, to be able to approach tool vendors and specialist product providers with a solution technique - ***cast in domain-specific terms that they are familiar with, demonstrating solutions to real problems that they care about*** - and make a case for the commercial uptake of the developed technology.

SpiNNaker and POETS:

SpiNNaker is a distributed multi-core system, consisting of a network of 65 000 **nodes**, (each containing 18 200MHz ARM9 cores), embedded in a bespoke (hardware) message-passing infrastructure. The nodes are triangularly connected in a two-dimensional (2D) planar mesh, the edges of which are identified with each other and the whole plane wrapped onto the surface of a torus. Cores communicate via hardware packets of 72 bits. Each node also contains a router unit to control all packet movements, both inter- and intra-node. The design of SpiNNaker explicitly disregards three of the central planks of computer architecture dogma:

1. There is no central synchronising system clock, and all the inter-node (and much intra-node) communication is asynchronous.

2. There is no attempt made to enforce overall memory coherency. Each core has its own private memory, which is not visible to any other core.

3. The message passing infrastructure is non-deterministic (and may, under certain circumstances, be non-transitive).

SpiNNaker is designed for use as a neural simulator. At the level of abstraction utilised by SpiNNaker, a neuron consists of a multi-input, multi-output unidirectional discrete component, communicating with its peers via **action potentials**, modelled as discrete events. A neural system or circuit is represented as a graph of neurons, mapped onto the physical core mesh. A thin hardware layer (the routing system) enables transparent neuron-neuron communication over the underlying hardware - the biological model does not 'see' the underlying electronics.

SpiNNaker is intended to simulate neural aggregates in real time: the biological information contained within a packet resides in the wallclock time of arrival; the 72 bits interact with the underlying routing system to ensure the right packet gets to the right neuron model.

POETS: SpiNNaker is extremely good at the task for which it was designed: neural simulation. The idea of virtualising an abstract arbitrary graph and mapping it to hardware via a thin, hardware, parallel routing layer is immensely powerful, and opens the door to a large array of application domains. However, SpiNNaker is an ASIC, and contains aspects - that we cannot change - that make it unsuitable for the generalisations we wish to explore in POETS.

- In biology, the existence of a spike (packet) contains the only biological data. It is here at this time, or it isn't; this is how mammalian neural systems work. For more diverse applications, we need to be able to put more data into a packet. To keep the speed advantage, the packet size must be small and fixed size, but the bit length of SpiNNaker is crippling for other domains. A few dozen bytes would be fine; 72 bits is not enough.
- Data exfiltration: SpiNNaker relies on gross neural activities for I/O - this is how biological systems work. We need to be able to extract global state data reliably.
- Global synchronisation: Biological systems do not support this behaviour -

neither does SpiNNaker. There exists a wide set of circumstances - in diverse application areas - where this functionality is essential.

Event triggered computing - key points:

Architecture

- *Extremely* large numbers (1000000+) of *extremely* simple cores
- Short (a few bytes), uniform messages
- Hardware massively parallel communications network (on and off-chip)

Disadvantages

- Not a general purpose architecture
- Cannot port existing codebases
- No existing support toolsets

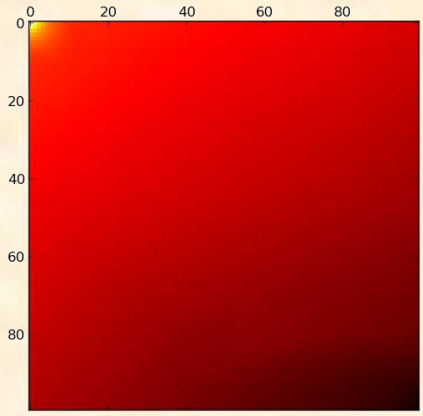
Advantages

- *Massive* speedups for certain classes of problem: $\mathbf{O(n^m) \rightarrow O(k)}$
- Highly fault tolerant
- Low power: **25000 cores < 13A**

Use cases

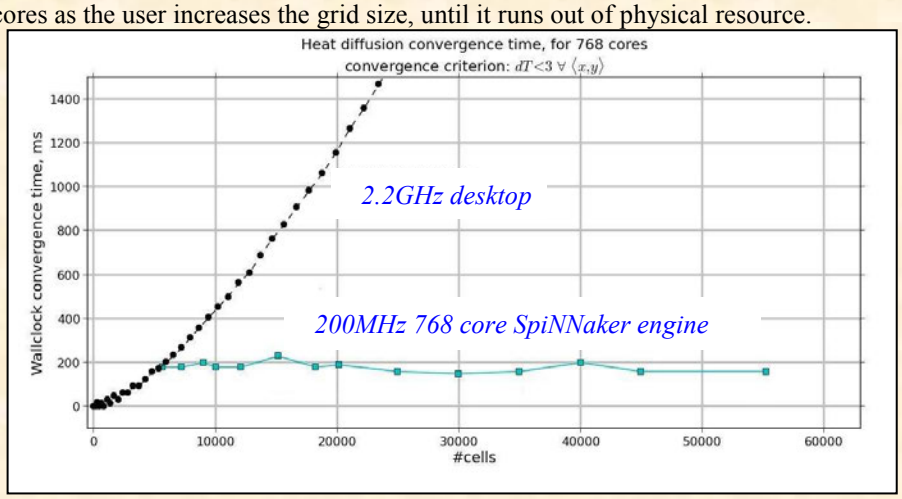
A large proportion of real engineering problems can be broken down to a discrete graph, albeit one with sometimes millions of nodes. If we have millions of cores and a fast communications infrastructure, we can trade cores off against computational complexity, and exploit the near-perfect parallelism of the hardware interconnect. The Use case portfolio suggests some of the industrial application areas for POETS technology.

Use case: Finite difference calculations

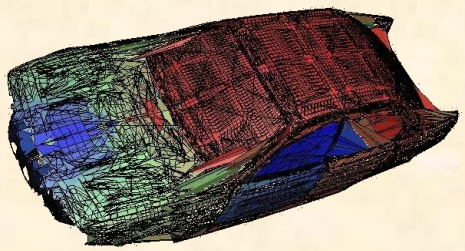


Consider the canonical finite difference heat equation on a 2D square grid: ***each grid point is represented by an individual core***, which holds the grid point state (temperature) plus ghosts of the immediate neighbouring states, and communicates only with its direct neighbours by messages. On receipt of a message - any message - a grid point recomputes its state; if this has changed, it broadcasts the new state value to its neighbours. All the cores do this simultaneously (asynchronously), triggered by the arrival of messages. Pinning the opposite corner temperatures and letting heat flow freely produces the obvious result.

The interesting point is the wallclock solution time as a function of grid size: the algorithm, as cast, will continue to operate in constant time, using more and more cores as the user increases the grid size, until it runs out of physical resource.

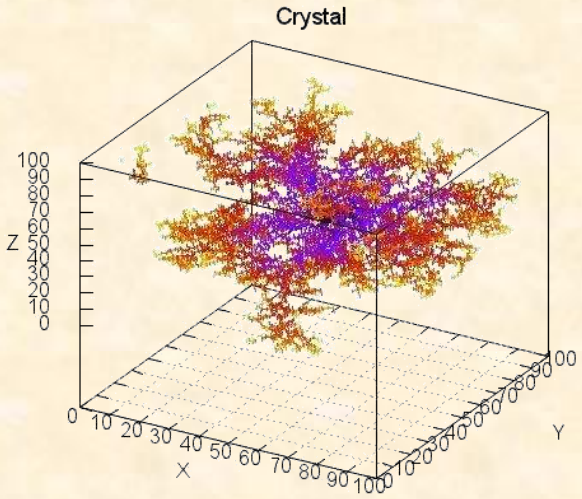


There is, obviously, no reason why we need restrict our analyses to a uniform grid:

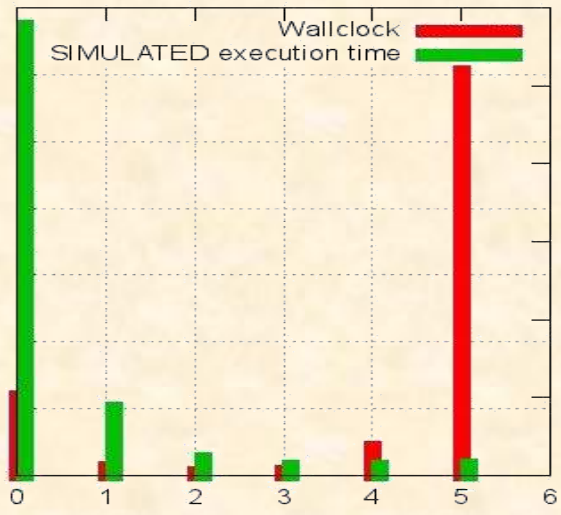


Use case: Neuron synthesis

Large scale neural simulations - which underpin almost all of computational neuroscience - require *realistic* models to simulate, and the generation of these models is not trivial or computationally cheap. In biology, $1mm^3$ of neural matter contains around 10^5 cell bodies, 4km of axons, 5.10^6 dendrites and 7.10^9 synapses. *Each* neuron is represented as a space-filling tree, which does not intersect with itself or any other neuron. Vasculature - essential for accurate modelling - approximately doubles the complexity of the space.



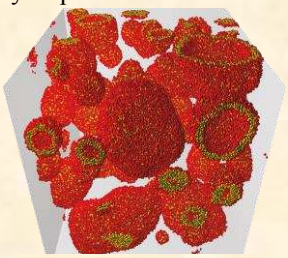
A popular way of approaching this problem is to tile space (the universe) with three-dimensional cubes, populated with 'virtual neurites'. These move about randomly, condensing (and sticking) onto a seed neuron whenever they touch one.



Using POETS methodology, we can allocate ***each spatial 3-cube to an individual core***, and handle the passage of neurites and the growth of neurons across cube boundaries by passing messages. Run 0 below shows the universe modelled by one POETS core; run 5 by 32 x 32 x 32 (= 32768) cores. (Intermediate data points are for 2^3, 4^3, 8^3 and 16^3 cores.) The figure itself is a simulation, but nevertheless the speedup trend is clear and impressive.

Use case: Spatio-temporal simulation of stochastic biochemical processes

Biochemical processes are increasingly being modelled in-silico, where a low-level description of chemical interactions is used to drive a simulation of higher-level biological activities; these models have been made possible by improved abilities to automatically extract individual molecular pathways. Modelling the interactions within an entire cell is computationally infeasible, due to the large number of molecules, and the huge number of interactions needed for the cell to make enough progress to be interpreted at a high level.



*J. Shillcock, Langmuir 2012, **28**, 541-547*

Approximations are used to interpret and capture low-level processes as coarse behaviour, which have recently allowed the creation of whole-cell models for simple bacteria. However, there remains the question of whether some important behaviour is only captured by the low-level interactions, so there is still a need to perform high-fidelity simulations of chemical processes which track individual molecules.

Current cell simulation techniques provide an efficient method for simulating systems with tens of thousands of reactants, but current compute systems are too slow to come close to the speed needed to model all interactions within a cell. The spatial nature of the problem, and the heavy reliance on local lightweight communication, means that space can be discretised and ***each cube mapped onto a core***:

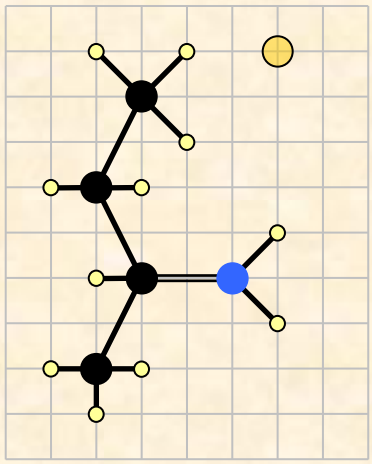
- *Loose temporal coupling:* the notion of time within a simulated system is intrinsically fuzzy, and only local causality matters.
- *Local fault tolerance:* as long as molecules can propagate between local volumes within a spatial region, the failure of one or two volumes within that region is largely irrelevant.
- *Scaling via spatial decomposition:* due to the huge number of molecules involved, the problem can be decomposed spatially until all available CPUs are occupied, achieving good utilisation of all available CPUs.

As well as being a good fit for the architecture, stochastic chemical simulation also presents some interesting challenges and research opportunities:

- *Dynamic molecule balancing:* during the simulation molecules naturally migrate around the system, potentially requiring cores to negotiate the size of the volume they manage.
- *Dynamic rate balancing:* the rate at which reactions within a volume occur depends on the number and balance of local molecules, and "hot" areas will eventually limit the rate of progress within the entire system.

Overall we can expect to see this application scaling linearly with the number of cores in the system; where a traditional multi-core or GPU simulation becomes communication limited, the intrinsic spatial communication capabilities of POETS means the bottleneck is removed.

Use case: Particle & field



Advances in conventional computer technology have made it feasible to simulate the mutual interactions of huge ensembles of particles, but at a massive core-hour cost. By employing sometimes innovative and sometimes brutal approximations, it is possible, for example, to model the migration of proteins through a cell wall at the level of individual particles (where a particle is a group of atoms - a sort of sub-molecule). These computational experiments push at the boundaries of what is possible today - and the further introduction of long-range forces into the experimental regime (for example electrical charge) places many interesting and useful studies out of range. The difficulty here lies in the fact that non-trivial forces extend over many particle-particle separations, making the computational graph necessary to solve the system almost a clique. Any attempt to parallelise such a system computationally rapidly becomes communication bound.

POETS, however, offers a (partial) solution to this. Whilst particle-particle analyses will not map usefully onto a POETS system, an alternative representation, particle & field, does. In a particle & field analysis, ***space is tiled, and each core "owns" a volume***, managing the particles that inhabit that volume. Particles do not, however, interact with each other, they interact with a global field (which may be multi-valued in space):

Particles tell the field how to deform, and
the field tells the particles how to move.

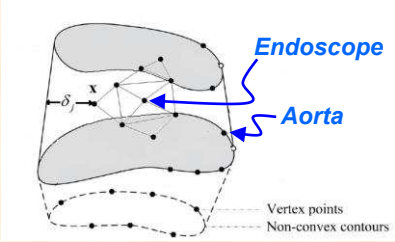
The big difference - from POETS point of view - is that deformations in the field can spread out from their source via core-core communication, and the intensity of the field can be calculated locally (and simply) by the local core - no reference back to the originator of the perturbation is necessary. Particles derive the force incident on them from the local field: again, information that is to hand.

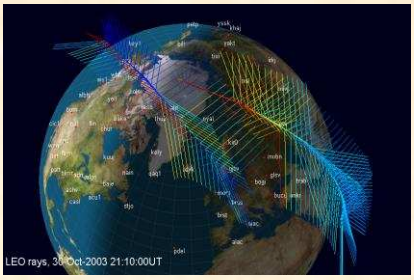
Use case: Industrial image processing

At their core, many industrial problems resolve to **[A]x = [B]** or similar. Whilst matrix solution techniques are the stuff of undergraduate textbooks, industry is interested in matrices of massive ranks (thousands), which are often sparse and ill-conditioned. Further, *sequences* of matrices that represent continuous processes are often mutually inconsistent.

Mapping a core to each matrix element allows the inversion of matrix equations in O(n) time, better than any low-thread solution on a conventional machine. This opens the door to a host of real-time image based applications:

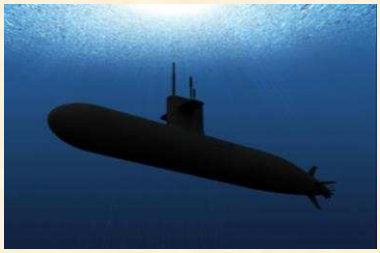
Medical: detailed non-invasive tomographic imaging of biological structures - bones, brains, vascular systems; image-guided surgery. The last requires the reconstruction of images that are noisy and fast moving (typically around 10^6 points/s), where inaccuracy can easily cause death.



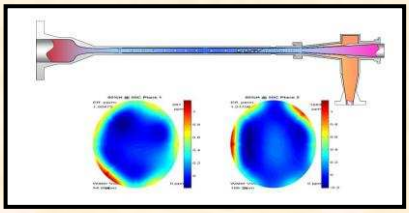


C.N. Mitchell, University of Bath

Measuring ionospheric weather: can decrease the error of GPS fixes by a meter. So what? GPS guided ocean oil drilling costs around 10^6 £m^{-1}.



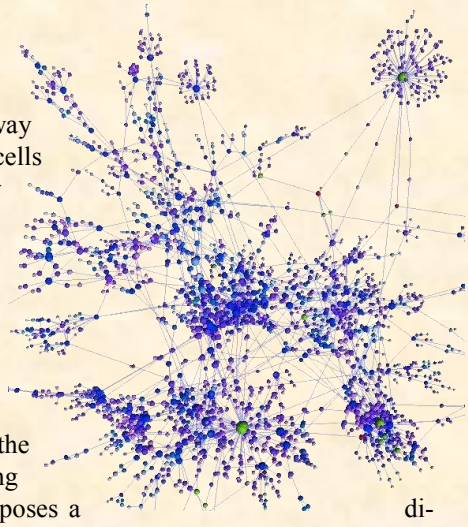
Inverse field problems: detection and location of submerged cylindrical magnetohydrodynamic anomalies



W. Yang, University of Manchester

Production line quality control: mixing efficiency, void detection, structural integrity

Use case: Drug screening



Computational chemistry has a long way to go before the interactions of drugs and cells can be modelled and simulated accurately and usefully at a molecular level. The difficulty arises from the sheer volume of computation necessary to model the interactions of the millions of atoms comprising even the simplest biological drug-relevant system.

The natural response of the simulation engineer in this situation is to increase the level of granularity of the system, modelling at larger and larger resolutions, which exposes a dilemma: the higher the modelling level, the more tractable the total problem, but on the other hand, by coarsening the level of modelling abstraction, interactions were discarded that may turn out to be dominant in some unexpected way; and the ultimate object of simulation is to illuminate interactions that were unexpected. The *art* of modelling for simulation - in any discipline - consists of finding ways to capture relevant interactions as simply as possible without compromising (too much) the representation of reality embodied in the model.

Drug discovery is the process through which potential new medicines are identified. It is traditionally slow and labour intensive, but remains a vital step in the identification of new medicines and treatments. A difficulty arises from the fact that even the simplest drug interacts not only with its primary target (cell), but also with secondary structures - other proteins in other cells. These also interact with each other, in complex ways, making the prediction of the impact of a specific pharmaceutical intervention an almost impossible task, unless the system is modelled at infeasible levels of granularity.

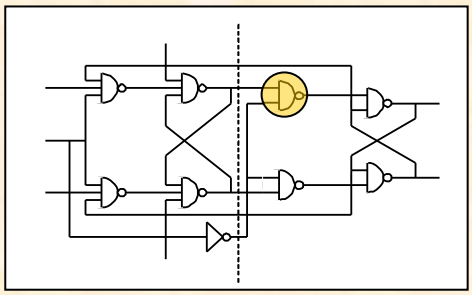
One attempt to overcome this bottleneck employs a radically different methodology to represent a cell and its constituent proteins: a cell is represented by a graph. The nodes of the graph are the proteins contained within the cell, and the edges of the graph the known protein interactions. These edges may themselves be quite complex, to model known adjuvant and chaperoning effects. In a similar manner, a potential drug may be represented by a graph, modelling the effects of the drug on specific proteins. The screening process then involves running a "cross-product" between the biological cell library and the putative drug, analysing the effects by looking at the topology of the affected cell graphs.

None of the steps in this process are particularly individually demanding, but again the difficulties arise from the sheer size of the graphs: a cell graph can contain tens of thousands of nodes. ***Mapping the model graph nodes onto POETS cores*** opens the way to parallelising the graph-graph interactions, with a potentially dramatic impact on computational throughput.

Use case: Discrete simulation

In 1979 - four years before the PC became available - a paper about discrete simulation (*K.M. Chandy and J. Misra, IEEE-T Software Engineering, SE-5 no 5 1976 440-452*) was published by the IEEE, where the authors stated in the abstract:

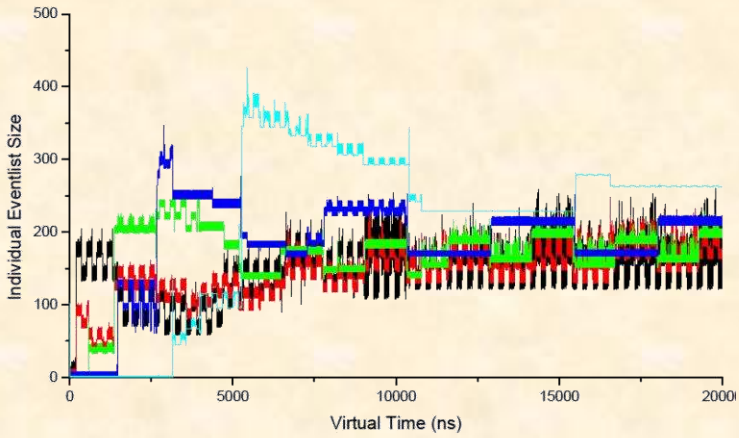
... We propose a distributed solution where processes communicate only through messages with their neighbours; there are no shared variables and there is no central process for message routing or process scheduling. Deadlock is avoided in this system despite the absence of global control. Each process in the solution requires only a limited amount of memory.....



They were talking abut POETS, thirty-six years ago.

The match to the POETS technology is quite remarkable; ***mapping one logical device to each core*** - something unimaginable in 1979 - allows the simulation of industrially relevant systems today. The perennial problem of maintaining overall simulation causality is elegantly overcome by the introduction of timing events that are broadcast along the same signal paths as contained in the circuit under simulation; thus the overhead is an approximate doubling of the signal traffic, a negligible cost considering the speed gearing from all the cores.

Further levels of sophistication are possible: where the circuit under simulation has more devices than the POETS engine has cores - it can happen - we can map multiple devices to a single core, and further, allow the POETS engine to dynamically modify this mapping, load-balancing the simulation on the fly.

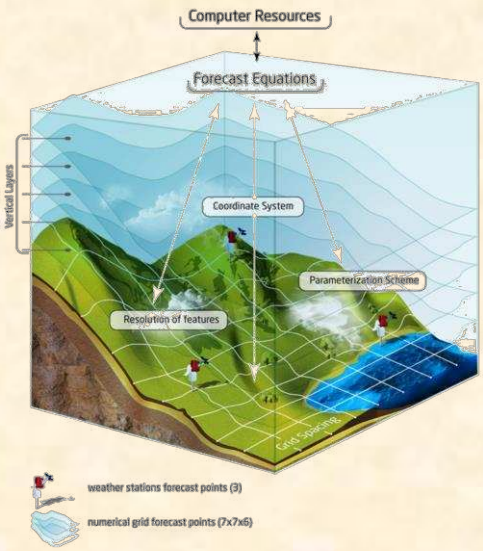


Use case: Weather modelling

Weather modelling involves predicting the interactions of wind, solar radiation, ground conditions, pollution and a host of other features into a numerical model whose state is capable of extrapolation into the future in a computational timeframe that is faster than real time - there is little point in coming up with an accurate prediction of tomorrows weather if it takes two days to do it.

Current methodologies decompose the atmosphere into a non-uniform (multi-scale) grid, and solve the equations concerning the movement of air between grid cells, ensuring continuity of pressure, temperature, density et al across the cell boundaries. The atmosphere over the UK is divided into cells around 1.5 km on a side (giving a UK - based cell count of around 6.10^7 cells); over Europe around 4 km on a side and the rest of the world is modelled at a resolution of around 17 km.

The solution technique revolves around mapping the atmospheric cells onto the available cores of whatever machine is being used to solve the system - there is a tradeoff between cell size (accuracy - pushes the cell size down) and the inter-core traffic load (speed - which pushes the cell size up). Much effort is required to find the 'sweet-spot', resulting in the best accuracy from the fastest cell configuration. (Much effort is also expended in finding better models to represent the atmospheric behaviour, but POETS solves equations, it does not derive them.)



Using ever smaller cells - and ***mapping only a handful of these to each POETS core*** - provides two-fold benefits: the cell-cell traffic maps comfortably onto the hardware routing fabric of the engine (which in any case in POETS engines is hardware and fast); and the equations governing the behaviour of the atmospheric model can become much simpler, as the range scale of the non-linearities intrinsic to the physics become comparable to the new, reduced cell resolution.