POETS: An event-driven approach to Dissipative Particle Dynamics

Implementing a massively compute-intensive problem on a novel hard/software architecture.

Andrew D. Brown

Department of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK. E-mail: adb@ecs.soton.ac.uk

Jonathan R. Beaumont

Department of Electronic Engineering, Imperial College London, South Kensington Campus, London SW7 2AZ, UK. E-mail: j.beaumont@imperial.ac.uk

David B. Thomas

Department of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK. E-mail: d.b.thomas@southampton.ac.uk

Julian C. Shillcock

Blue Brain Project, Ecole Polytechnique Federale Lausanne, 2149 Batiment A1, Station 19, CH-1015 Lausanne, Switzerland. E-mail: julian.shillcock@epfl.ch

Matthew F. Naylor

Computer Laboratory, University of Cambridge CB3 0FD UK. E-mail: matthew.naylor@cl.cam.ac.uk

Graeme M. Bragg

Department of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK. E-mail: gmb@ecs.soton.ac.uk

Mark L. Vousden

Department of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK. E-mail: m.vousden@soton.ac.uk

Simon W. Moore

Computer Laboratory, University of Cambridge CB3 0FD UK. E-mail: s.w.moore@cl.cam.ac.uk

Shane T. Fleming

Department of Computer Science, University of Swansea, UK, Email: s.t.fleming@swansea.ac.uk

HPC clusters have become ever more expensive, both in terms of capital cost and energy consumption - some estimates suggest that competitive installations at the end of the next decade will require their own power station. One way around

this looming problem is to design bespoke computing engines, but while the performance benefits are good, the design costs are huge and cannot easily be amortized. POETS (Partially Ordered Event Triggered System) - the focus of this paper - seeks to exploit a middle way: the architecture is tuned to a specific algorithmic pattern, but within that constraint, is fully programmable. POETS software is quasi-imperative: the user defines a set of sequential event handlers, defines the topology of a (typically large) concurrent ensemble of these, and lets them interact. The 'solution' may be exfiltrated from the emergent behaviour of the ensemble. In this paper, we describe (briefly) the architecture, and an example computational chemistry application, dissipative particle dynamics (**DPD**). The DPD algorithm is traditionally implemented using parallel computational techniques, but we re-cast it as a concurrent compute problem which is then ideally suited to POETS. Our prototype system is realised on a cluster of 48 FPGAs providing 50K concurrent hardware threads, and we report performance speedups of over two orders of magnitude better than a single thread baseline comparator, and scaling behaviour that is almost constant. The results are validated against a "conventional" implementation.

CCS Concepts: •Hardware~Emerging technologies~Analysis and design of emerging devices and systems; •Computer systems organization~Architectures~Parallel architectures~Multicore architectures; •Computing methodologies~Parallel computing methodologies~Parallel algorithms; •Computing methodologies~Parallel computing methodologies~Parallel algorithms~Self-organization

Additional Keywords and Phrases: Parallel architectures, event-driven processing, massively micro-parallel systems

1 Introduction

High performance computers get ever more powerful, year on year, and this brings obvious and tangible benefits: existing problem sets can be resolved faster, and larger and larger problems become - for the first time - computationally (i.e. financially) tractable. However, the capital costs of building large machines, and the energy costs of running them, are growing non-linearly, and the concomitant advances in performance are not keeping pace. One of the reasons for this is that HPC machines are general purpose - they are designed (unavoidably) in the absence of knowledge of their intended application domain. They do everything, and inevitably as a consequence, they do it sub-optimally.

One strategy for overcoming this is problem-specific bespoke hardware, but the roadside of this path of history is littered with machines whose putative performance improvement was overtaken by that of conventional architectures before they were commissioned [1-4]: in this paper we describe a compromise approach.

Algorithms well-suited for HPC are obviously eminently parallelizable, and much research effort goes into casting domain-specific problems into a form that can capitalize on the capabilities of the underlying hardware [5,6]. One of the goals is to minimize the overhead cost of message choreography. This paper describes a compute architecture designed to address not a particular *domain*, but a large though specific compute *pattern*. POETS is an *event-based* computing architecture. Like Message Passing Interface (MPI) programs, multiple processes are executed simultaneously (in POETS, threads) communicating by messages (in POETS, packets). Unlike MPI, POETS packets are small (64 bytes), fixed size, hardware brokered, and fast due to the absence of a software infrastructure - thus the overhead of message choreography is avoided. Similar to GPGPU systems, the POETS hardware architecture contains many (thousands) of computing units, but unlike GPGPUs, these are fully-fledged distributed cores, in our case RISC-V that scale naturally to large numbers of chips optimized for communication.

POETS is ideally suited for compute problems that can be addressed by decomposing the problem domain into a large (but not necessarily regular) mesh of nodes, and allowing these nodes to communicate state changes *asynchronously* with their neighbours. It should be stated up-front that problem classes exist for which POETS is extremely *ill*-suited: algorithms with long, narrow data-flow graphs do not fit well because the underlying fine-grained massively concurrent hardware cannot be fully exploited. That said, there are many problems with short, wide data flow graphs that do suit POETS well, including the case study presented here.

Computational chemistry and its younger, less expensive sibling, Dissipative Particle Dynamics (DPD) is a computational technique that - broadly - solves the equations of motion of biochemical particles (beads), models their

local interactions, and displays emergent properties of the system [7,10,30]. The technique tracks the behaviour of individual molecules/beads and allows the human user to view, for example, via simulation, a virus passing through a cell wall - biochemistry in a computer. The technique requires the calculation of the physical trajectories through phase[1] space of (at the very minimum) hundreds of thousands of particles, and the corresponding computing costs are massive.

In this paper, we present

- A (brief) description of the POETS hardware/software stack.
- A (brief) description of the underlying DPD algorithm. This is described in detail elsewhere [7,10,11], and we make no claim to contribute here to the methodologies of computational chemistry. It is, however, a typical, important and compute-heavy example of the type of problem for which POETS is ideally suited.
- A description of how this (kind of) algorithm may be re-cast to fit the POETS architectural capabilities (POETS-DPD).
- A performance analysis of POETS-DPD, and comparisons to existing HPC systems running conventional DPD simulation algorithms.

The novelties of POETS-DPD are two-fold: absolute speed and near constant scaling (see Figure 15).

In more detail, the paper consists of 8 sections. Section 2 provides a description of the POETS compute stack, both hardware and software infrastructure. A description of the POETS computing model is included here. Section 3 contains a description of the DPD approach to a simple computational chemistry problem, and section 4 describes its implementation in both a simple parallel environment and a more current heterogeneous GPU-based system. In section 5, we describe how a suitable DPD subset (chosen for simplicity of explanation) may be re-cast into a form suitable for execution on the POETS architecture. In section 6 we provide an analysis of performance metrics and measurements, illustrating the power of the techniques embodied in POETS. We give a quantitative description of the speed advantage of this technique: three (initially mixed) immiscible fluids phase separating. Over a simulation interval of 10000 timesteps, POETS performs over two orders of magnitude faster than a single-thread COTS platform, and outperforms parallel COTS implementations. Section 6 also includes an analysis of the electrical power requirements. Further application domains for this technology are briefly discussed in Section 7. Section 8 contains final comments.

2 POETS ARCHITECTURE

The POETS message-passing architecture is subtly different from its peers. It is ideally suited (and intended) for *simulation* problems based on large graphs (known as **application graphs**), where small, independent tasks (**devices**) are connected by **edges** carrying small, atomic, asynchronous **packets** [9,12]: examples being neural simulations (massively non-isotropic application graphs with device degrees from 1 to ~10^5) [13,14] and finite difference methods (more uniform graphs) [15]. The graph can represent the physical topology of the system under simulation (in which case the connectivity is arbitrary) or we can tile some (arbitrary dimensional) space, and use each *device* to represent the behaviour of the system in that particular spatial unit cell. (In the example chosen as a sheet-anchor for this paper, we tile three-dimensional space with conventional cubes, allowing each cube to interact with its (26) nearest neighbours. The application graph in this case is thus highly regular, but this is irrelevant to the functioning of the system.)

The behaviour of each device (they need not all be the same) is captured by a set of **handlers**. These are small sections of executable code that *react* to incident events. An event is delivered to a device (by the POETS hardware infrastructure); the appropriate handler is awoken by the arrival of the event, and it executes. During the course of the execution it may modify its own (internal, persistent) state, and/or it may emit packets of its own. *Thereafter it returns to quiescence, awaiting the arrival of another packet*. The entire compute trajectory is **event-driven**. There is no notion of global state or wall-clock synchronization of the handlers - the final solution is distributed throughout the local memories of all the devices, and may be exfiltrated for conventional visualisation.

[1] The phase space coordinate of a particle is a 6-dimensional vector consisting of 3 orthogonal position components and 3 associated velocity components.

2.1 Design intent

In a massively parallel unsynchronized system of thousands of physical cores and millions of logical devices (POETS), at every point in time some part will be communication limited and some part compute limited; attempting *a priori* scheduling on this scale is an unwinnable battle. We can here usefully identify the concept of "**design intent**": the hardware (and software) of a performant system will utilize heuristics - usually derived from some variant of Amdahl's law - to generate high throughput for *most* processes *most* of the time. If the software designer stays within the design intent of the system designer, programs will perform well. If the software architect strays from this design intent, the process will still execute correctly, but the performance will be disappointing. The design intent (recall POETS is not a general purpose machine) is that the rate-limiting steps flicker and move about the architecture as execution proceeds. We cannot mandate this behaviour, but we can influence it at setup (via the configuration subsystem) and we can monitor the packet fluxes and handler behaviour as execution progresses, to better understand the dynamics of the system.

With POETS, then, the design intent is that:

- The application graph is extremely large.
- The communication infrastructure (hardware) supports multiple, parallel communication patterns.
- The (user defined) event handlers will be *small*, *fast* and approximately the same *size*.
- The individual device states will be small.
- The program structure and underlying numerical methods are such that the overall system will produce a physically sensible result.

Stepping outside these intentions (at least for the first three bullet points) will produce a system that still works (in the sense that it produces a correct result), but probably has little or no speed advantage over a conventional platform. POETS is based on the idea that the cores are (almost) free, and the wall-clock budget is spread (more or less) evenly between communications and compute.

2.2 The compute stack

The POETS system is a platform consisting of a network-optimized FPGA cluster containing thousands of custom multi-threaded RISC-V **cores** [8], arranged in a hierarchy. An overview of the system is shown in Figure 1. At the lowest level of the hierarchy, the behaviour of an individual device is represented by a set of *software handlers*. A device may have more than one handler, to react to different types of incoming packet. Multiple devices may be mapped to a single thread: device handler code is grouped together and linked into binary images called **softswitches**. As the name implies, these are little more than small pieces of switching code that direct events landing on the softswitch to the appropriate device handler. The behaviour of the softswitch is outlined in Figure 2. The underlying hardware communications infrastructure exposes a set of registers to the softswitch (the interactions labelled "hardware" in Figure 2). Strictly, the softswitch is a spinner, but the *design intent* is that it spends most of its time in the wait state at the right of the figure. **`OnRecv`**, **`OnIdle`** and **`OnSend`** are handlers provided by the user (any/all may be omitted) and the rest of the code is boilerplate provided by the system. (The boxes labelled [**I**] are points at which *instrumentation* packets may be assembled and passed up the compute stack to allow the user to visualise the behaviour of the system.) At this low level, the behaviour of the (<1024) devices in a softswitch binary are serialized: thus the configuration subsystem (see section 2.3) will attempt to distribute device behavior uniformly across the available compute fabric to minimize this effect. The intention here is that there will be ~1 device per softswitch, so there exists no serialisation. (The experimental results in section 6.3 demonstrate the consequence of straying away from this.) At the next level in Figure 1, a softswitch binary inhabits a **thread**, which is one of 16 running on a RISC-V **core**.

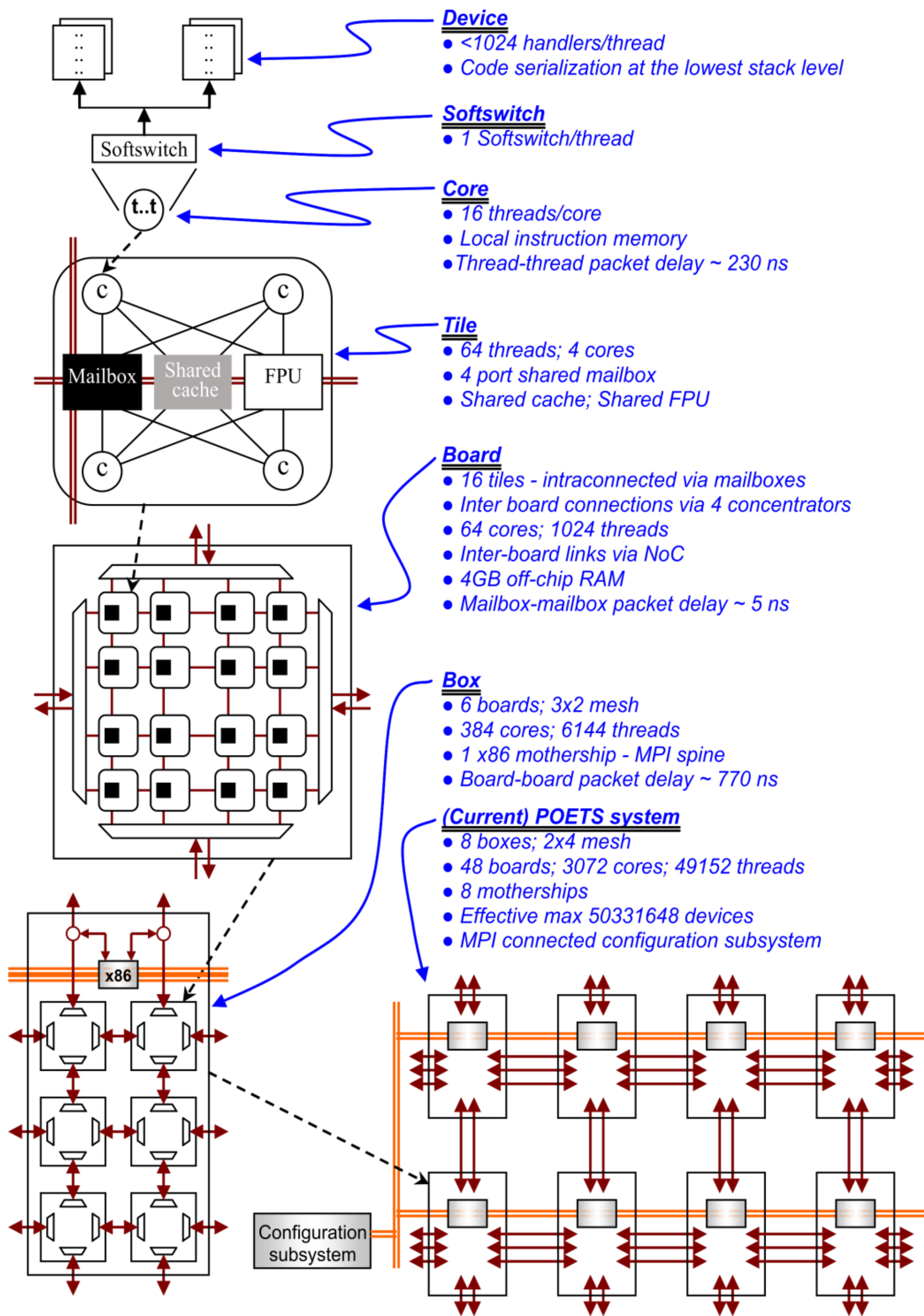


Figure 1: **The hierarchy of the POETS hardware, from device handler to complete (current) system.**

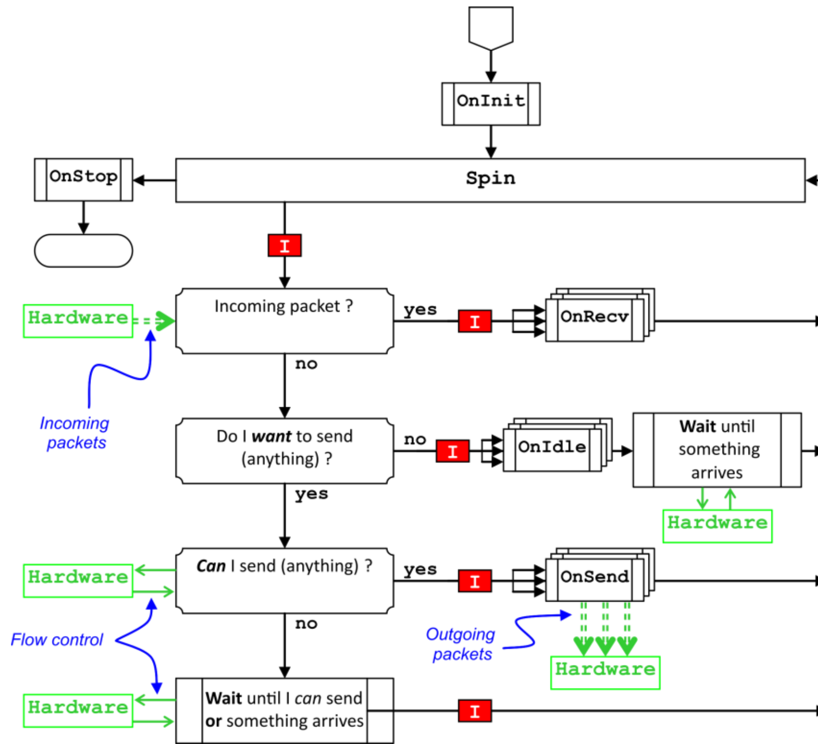


Figure 2: Abstract SoftSwitch (a spinning device multiplexer) model.

At the next stage of the hierarchy (the first hardware level in Figure 1), four cores are assembled with a mailbox, a floating point unit and a memory cache to form a **tile**. The memory cache connects to 4Gbyte (the system has 32-bit addresses) of external DDR memory; the mailbox brokers communication with other tiles. Sixteen tiles are realized in a single DE5-Net (Stratix-V based) FPGA board, interconnected in a canonical 4x4 grid. Each board has four 10 Gbps links which connect to the edges of adjacent boards, extending the communication network across any number of boards. Topological variations are discussed briefly in section 2.2.1. Six boards are assembled into a box (this figure is driven by thermal considerations), along with a conventional x86 machine (the **mothership**), used for configuration and data in/exfiltration. The motherships are connected together via a conventional MPI network; the mothership processes themselves are multi-threaded, allowing the X86 machines to be effectively event-driven: *here* an event can be a POETS packet from a RISC-V or an MPI message from another mothership[2]. POETS contains two closely coupled networks - Figure 3 - one consisting of X86 motherships (MPI) and one of RISC-V POETS cores. Further details of the message-passing architecture can be found in [38].

[2] The MPI universe (Figure 3) contains a user control console process, an arbitrary number of instrumentation display processes, processes to broker interaction with the outside world (supporting near real-time interaction with neural network applications) and network storage.

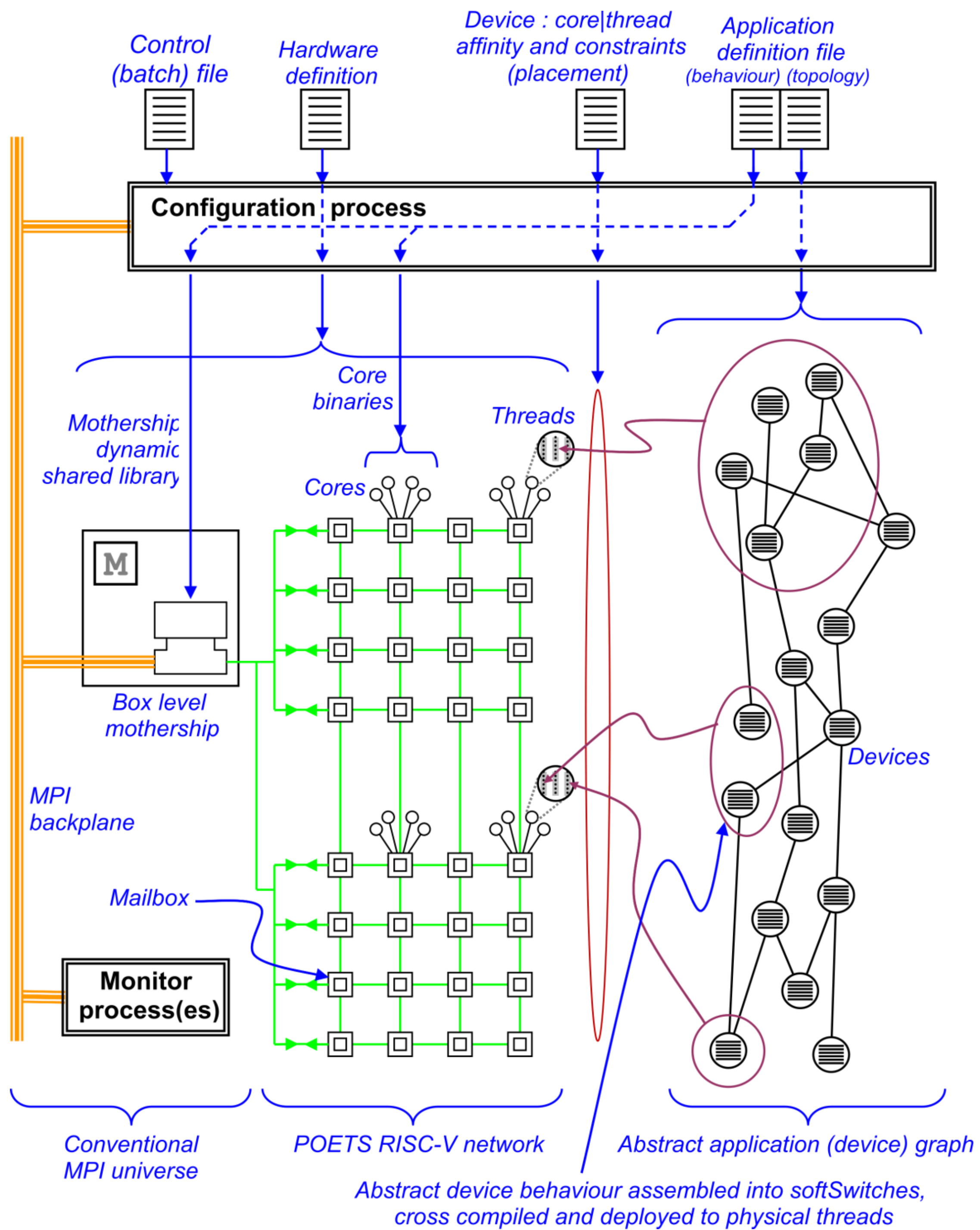


Figure 3: Overall system layout.

Finally, boxes may be assembled almost arbitrarily. The intention here is that the system should physically scale gracefully to however big is necessary for a given problem. The current packet size is 64 bytes, and this has to hold a payload plus a system-wide unique *device* address: (device offset || thread || core || board). Ultimately the size limit will probably come from power or, more prosaically, capital cost.

Currently, a single POETS box provides a total of 6,144 hardware threads, and we have a cluster of 8 boxes, connected in a 2 x 4 mesh, giving a total number of 49,152 threads. With a thread occupancy of one device, we have 49,152 truly parallel compute units. Exploiting the serialization provided by the softswitches allows application graphs of tens of millions of vertices - much larger systems are possible, but the serialization has a serious impact on performance, in the form of cache thrashing and FPU contention.

It is worth looking at the low-level architecture in a little more detail: the principal design criteria for POETS is of an extremely high number of very simple, fast compute units, embedded in a fast hardware communication fabric, and the problem solution is generated as an emergent property of the (massively parallel) interactions between these small units; for minimum wall-clock execution, serialization is to be avoided if possible. There are two potential areas where this can occur:

- (Figure 1, Devices) The softswitch is essentially a (sophisticated) multiplexer, containing the code and local state data of every device assigned to it by the configuration system. If a softswitch contains more than one device, serialization is inevitable. However, the design intention is that each softswitch ideally contains 0 or 1 devices; the ability to handle more is incorporated to support the inevitable situations where the application graph is too large for this ideal situation to pertain. Serialization in the softswitch is thus regarded as a fall-back situation where the application is bigger than the physical compute hardware (analogous to a virtual memory system thrashing in a conventional machine). It can be ameliorated simply by adding more boxes to the system.
- (Figure 1, Cores) Multithreading: The core pipeline will in general contain multiple instructions from more than one thread, and data and control hazards in one thread can impact on the throughput of *all* the threads with instructions in the pipeline when the hazard occurs. The POETS core pipeline (Figure 4) has *at most* one instruction per thread in the pipeline at any one time, thereby eliminating any control or data hazards and corresponding pipeline flushes. The system achieves maximum throughput when there are *at least* as many unblocked threads as there are physical pipeline stages.

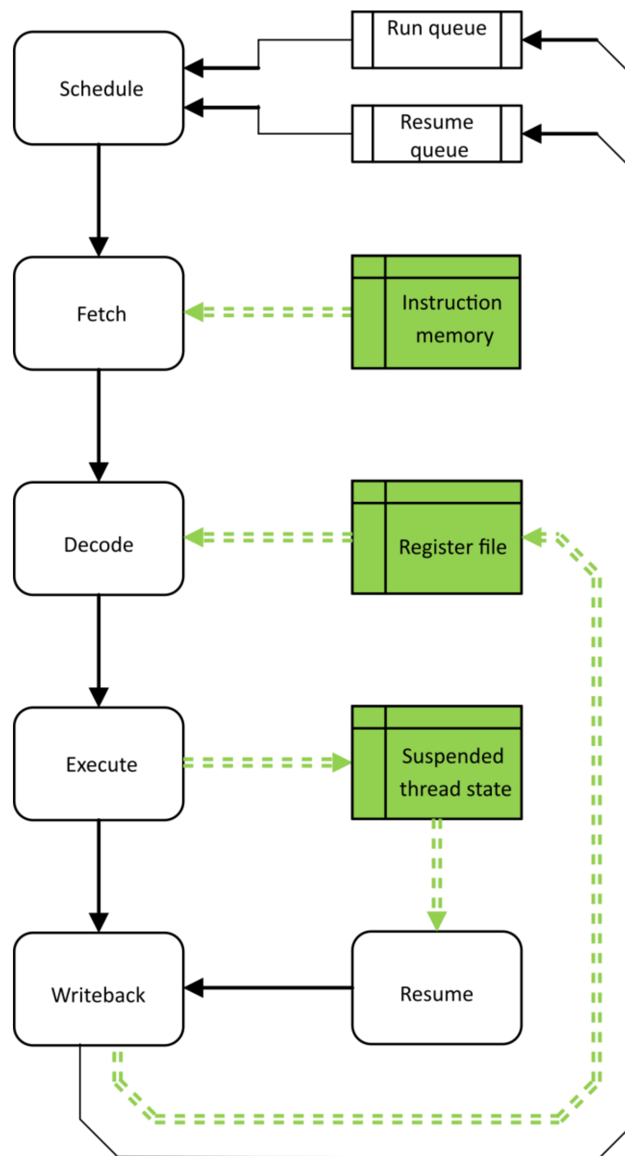


Figure 4: POETS core pipeline: ≤ 1 instructions/thread present at any time, resulting in a compact, fast core with high throughput when |threads| > |stages|.

POETS consists of a large number of compute units, embedded in a custom hardware infrastructure that brokers the asynchronous transfer of small packets of information between device handlers. As with any system of this nature, if the rate of packet injection exceeds the rate of packet consumption - locally or globally - there is a problem. The problem can be *moved* (but not solved) by the strategic insertion of buffers (hardware and software); it can be *solved* (albeit rather drastically), by simply dropping packets in congested areas, as long as the high-level analysis can be designed to be tolerant of this [13,14,15]. POETS takes a middle route: the handlers *request* permission from the network to send a packet. The network will only grant permission to send if it can guarantee successful delivery. Again, this merely moves

the problem, but it moves it (back) to the point that is most capable of behaving responsibly with the information: the putative sender can (1) continue to attempt the send (2) abandon the attempt (3) do something else (4) attempt to send to another recipient - or any combination of these. This is discussed further in section 5.2.

2.2.1Variations on the hardware.

The principle point of this paper is to describe the behaviour of DPD on an event-triggered architecture, in particular the (near constant) scaling behaviour - see Figure 15. To this end, all the experimental results presented in section 6 are taken from a consistent architecture (modulating the overall hardware system *size* by changing the number of boxes - Figure 1 - involved). POETS has evolved through multiple architectures, from the SpiNNaker torus [13,14,15], including Figure 1 here and those in [49,50]. The configuration used here supports/exploits:

- Dimension ordered routing (hardware brokered and fast).
- Avoidance of routing deadlock (under all possible problem placements - Figure 5).
- Back-pressure controlled packet transmission (section 5.2).

We note that alternative topologies we have explored show no substantial speed advantages. That there are better configurations is always possible - this is an active research thread. The algorithms presented here do not depend on the underlying processor technology.

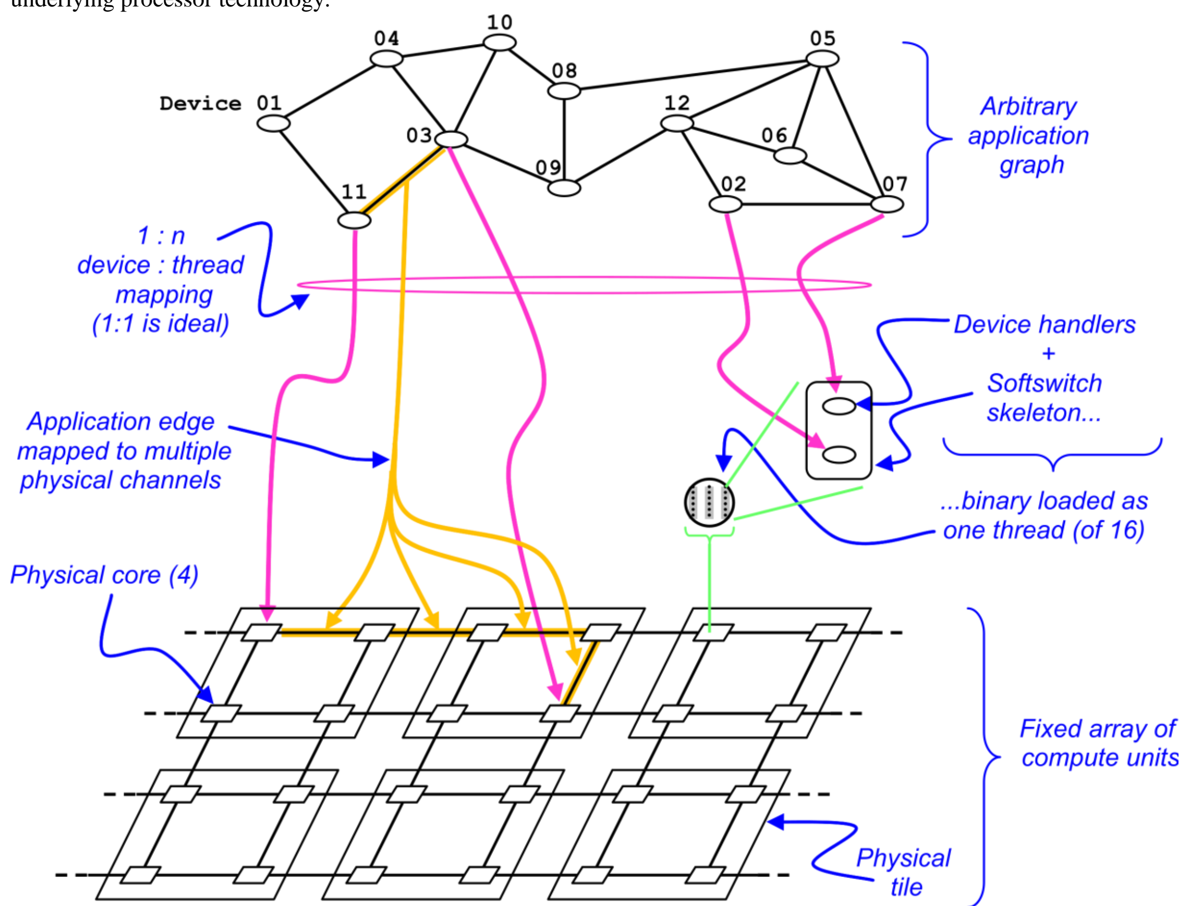


Figure 5: The arbitrary device application graph is mapped to the regular underlying POETS hardware.

2.3 Configuration

The hardware is a fixed (notwithstanding the intrinsic reprogrammability of the underling FPGA fabric), usually but not necessarily regular, array of compute units, arranged in the hierarchy outlined in Figure 1. The application graph is problem specific, of *arbitrary* user-defined topology and size. The latter must be mapped to the former - see Figure 5 - and this is the responsibility of the *configuration process* (Figure 3). In a conventional compute environment, the compile/link stage has a very specific function; in a massively micro-parallel system (POETS, SpiNNaker [13,14], GPGPUs) the "compiler" is more complex, and includes a device to core mapping stage. The configuration system is run on an external x86 conventional machine (which is part of the MPI universe containing the Box motherships in Figure 1). It performs the following tasks:

- The application graph topology and handler code definitions are loaded, parsed, and assembled into a datastructure. Note that the handler code will be quite small (there are few *types* of handler) and the graph definition potentially massive.
- A (hierarchical) definition of the hardware is loaded, and the application devices mapped to the core threads. This is more analogous to the placement problem of IC and PCB design than the core affinity problem [16]; we currently use an algorithm derived from the Kernighan-Lin placement algorithm [17,51] and simulated annealing [52]. We convert the application graph (a 3D connectivity mesh) and place it hierarchically using METIS [51]. The graph is placed at three levels of granularity (refer to Figure 1 - the numerical parameters cited here refer to the 48 board system).
  1. The entire graph is partitioned into 48 subgraphs, minimising the inter-FPGA mapped edge density.
  2. The devices allocated to each *FPGA* are themselves allocated to the 16 tiles.
  3. The devices allocated to each *tile* are clustered into 64 threads inhabiting the 4 cores of each tile.

  Unsurprisingly, by far the most important of these is step 1, minimising the inter-FPGA traffic. The overall throughput of the system is relatively insensitive to the lower levels (2 and 3 above) of the hierarchical decomposition.
- Once this is done, each of the handlers (these are fragments of C) mapped to a given thread is assembled with a system-supplied softswitch skeleton, and cross-compiled and linked on the X86 platform into a RISC-V binary. Once assembled, the binaries are distributed over the MPI spine to the motherships, each of which has access to the RAM associated with each Board in its Box (Figure 1). This is analogous to the cross-compilation and download steps common in GPGPU configuration [18].

2.4 Execution model

POETS operates in a similar manner to a large conventional compute platform, in that multiple cores (in the case of POETS, compute *threads*) support the parallel execution of multiple control trajectories. (In one sense, POETS is simpler: there is no overseer operating system to dynamically control the process to core mapping - device code is statically bound to a physical POETS core by the configuration system.) In a conventional MIMD system, processes are mapped to cores (core affinity masks allowing), and there is no requirement other than efficiency for the number of processes to be less than the number of cores. Parallel (usually MPI) universes communicate via arbitrary sized messages - brokered by software - and have access to a monolithic or partitioned shared memory space. The communication patterns are usually complex, and require significant human design input - Figure 6(a).

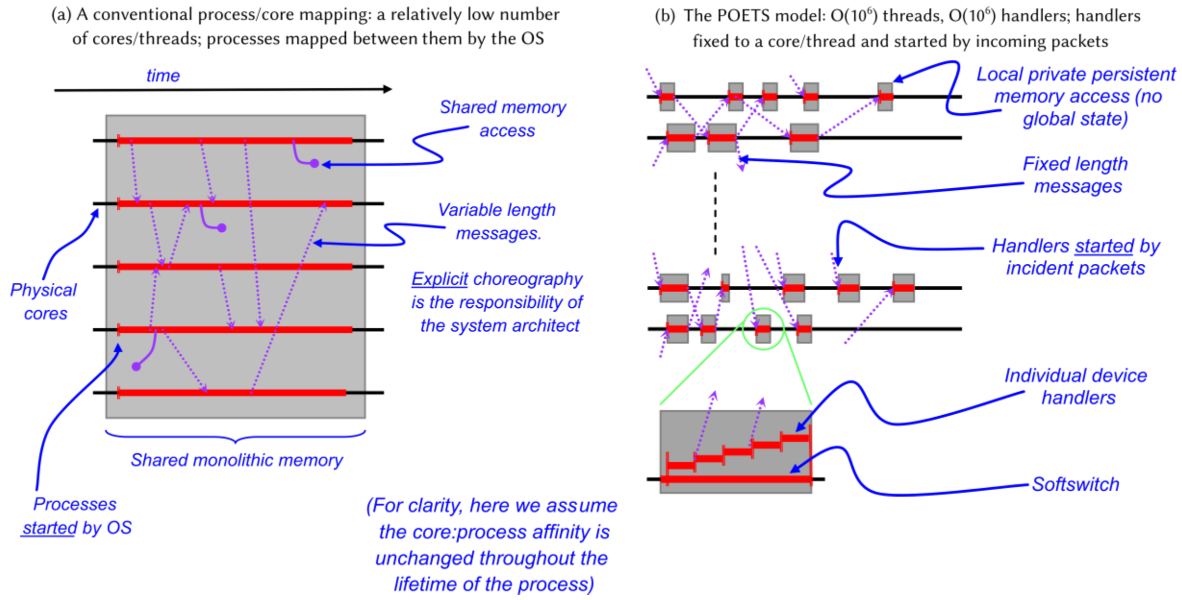


Figure 6: **Conventional execution model (a) and the POETS execution model (b).**

In POETS, device handlers (or more accurately, softswitches containing device handlers) are bound to a specific thread at configuration time, and cannot change affinity. The flow of execution through the machine is depicted in Figure 6(b). Each handler has a small, persistent, private memory, and starts when invoked by its parent softswitch due to an incident event. During handler execution, the device may realise it needs to send another message, but in order to avoid deadlock it cannot freely send messages. Instead, it will indicate that it is "ready to send", which notifies the softswitch that once network space becomes available it should call the relevant send handler on the device.

The hardware is a physical representation of a Hasse diagram [30,31]. Returning to the idea that cores are effectively free (or at least, the cost is not a first order barrier to research), if we can describe an application in terms of its dataflow graph (Hasse diagram) we can *physically* implement the application by building the dataflow graph, and represent each node by the appropriate functional unit (the handlers). This idea is at the heart of high-level behavioural synthesis [32-34] where the mathematical functional units are implemented as their exact physical counterparts; in POETS, we move the level of granularity up so that *clusters* of functionality are implemented as POETS *devices*. Each device handler is a fragment of sequential code, but there are many thousands of them, all running in parallel.

Execution is akin to a hardware-supported numerical relaxation process. A device changes its state, and communicates this change to its computational neighbours. They, in turn, may change their state, and this change in state is also communicated. The intention is (Section 6.2 provides evidence to support this in the context of this paper) that *local* compute provide changes to *local* state in a manner that ultimately results in a *global*, physically realistic solution.

2.5 Application deployment

The process of implementing an algorithm on a POETS machine is not simply a question of porting pre-existing code. However elegantly structured, an algorithm implementation must be stripped back to the underlying mathematics and re-cast in a form suitable for the POETS compute architecture. However, this is a minor development cost to be set against the significant wall-clock speed gains. Conventional approaches (MPI, CUDA) have major design costs, in that they are effectively *two* programs: one to solve the problem locally (within a process) and a completely different one to construct the global solution via inter-process communications. POETS merges, flattens and simplifies these problems.

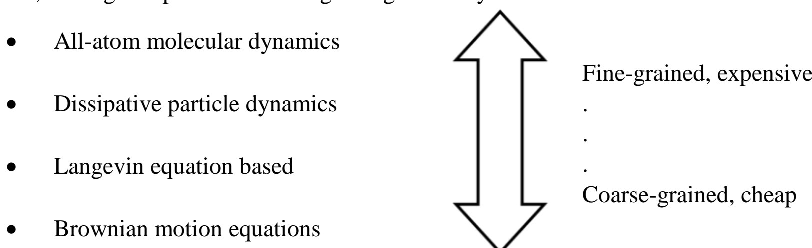
2.6 Instrumentation

Designing software to run on massively parallel microsystems such as POETS, where there is no 'overseer' operating system, the communications system is non-deterministic (from the perspective of the motherships) and there is little internal visibility or debug is challenging. Sniffing individual packets is possible, but usually unhelpful, partly because of the sheer volume of data traffic, but mainly because the *act* of data exfiltration perturbs the very dataflow we are trying to inspect.

To support some visibility of the dataflow, the softswitches can be ordered to launch instrumentation packets (**I-packets**) ([**I**] in Figure 2) to their motherships at (programmable) intervals. I-packets contain performance statistics: packet/loop counts, cache misses and handler invocation counts. Although this data is rather coarse and generic, it is possible to form a general view of the activity of the system. (It is also possible for specific handlers to send their own application specific diagnostic packets to the motherships.) Once in the motherships, the I-packets can be processed and displayed by a (conventional) monitor process attached to the MPI universe - Figure 3.

3 DISSIPATIVE PARTICLE DYNAMICS

Here we describe the Dissipative Particle Dynamics (DPD) formalism. Before the advent of powerful computing technologies, the only way to predict the outcome of molecular interactions was by making use of theories that operated on the macroscopic properties of the constituents. Emergent properties of the whole were predicted from the (measured or predicted) properties of the parts, with all the inaccuracies that that entails. Clearly, the finer the detail of the constituent model, the more accurate it is, and the more meaningful the corresponding results. The problem lies in the *range* of problem component definitions necessary: if we model a chemical compound at the level of atoms, we can expect sensible interactions predicted at atomic time and distance scales. The problem becomes significant because emergent properties (the macroscopic outcome of any reaction) only become evident in massive ensembles of atoms and over huge timescales: you cannot predict much about the gross properties of concrete by simulating the hydration of a single calcium oxide particle. We live in interesting times: computer capabilities grow year-on-year, and current technology is *almost* at the point of being able to simulate the behaviour of chemical reactions (all-atom molecular dynamics) at scales that are biologically and materially interesting. This aside, approximations must still be made to get results in reasonable compute times. Particle-based simulations all integrate some form of Newton's laws, and there is a fine/coarse spectrum, trading computational cost against granularity:



The bottom two are macroscopic techniques, and the top, all-atom, is *massively* computationally expensive.

DPD, originally proposed in [10-11], is a particle-based Newtonian march-in-time simulation scheme that was developed to allow simulations of complex fluids on near-macroscopic length scales while retaining near-molecular details. The forces between particles are soft[3], which increases the size and time scales of systems that can be simulated using reasonable computational resources. DPD simulations are most commonly performed in a simulation box with constant volume, constant number of particles, and constant temperature. The simulation box is subject to periodic boundary conditions to avoid artefacts due to hard walls: a particle that moves beyond any boundary of the simulation box instantly reappears at the opposite face (Figure 7 and Figure 8).

[3] A *hard* molecular dynamics force has infinite repulsion at zero separation, typically via a $1/r^6$ term. A *soft* force remains finite even if two particles occupy the same position in space.

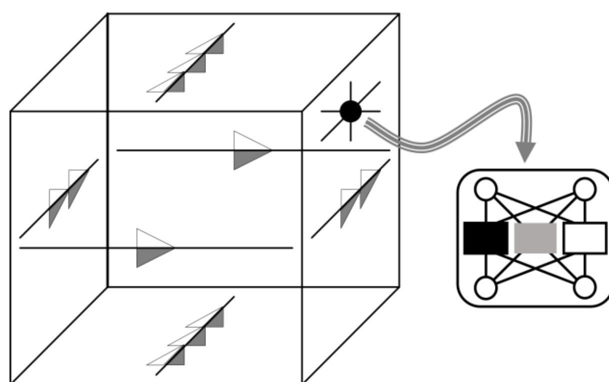


Figure 7: **Triple torus application graph mapped to the compute stack.**

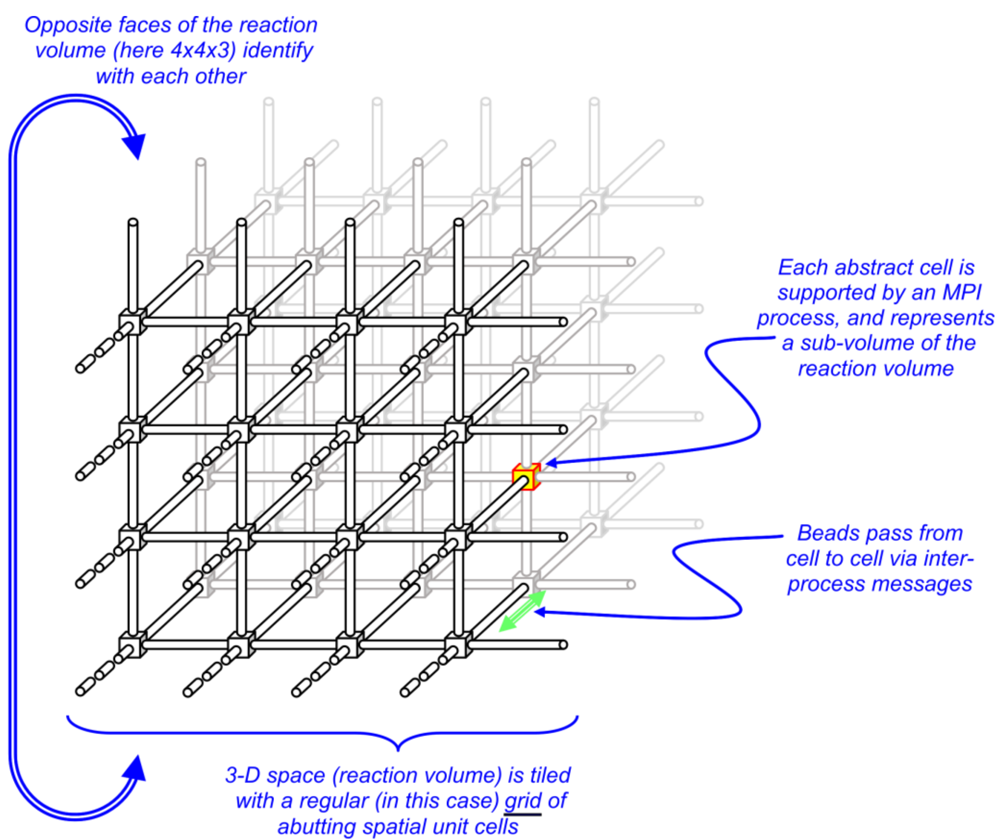


Figure 8: **DPD data layout in a parallel environment.**

DPD is commonly used to simulate dynamic chemical and biological systems whose interesting behaviour emerges on length scales of microns and timescales of hundreds of microseconds. This typically requires following the evolution of tens of millions of particles for millions of integration steps, and leads naturally to the use of a parallel version of

DPD. The communication cost involved in calculating the inter-particle interactions constitutes the main bottleneck in applying this technique.

In DPD, particles do not represent atoms, but *groups* of atoms or molecules. These particles are conceptualized as spheres, and more commonly referred to as **beads** (see Figure 9). With this we sacrifice the fine detail of the molecules and ligands, but reduce the compute load significantly. A bead might be an entire water molecule, or an assembly of atoms within a larger molecule.

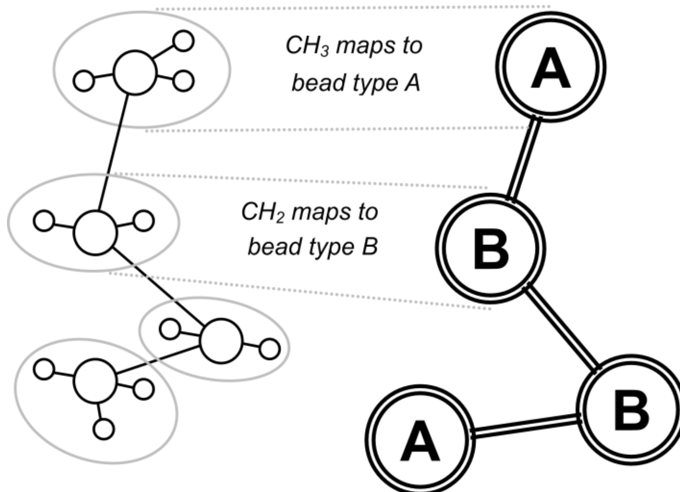


Figure 9: The structure of linear butane (14 atoms: $CH_3CH_2CH_2CH_3$) maps to just 4 DPD beads A-B-B-A.

Each bead $\boldsymbol{i}$ has an associated point position vector $\boldsymbol{r_i}$ within the simulation volume, and a vector $\boldsymbol{v_i}$ holding its velocity at the current timestep. (Together these make up the phase space coordinate of the bead.) A non-trivial simulation typically has multiple bead types, for example water (one bead) and oil (a composite of beads). Each type has properties which affect how they interact with the other types. Bead interactions are considered in a pairwise fashion; and arise from different sources:

- Non-bonded (chemical potential) bead-bead forces
- Bead-bead Hookean chemical bonds (2-body)
- Bead-bead torsional chemical bonds (3- and 4-body)

The latter two components are obviously essential for non-trivial systems, but *we will ignore them here*, as all beads take part in non-bonded interactions, while only a small fraction take part in bonded interactions – as a result non-bonded interactions dominate compute time. The focus of the paper is on how the data representing the system and its interactions may be organized from an event-driven perspective to produce the wall-clock speedups we report. Ultimately all the forces above resolve to a single force on each bead; the exact makeup of the force is not relevant to this paper.

Restricting further discussion to non-bonded forces, the total force acting on one bead by every other bead is used to find changes in velocity, so in principle, every possible *pair* of beads in the simulation must be considered. However, as beads in DPD represent fluid elements that contain several molecules or molecular groups, their interactions are limited to short range forces across their boundaries. All DPD non-bonded forces are therefore short-ranged and vanish beyond a cut-off distance $\boldsymbol{r_0}$ [7]. This is captured by the function that defines the force between two beads $\boldsymbol{i}$ and $\boldsymbol{j}$ separated by the Euclidean distance $\boldsymbol{r_{ij}} = \boldsymbol{r_i} - \boldsymbol{r_j}$

$$F_0(r_{ij}) = \begin{cases} 1 - \frac{r_{ij}}{r_0} & r_{ij} < r_0 \\ 0 & r_{ij} \geq r_0 \end{cases} \qquad (1)$$

The bead-bead non-bonded force has three components:

Conservative force component:

This force gives beads a chemical identity, for example hydrophobicity.

$$F_c = aF_0(r_{ij}) \qquad (2)$$

For an analysis consisting of more than one bead species, the coefficient $\boldsymbol{a}$ is replaced with a (symmetric) matrix, $\underline{\boldsymbol{A}}$, that captures the mutual affinities between species.

Dissipative force component:

This represents unobserved degrees of internal freedom, and *damps* relative momentum.

$$F_D = -\gamma F_0^2(r_{ij}) \cdot \underline{r_{ij}} \cdot \underline{v_{ij}} \qquad (3)$$

Random force component:

This *creates* relative momentum between two interacting beads, and together with the dissipative force constitutes a thermostat that maintains the system temperature constant throughout the simulation.

$$F_R = \sigma F_0(r_{ij})\xi \qquad (4)$$

The coefficients $\boldsymbol{\gamma}$ and $\boldsymbol{\sigma}$ are not independent; numerical details may be found in [7]. $\boldsymbol{\xi}$ is a noise source with Gaussian statistics. These equations are extremely numerically simple, and chosen for ease of computation; it has been shown [19] that as long as certain structural constraints are met, they are almost arbitrary. Having calculated the force on every bead at a timepoint, Newton's equations of motion may be integrated discretely to give the trajectory of each bead through phase space. An example of POETS-DPD used in a chemical physics application maybe found in [39].

4 IMPLEMENTING DPD IN A traditional PARALLEL ENVIRONMENT

Implementing DPD in a traditional parallel compute configuration is done in much the same way as many particle-particle and particle-field simulations [20].

4.1 Data layout

The system under simulation (**SuS**) reaction volume is tiled - usually, but not necessarily, with cubes (Figure 8). Opposite faces of the composite cube are identified with each other (Figure 8) so that the simulation environment is isotropic from the point of view of the beads. How many cubes depends largely on how many processors are available, and the number of beads hosted by each processor/cube depends largely on the amount of compute wall-clock available. Typically there will be 10^4-10^5 beads/cube. Each processing cube will have a **halo** (or *ghost*) layer immediately inside its boundary, $\boldsymbol{r_0}$ (equation 1) thick.

At the start of the simulation, the beads are distributed randomly in space, and with some velocity distribution that approximates to the expected distribution in reality. In general, this distribution will not represent a thermodynamic equilibrium, but it will be close.

The simulation timestep ($\boldsymbol{h}$) is chosen *a priori* and not usually altered throughout the course of the simulation. (This is a safe assumption, because the equations of motion in DPD are not mathematically stiff.)

4.2 Simulation

The goal of the simulation is to follow the phase space trajectories of each bead in the system, sampling the state periodically to measure any emergent properties of the bead ensemble. In précis:

(1) The forces on each bead (equations 1-4) are computed. This is the responsibility of the processor hosting each cube. In order to correctly calculate the pair-wise forces, each processor must have knowledge of the state of the beads inhabiting the relevant part of the halo of its adjacent cubes.

(2) For a current simulated time ***t***, the forces on each bead are used to derive a change in bead state (phase space coordinates) for time ***t+h***. (This requires no intercube communication).

(3) The bead state changes are applied; for the majority of beads that only move *within* their cube, this is trivial, but some beads (potentially those within the halo of a cube) may move to an adjacent cube; this requires inter-cube communication: the process representing a source cube will send a message to its (spatial) neighbour, bundling all the bead data for those beads that are moving *from* the source cell *to* the target cell. Thus the *ownership* of beads passes from cell to cell as the beads move about in space; the *volume of space* represented by a cell remains unchanged.

(4) Simulation time (***t***) is incremented (***t = t+h***), and the sequence repeated from step (1).

The initial configuration will not be in thermodynamic equilibrium, because the bead states were initialised randomly. It is necessary to allow the simulation to run for a few hundred timesteps to allow the thermostat (equations 3 and 4) to establish a uniform numerical temperature throughout the system. This can be monitored by calculating the system temperature at every timestep from equation (5) [21] and checking that it remains stable.

$$\mathrm{T}(t) = \left(k_B N_f\right)^{-1} \sum_i^N m v_i^2(t) \qquad (5)$$

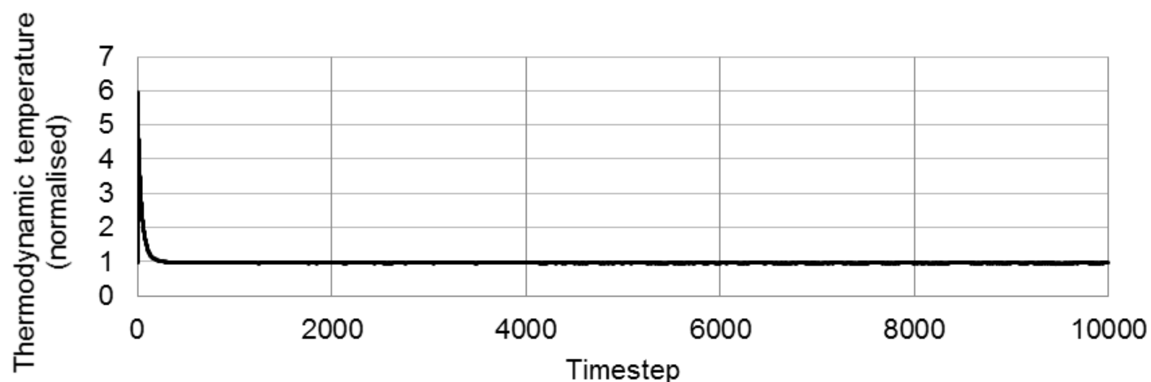
(5) On setup, a "stop time" is broadcast to all cells; when every cell reaches that simulations (irrespective of the wallclock time), the analysis terminates.

5 IMPLEMENTING DPD IN A POETS ENVIRONMENT

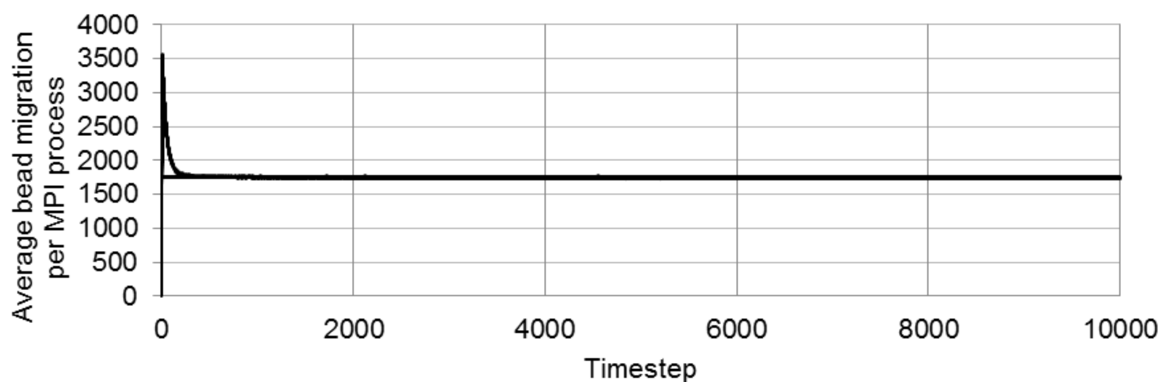
This algorithmic process is implemented in an event-driven graph-based environment in almost exactly the same way (Figure 8) - the differences lie in the numeric parameters of the simulation. Again with periodic boundary conditions, our application graph consists of a three-dimensional (triple) torus. The unit spatial cells correspond to the 'devices' of Figure 5, and each device is responsible for controlling the trajectories through phase space of the beads contained within the small element of space it represents, handing them off to adjacent devices as and when necessary, as in the conventional implementation. Sections 5.1-5.4 discuss differences of note that exist between the implementation techniques.

5.1 Data layout

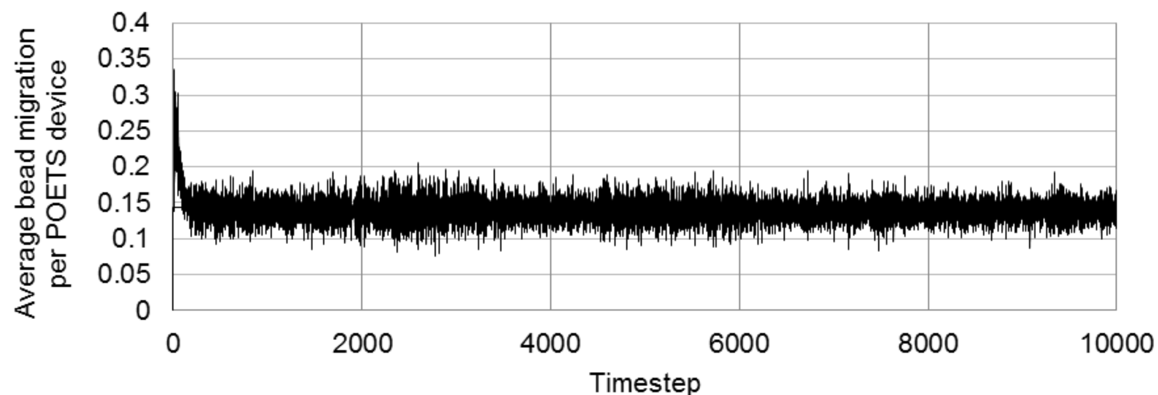
With POETS, the design intent is that we have available not a handful of processes, but tens of thousands of devices (ultimately millions - the system is designed to be scalable). The absolute number of beads necessary to perform sensible experiments is obviously a function of the SuS, but rather than have tens of thousands of beads in a cell (device) and transit thousands between cells at every timestep via a large MPI message, in POETS we have only two or three (and usually zero) beads/device, and these transit between devices where necessary in packets each containing only one bead. The cell size is chosen to be equal to $\boldsymbol{r_0}$ (equation 1) - there is no halo. Figure 10 shows the bead transit densities for a conventional MPI-based implementation (b) and a POETS system (c).



(a) Numerical thermodynamic temperature



(b) Average bead migration per MPI process (64 process system)



(c) Average bead migration per POETS device (1,000,000 device system)

Figure 10: Reasonable behavior: temperature and bead migration.

The DPD problem used in this paper is the phase separation of three immiscible fluids, simulated over a period of 10000 timesteps. Figure 10(a) shows the (simulated) temperature (equation 5) of the entire system - note that thermodynamic equilibrium is attained in a few hundred timesteps, corresponding to the onset of physically realistic

behaviour. (The configuration *before* equilibrium is also physically possible, but highly unlikely to occur in nature). Figure 10(b) shows the average number of beads moving between spatial cells at each timestep for a 64 process (8x8x8 cell) system. The value starts high (in the initial non-equilibrium phase) and rapidly decreases to an almost constant value. The curve is smooth because each point is the average of thousands of bead movements; each MPI cell has ~10^4 beads.

Figure 10(c) shows the average bead transit density for the equivalent POETS simulation: 10^6 (100x100x100) cells, each hosting 2-3 beads. The *absolute* numbers are very low and hence the average noisy.

5.2 Managing causality

Managing the passage of simulated time requires more care. On a platform where inter-process synchronization is easily available, it is relatively easy to keep the processes/threads in lockstep and allow simulated time to remain coherent across the compute fabric. However, POETS contains no intrinsic synchronization capabilities - event-based processing is intrinsically *reactive* - and simulated time must be handled explicitly alongside all the other variables of the simulation. An advantage of this is that there is no algorithmic requirement for simulation time to have the same value at the same wall-clock time over the compute fabric, as long as causality is not violated.

We move the simulation time 'wavefront' monotonically forward throughout the simulation compute fabric without violating causality with a globally-asynchronous-locally-synchronous approach (**GALS**) [22]. Each device contains a small finite state automaton, outlined in Figure 11. Broadly, the packet traffic consists of packets of four types; keep in mind that the entire system is asynchronous and non-transitive, so any packets may arrive at any device whilst it is in any state. At any time, a device may refuse to accept an arriving packet if it is currently busy, in which case the packet will remain buffered in a processor-local receive queue until it can be delivered. If the receive queue is full, it will be held in a network buffer en-route to the processor. As network buffers on a particular network route fill up, eventually the congestion reaches all the way back to potential senders. However, because send handlers will only be executed if there is capacity to transmit the packet one hop, we avoid deadlock by not trying to send messages until there network buffer space available. Pushing the non-sending behaviour decision back to the sender in this way allows the system to 'self-throttle' by delaying the *sender* state transitions.[4]

[4] Without this hardware capability, we would need either unbounded receive buffer space or the ability to interrogate the target buffer occupancy continuously, at a rate faster than the payload traffic can move. (Recall that here the 3D application problem is mapped to a 2D processor mesh.) In principle this can be achieved with a protocol like TCP by carefully controlling the receive window size, but the downside of this is a significantly reduced overall bandwidth.

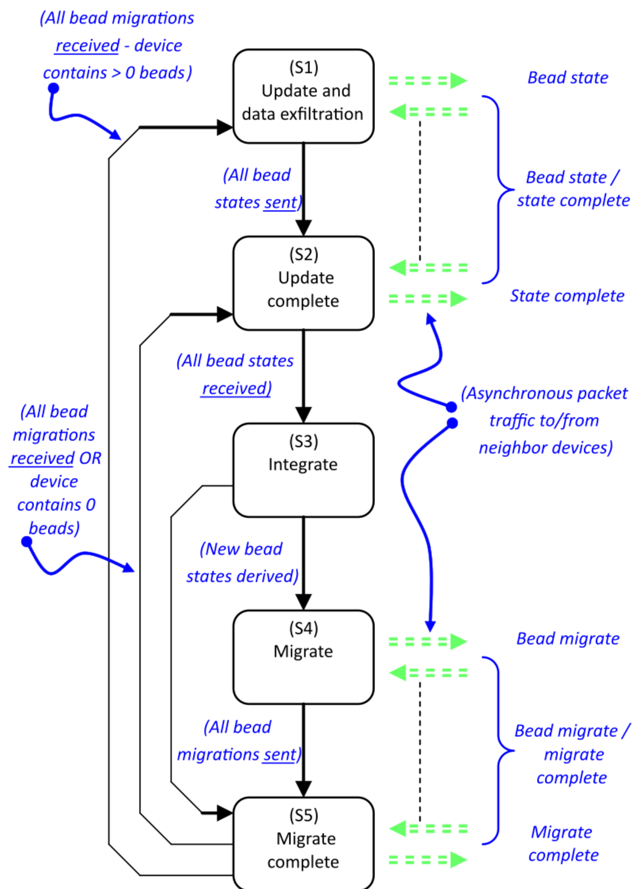


Figure 11: State automaton realizing a device.

The behaviour of this machine in outline is as follows:

(S1) Update state: the device sends the states of all its beads to all of its neighbours. (It may *receive* corresponding update assertions from its neighbours whilst this is going on). During this machine state, the device may exfiltrate the local data state to the relevant mothership (Figure 3). When all the bead states have been sent, the device moves to:

(S2) Update complete state: the device remains in this state until it has received all the bead states from all of its neighbours. It knows when this has happened because the *first* bead sent by every device in Update broadcasts how many beads are to come. The packets may or may not arrive in order, but when a device has received a 'bead count' packet and that number of bead state packets - in any order - it knows that it has everything. Once a device has received everything it is going to receive from all of its neighbours, it moves to:

(S3) Integrate state: Here the device updates the phase space states of each of the beads it contains, i.e. it integrates the equations of motion for the next timestep, $t+h$. The device increments its local time ($t = t+h$), and starts to migrate beads to other devices if necessary (i.e. if the new position of a bead takes it out of the spatial volume for which the device is responsible):

(S4) Migrate state: In a manner similar to the two update states, a device sends a 'bead to migrate' count along with the migrating bead state data. When a device has *sent* all its migration data, (it may concurrently be *receiving* migrate packets from its neighbours) it moves to:

(S5) Migrate complete state: the device spins until all the incoming migration packets have arrived, whereupon it emits its own 'migrate complete' packet to its neighbours, and returns to state **Update (S1)**.

One non-obvious comment is relevant: migrating a bead requires moving it from just one device to one other; however, in practice, all migrations are broadcast to all neighbours. The reason for this is that the overall objective of this entire architecture is to maximize speed; to send a migrating bead to the correct target requires (in terms of elapsed wall-clock):

A1: The identification of the correct target (the *sender* has to execute a 26-way switch to establish this).

A2: The bead is sent.

A3: The target handler is awoken.

All these three steps are temporally sequential. To send a migrating bead to all 26 neighbours and discard the unwanted ones requires:

B1: The beads are sent to all the neighbours (26 in parallel by hardware).

B2: The target handlers wake, and check that the incoming bead is relevant to them (26, in parallel).

Crudely, we are trading a 26-way switch - A1 (necessarily sequential) against 26 conditionals - B2, all executed in parallel.[5]

There is no synchronization barrier anywhere in the loop of Figure 11, and devices can compute bead movements as fast as they can, so the wall-clock time required to accomplish this will be a function of the number of beads hosted by the device, which has nothing to do with the passage of simulated time. Thus there is a danger of the simulation time wavefront becoming uneven over the compute fabric. The effective handshake between adjacent devices (S1/S2 - S3/S4 in Figure 11) ensures that the simulation time shear between *pairs* of devices is never greater than $\boldsymbol{h}$. This allows significant (data-driven statistical) time shear to develop over the platform as a whole, but without compromising causality or dropping data.

Without barriers, simulation shear is inevitable: the only way it can be avoided is if all the spatial cells (devices) behave identically all the time. Limiting the shear to ***1*h*** is a significant strength - it provides automatic fine grained localised load balancing in space and wallclock time.

5.3 Random number distribution

The random force component (equation 4) obviously requires a random number generator of some sort. It is a pairwise antisymmetric force, so it is asserted on *pairs* of beads. Each bead experiencing this force may be in a different (although adjacent) device, so it is necessary that we have identical sequences of random numbers in all *pairs* of adjacent devices, which naturally extends to the requirement that the RNGs in *every* device are in lockstep (with respect to simulation, not wall-clock time). This is achieved by implementing the RNG using a hash-based generator. Every bead ***i*** has a unique identifier ***id(i)***, which is unique amongst the beads in the SuS, with any unused LSBs filled with entropy. There is also a per-timestep seed ***S(t)***, generated independently in each thread, but guaranteed to be identical for each device at a given time point. (Even if this is not a coincident wall-clock time.) Thus the beads themselves carry a necessary component to synchronize the generators across the compute fabric with simulated time, which, of course, a global synchronization primitive would not be able to do. The final RNG function is

$$\mathrm{H}(i,j,t) = \left(id(i)^{S(t)} * id(j)\right) + \left(id(j)^{S(t)} * id(i)\right) \qquad (6)$$

Symmetric in ***i,j,*** the generator function passes SmallCrush (but not Crush or BigCrush) [23]. This has been tested in terms of temporal correlations (holding ***i,j*** constant and varying ***t***), spatial correlations (holding ***t*** constant and varying ***i,j***), and by considering the sequence of ***i,j,t*** values arising in a real DPD simulation.

5.4 Scalability

The scalability of the POETS approach is asymptotically going to be determined by some combination of the worst-case latency and throughput of the set of communicating pairs of cells in the physical hardware. In our hardware architecture

[5] Informal measurements taken during the development of POETS indicate that the multicast technique is approximately 30% faster, and the unicast method was abandoned early in the project.

the mailbox-mailbox (intra-board) communication is very low, of the order of 5ns per hop, with parallel links having a sustained bandwidth of 3.3 GB/s. Asymptotically the size of each FPGA is also constant, so we should not expect the bottleneck to lie within each board. The weak point is going to be the inter-board (i.e. inter-FPGA) links, as our main method of scaling is to add more boards to the mesh – currently we have a 2x3 board mesh in a 2x4 box mesh; the results presented in the next section support the claim for much larger meshes.

The measured inter-FPGA hop time is ~770 ns per hop, and in the current 8-box system the worst-case is for 12 hops to connect any two FPGAs, with a delay of ~9.2 us. However, with the cluster-based placement we use, this worst-case almost never happens – the vast majority of routes are intra-board, while very few involve more than 1 hop. It is noticeable that the 10 us worst-case delay is much less than the 10 ms per time-step that we see in practice, so given the GALS approach taken there appears to be a lot of head-room to scale. Clearly this scaling cannot be unlimited in the current system, but the current results suggest that scaling should continue up to around 256 boxes, as there would be a maximum 29 us delay versus the current 10 ms timestep – beyond this point it is difficult to extrapolate without actually building larger systems.

6 QUANTITATIVE BEHAVIOUR

This section describes the behaviour of the DPD algorithm, operating on the trivial physical system outlined previously, implemented on POETS.

6.1 Experimental setup

The test volume is a triple torus of devices (figures 7, 8), the *size* of which is a parameter in the results to be presented. Our test ensemble consists of three species of bead, all mutually repulsive, with an affinity matrix (equation 2) of

$$A = \begin{bmatrix} 25.00 & 75.00 & 35.00 \\ 75.00 & 25.00 & 50.00 \\ 35.00 & 50.00 & 25.00 \end{bmatrix} \qquad (7)$$

(Note that the larger the *affinity*, the greater is the conservative *repulsion* between the beads.) The experiment is simple: we initialize the system (beads are positioned randomly with a uniform distribution throughout the experimental volume, and the velocities are initialized with a Gaussian distribution centered on zero) and march forwards in time, integrating the equations of motion (as derived from equations 1-4) of the beads. Simulations are performed on bead ensembles ranging in size from 100 to 5000000; the relative abundance of the three species is 60:30:10 in every run.

6.2 Reasonable behaviour ?

We are integrating equations which contain a non-deterministic component (equation 4) on a message-passing hardware platform which itself has no global synchronization. The fundamental algorithmic basis of this is sound [7,10,11,24,25] but we need to ensure that the compute architecture is not introducing artefacts that may *appear* credible but which make the results non-physical.

Our experiments are designed to answer two questions:

- Is our system behaving reasonably, and generating emergent bead behaviour that is physically realistic and consistent with other analyses?
- How *fast* can it deliver this behaviour?

6.2.1 Consistency.

Performance is irrelevant if there is no trust in the accuracy and consistency of the results; we have five indications that POETS-DPD is generating credible information:

Indication 1: **Temperature**

Monitoring the bead ensemble temperature (as derived from equation 5 - Figure 10(a)) as a function of simulation time shows that

- The temperature asymptotes to 1 after a few hundred timesteps. It has 'forgotten' the non-equilibrium initial state and the thermostat has established a uniform temperature throughout the system. This asymptote is consistent with results from statistical analysis.
- It is consistent with analyses performed on other mature conventional simulation systems [35,36].

Indication 2: **Bead transit densities**

Figure 10 shows the density of beads transiting between devices at each timestep boundary, for the POETS implementation (Figure 10©) and a conventional MPI-based system (Figure 10(b)). The MPI devices hold thousands of beads, and so the mass cell-cell transit magnitude is a smooth line. Each POETS device, on the other hand, holds only two or three beads, so the corresponding device-device bead migration densities are low and noisy. The figure shows the number of beads *exported* from cells as a function of time, which is why the lines are not centered on zero; there is - averaged over time - an equal number of beads *imported*. Again, the stable asymptotes indicate that the system is in (numerical, at least) equilibrium.

Indication 3: **Radial distribution function**

The **radial distribution function** (RDF) is a measure of long-range order within a particle ensemble. It is the ratio between (the average number density of a particle type A at a distance ***r*** from any given bead of type B) and (the density at a distance ***r*** from a bead of type A in an ideal gas with the same overall density). A and B may be identical. It is an attractive metric to computational chemists because it can be relatively easily experimentally measured and used to calibrate computational models. The fine details are not relevant to this paper; the point is that the RDF will change in a predictable way with time as order emerges from the ensemble. Figure 12 shows the RDF as a function of timestep for two different species in the experimental setup. It is consistent with the results generated from [35,36] and hence indirectly with physical experiments.

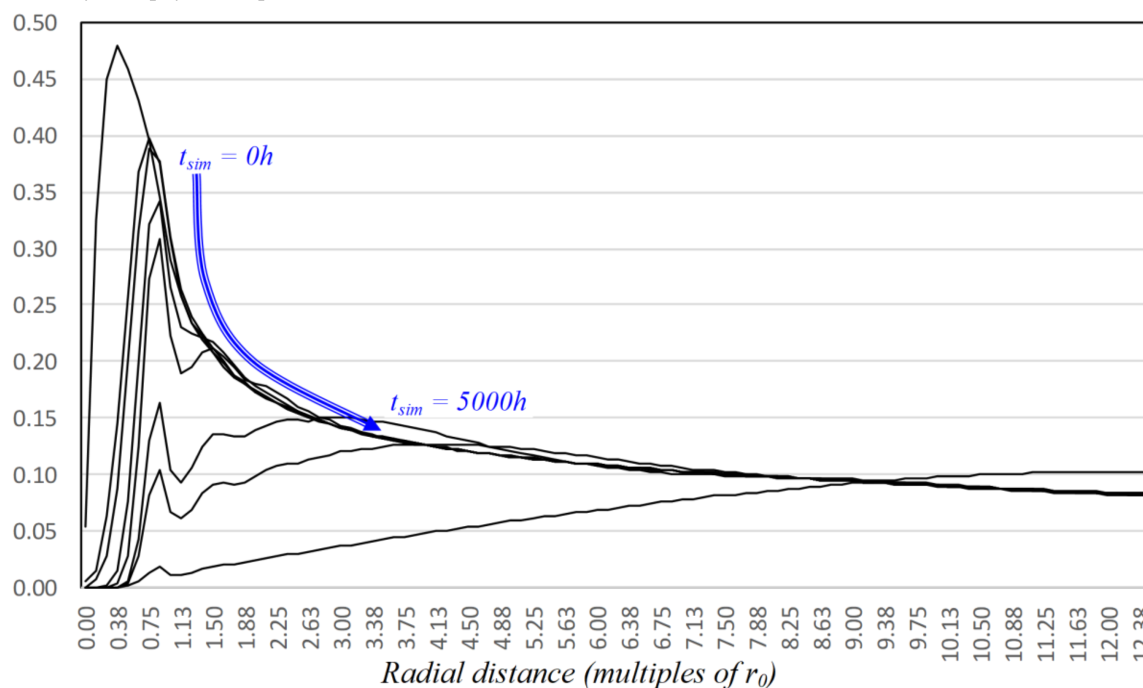


Figure 12: Radial distribution function changing in the course of the simulation.

Indication 4: **Bit level consistency**

We have a sequential software simulator which performs the same mathematical operations as the hardware POETS-DPD implementation, using the same types, expression orders, and hash-based random-number generation system. The software has been manually checked by inspection against the source code of the simulation system underpinning [35,36] - Osprey-DPD, verifying that they perform the same mathematical operations, albeit using different expressions, number types, and random number generators. We have verified this sequential software version against the concurrent version running in POETS-DPD, and verified that they initially produce exactly the same simulation trajectories.

After a certain number of time-steps (typically 100-200 steps) the simulation trajectories start to diverge, which is due to non-associative floating-point addition when accumulating forces – in POETS the messages will be received in a non-deterministic order, so the floating point force sums will be slightly different. To verify that this is the only source of differences, we implement a modified algorithm that uses slightly slower 22-bit fixed point to accumulate forces, making the accumulation associative. Flush-to-zero mode was enabled in software, to match the lack of floating-point de-normal support in hardware. Under these conditions we were able to get bit-exact matching results between our sequential software in x86 and our massively concurrent implementation with non-deterministic message ordering in POETS hardware.

Indication 5: **Emergent behaviour**

Finally, Figure 13 shows a 2D section of the 3D reaction volume at equilibrium. Three species of bead are present - only two are shown for clarity. The beads are all the same radius, but the image plane passes through them at different distances from their centroids. The emergent layering is horizontal by chance. The mixture of the three fluids has phase separated, leading to two homogenous phases of bead types (60%) and (30%) with the third type (10%) - whose conservative repulsion is smaller (equation 7) - coating the phase boundary. Fluid-fluid phase separation of this kind is an important phenomenon in polymer chemistry and biophysics, and this figure is entirely consistent with how physical systems are known to behave.

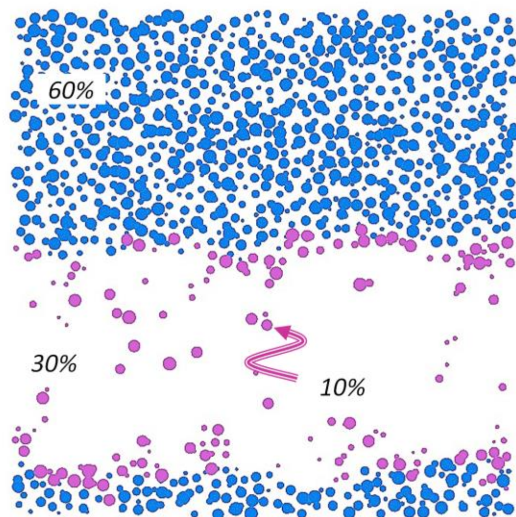


Figure 13: Emergent behavior: beads separating - spontaneously - into layers (30% "white" beads omitted for clarity)

6.3 Wall-clock performance

The focus of this work is on scaling and wall-clock speed. The results presented in this section (Figure 14 and Figure 15) all refer to the system parameterized in section 6.1. The figures show the elapsed wall-clock time as a function of bead count for a fixed number of simulation timesteps (here 10000).

6.3.1COTS platforms - wall-clock performance.

Figure 14 shows the behaviour of the experiment on a canonical single thread system (a baseline comparator), a conventional MPI platform, and a specialized implementation on a single **GPU** from a similar silicon generation to the POETS hardware [26,27]. The key point from Figure 14 is that whatever advantages accrue from parallelization, quite quickly these are overwhelmed by the cost of serialization and communication as the SuS gets bigger. We would expect a GPU implementation to be competitive for this application, because the DPD graph structure maps elegantly onto the underlying 3-D structure of the GPU. However, once an application moves away from this ideal match, a GPU implementation will become less competitive.

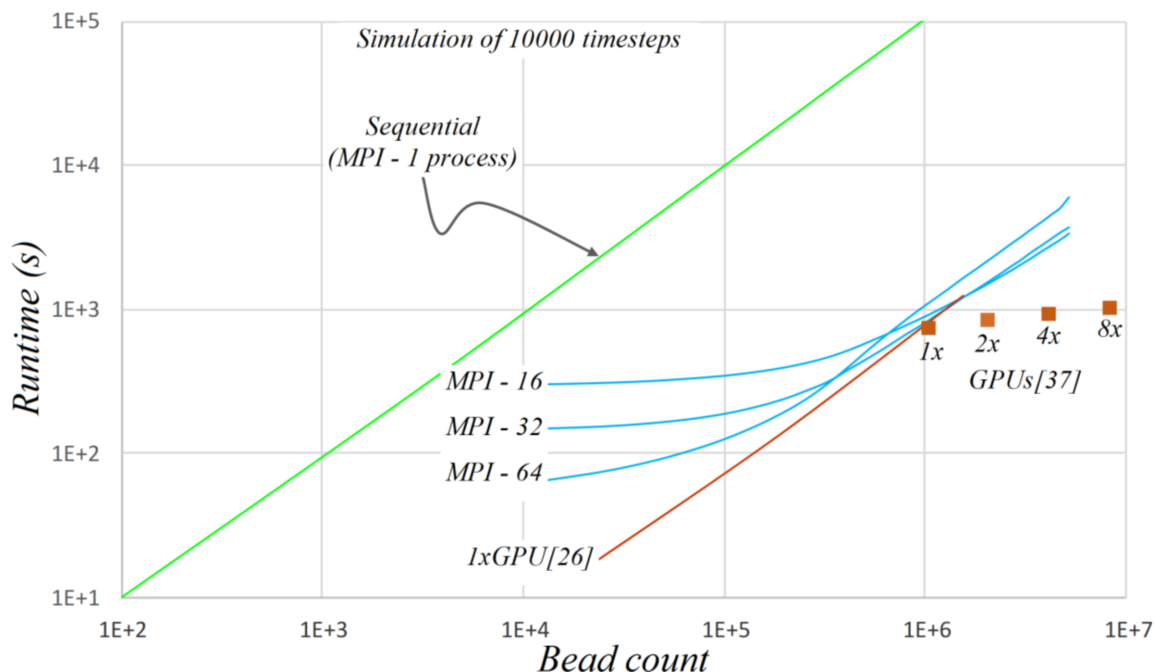


Figure 14: Runtime vs bead count on COTS architectures.

More recent work has focused on using multiple GPUs to handle larger volumes, using MPI to transport halos between GPUs. The four data-points shown in Figure 14 show weak scaling results from the Piz Daint supercomputer, showing performance results for 1,2,3 and 8 GPUs [37]. Note that these data-points were gathered with a lower density of 1.7 beads per unit volume, compared to 3 beads per unit volume for other results, so this comparison substantially favours the GPUs. We see that while the strong scaling is reasonable, there is still a noticeable upwards slope – every time we double the number of GPUs the performance of the system is less than double.

6.3.2POETS wall-clock performance.

Figure 15 shows the runtime on a number of POETS systems. (For comparison, the figure contains a copy of the data of Figure 14) The key points here are that when the number of devices is less than the number of physical threads available, the scaling is almost constant - *the amount of wall-clock required to perform the simulation is independent of the problem size* - the communication cost is tiny and massively parallelized.

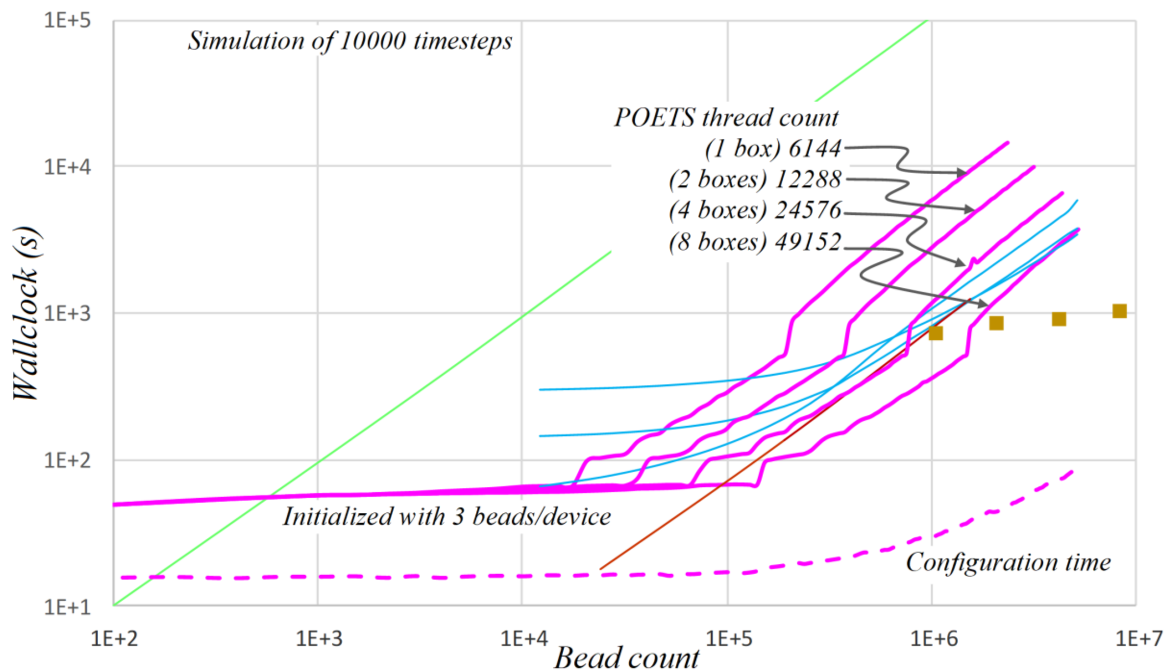


Figure 15: Runtime vs bead count on POETS - results superposed on the COTS results of Figure 14.

The shape of the timing curves arises from the nature of the underlying communication infrastructure, and may be understood by comparing with a conventional message-passing system. POETS provides a large number of compute units, embedded in a large communications mesh. It uses only small, fixed size packets to communicate, and the transmission of these is undertaken entirely by hardware. There is no low-level software layer involved. The consequence of this is that, at the lowest level, the message passing is truly parallel. In Figure 5, for example, to pass a packet between devices 11 and 03 can happen at *exactly* the same time as between 02 and 07 - there is no software bottleneck (although 02-07 will obviously be faster than 11-03, hence giving rise to the compute wavefront time-shear that the state machine of Figure 11 is there to control). As the network activity increases, packets will start to get in each others way at the mailbox level (the interface between hardware and the programmable component of the handlers) - hence the slight but finite slope of the line segments at the left of Figure 15. The inflection in the lines arises when the configuration system maps multiple devices to each thread and serializes the processing of the devices mapped to a particular thread. Doubling the size of the hardware doubles the abscissa value for which the serialization commences.

The (network unloaded) packet transit times in the compute hierarchy are shown in figure 1: to send a packet between devices mapped to different boards takes ~ 770 ns per inter-board hop; to go between devices mapped to different mailboxes on the same board takes ~ 5 ns per inter-mailbox hop; and the base cost for sending and receiving a packed is 230 ns. This inflection in packet hop times may be understood by looking at the low-level constituent actions of packet passing: broadly, for two *threads* to communicate requires behaviour supported by just one core, and so an element of cycle-based serialization creeps in. Communication between two cores is brokered by two cores, and so is marginally faster.

For completeness, the cost of *configuration* is also shown in Figure 15. Configuration has three components: application infiltration goes as the device (unit cell) count, the device mapping depends on the placement algorithm chosen, the cross compilation goes as the number of handler *types* (the design intent is that this will be very low, and parallel compilers are common), and the binary distribution costs depends on the granularity of the MPI universe (i.e. the number of *boards* - and hence motherships - in the system - see Figure 3).

6.4 Energy considerations

The POETS hardware described here is realized on a bank of FPGAs, and so will never be competitive against an ASIC solution in terms of power. For example, the event-based SpiNNaker machine [13] - which is ASIC based - achieves about 2.2 GIPS/W, which would have qualified it for a place in the Green500 supercomputer rankings when first commissioned, were it not that it is not a general-purpose machine. The FPGA design is not currently set up for energy proportional compute: for example, the inter-board links run at 100% traffic density all the time, sending dummy packets when there is no real traffic, to keep the link PLLs locked (saving loop stabilization startup time). Nevertheless, it is useful to look at the energy-compute relationships. Each compute FPGA dissipates around 50W, and ignoring the 'infrastructure' energy costs (motherships and associated interface circuitry in figure 1), figure 16 - right hand ordinate - shows the MIPS/W for the bead simulation. FLOPS and "non-FLOPS" are effectively separated out by the core. For this application, there are approximately 4x as many non-FLOPS as FLOPS, although they take the same time to execute and require approximately the same energy, so the curves (MIPS/W and MFLOPS/W) are almost identical in shape. The FPU utilization peaks at 97%, which is very high for such a fine-grain work-load. Usually this kind of efficiency would be associated with some sort of BLAS-style kernel. The instruction/CPU/cycle relationship also peaks at almost 1 (the latter two curves are not shown - they are the same shape as figure 16). For context, the POETS runtimes of Figure 15 are superimposed on the graph (left hand ordinate).

The figure shows the rate-limiting steps move with problem size: for small simulations there is not enough work to occupy the available hardware threads, so performance increases linearly with the problem size – each extra simulation cell gets assigned to previously idle resources, so the wall-clock time per simulation step remain the same even as more work is done within each simulation step. In the middle range (10^4-10^5 beads), processor utility tends to 1, and eventually the compute capacity of the system becomes a bottleneck. In the POETS hardware the zone between under-subscription and over-subscription of cores is smoothed out due the hyper-threaded nature of the underlying CPU core – each core becomes fully occupied at 8 active threads, and software scheduling is not needed until all 16 hardware threads are in use, providing a somewhat smoother peak in efficiency. The decline in MIPS/watt is due to the machine moving out of the intended design region as it becomes increasingly over-subscribed – the network becomes increasingly congested causing delays in send handlers, while the increasing number of simulation cells per thread causes more cache misses for each send and receive handler. Functionally the machine still works as intended, but the latency hiding and throughput optimisations become less effective due to the excessive over-subscription of threads.

Attempting to isolate 'compute (electrical) power' from 'infrastructure power' in COTS machines is extremely hard, but we can at least consider the TDP (Total Dissipated Power) of each system: the 8-box POETS system dissipates around 2400W; the MPI system dissipates around 500W; and a single Piz Daint GPU dissipates around 250W. However, it is difficult to directly compare these results, as the POETS system is constructed from FPGAs that are more than 10 years old, while the MPI and GPU systems use contemporary silicon and inherently provide more performance per watt and per silicon area. We are currently building a 2nd generation POETS system using Stratix 10 FPGAs, which will allow for a fairer comparison in terms of power and performance – however, the scaling results in the existing 48 FPGA system demonstrate that the overall algorithmic and hardware approach is impressive.

7 Other application domains

The range of application domains in which POETS can make a contribution is large, and can be divided loosely into two categories: regular and irregular discretisation. Fundamental to exploiting the power of POETS is that an application can be decomposed onto a very large number of simple interacting entities, which can be conveniently represented as a graph. (If this cannot be done, POETS is probably not the best technology for the application domain.)

- If the interaction graph is *dynamic* (i.e. the pairwise interactions of the entities changes during the course of the analysis and are not easily predicted *a priori*), then the most suitable formalism is to represent the *embedding space* as a (usually but not necessarily) regular graph, as in this paper.
  This methodology can be utilized in other domains: astrophysics, where the 'beads' are stars (or even galaxies); plasma simulation (the Culham Centre for Fusion Energy in the UK is a stakeholder in this work); high fidelity

semiconductor device simulation (the devices hold the substrate band structure, energy levels and fixed charges of the semiconductor, whilst mobile charge carriers are passed from device to device), and others.

- Examples of interaction graphs that are predominantly *static* (but not regular) include Petri nets and neural network models (both artificial and biological). These show [40,41,42] that the scaling relationships (c.f. Figure 15 here) are also effectively constants. This makes POETS an effective neuromorphic machine; we have stably simulated neural aggregates of over 10^6 neurons (devices) with 10^9 synapses (edges) using the canonical Izhikevitch differential neuron model [28]. Running on the 8-box system, we achieved 126M neuron steps/wall-clock sec with 2.1B spikes (packets) delivered/wall-clock sec. This, in turn, opens up fields in which ANNs are popular: machine learning (fintech, man/machine interaction - writing/speech/language interpretation), image analysis and anomaly detection.
  High voltage partial discharge (insulation breakdown) models involve using a derived electric field to trigger a microscopic discontinuous (hysteretic) impedance change, representing the spread of the discharge through the insulator; the spatial model of the system remains fixed, but here the *edges* of the device graph change state.
- (Particle-Particle)-(Particle-Mesh) - P3M [20] problems combine the strengths of both categories. Incorporation of long range fields (electrostatic, magnetic, gravitational) has the devices representing the fixed spatial tiling calculating the field [29], whilst other devices represent the movement of particle aggregates throughout the field: space tells particles how to move, particles tell space how to bend.

Specific areas in which we have usefully applied POETS (achieving near constant scaling) to date are

- Petri nets [40]
- Tsetlin machines [43,44]
- Graph analytics [8,45]
- Neuromorphic simulation [41,42]
- Machine learning [46]
- Genomic imputation [48]
- Earthquake modelling [work in progress]
- Magnetic field modelling [work in progress]
- Computational fluid dynamics [work in progress]
- Reactive analogue circuit simulation [work in progress]

An overview may be found in [47].

8 FINAL COMMENTS

POETS is a massively parallel system capable of delivering correspondingly massive processing speedups. It is not a general purpose compute environment; problems have to be capable of being rendered in the form of a graph, and the solution appears as an emergent property of a packet-brokered relaxation process. In this paper, we have described how an industrially relevant problem may be addressed with the POETS architecture. The introduction and background was lengthy, but this is indicative of the problems faced in porting a mature and subtle algorithm to a new and different compute infrastructure: both must be described and the descriptions reconciled.

This notwithstanding, for problems that can be cast in a manner amenable to POETS solution, the speedups are dramatic: Figure 15 shows a speed advantage over a baseline sequential implementation of well over x100; this can be improved almost arbitrarily by adding more hardware - the scaling curve is simply pushed to the right. (Optimistic estimates[6] suggest that for DPD a *further* three orders of magnitude should be possible before the system becomes communication bound all the time.)

POETS is about making industrially important computation problems resolve in hours, as opposed to weeks, on relatively cheap hardware. Aside from moving the scope of simulation problems to much larger domains, there is a

[6] Tuning MPI-DPD suggests that the code is at its most efficient (i.e. each process spends most of its time doing its own independent calculations and not waiting for messages) when each process owns a volume of space about 20-30$\boldsymbol{r_0}$ on a side. POETS has O(1) beads/device, which suggests that a further O(20^3) speedup should be feasible.

knock-on effect on budgets: no longer do massive compute resources need to be considered as budgetary line items. This, in turn, has an effect on the way humans explore a problem: if justifying the compute budget *a priori* is no longer a necessary burden, the users (here, biochemists) are free to explore avenues of greater risk/reward ratio than they could previously afford. Hitherto prohibitively expensive parameter space exploration becomes feasible.

For graph based applications that effectively implement a *relaxation* process of some sort - as seen in the inner loop of most simulation problems - the idea that compute resource is freely available (almost infinite) means that we can freely waste it: event handlers may operate on stale and/or duplicate data. If we view the simulation as a sequence of state space transitions (where only the final converged state corresponds to physical reality), then any movement of any state subset towards the solution will speed up the overall process. (The only algorithmic difference between conventional Gauss-Seidel and Jacobi is the 'freshness' of the update vector.)

Unlike conventional compute, event-based system designers do not design or control the solution *trajectory* (and therefore do not pay for it), they just control the point at which the solution emerges. This is a novel programming paradigm of immense potential: event based processing is much more "natural" than control-flow processing, and it offers solution scaling times that make the technology extremely powerful. We are only beginning to learn how it may be exploited.

ACKNOWLEDGEMENTS:

This work was supported by EPSRC Grant EP/N031768/1 (POETS). (J.C. Shillcock was supported by funding to the BlueBrain project, a research centre of the Ecole Polytechnique Federale de Lausanne (EPFL) from the Swiss Governments ETH Board of the Swiss Federal Institutes of Technology). The corresponding author is A.D. Brown (adb@ecs.soton.ac.uk). The codebase is available at `github.com/Osprey-DPD/osprey-dpd`, `github.com/POETSII/dpd-baremetal` and `github.com/POETSII/tinsel`.

References

<bib id="bib1"><number>[1]</number>Michael E. Glazier and Anthony P. Ambler "Ultimate: A hardware logic simulation engine" Proc 21st DA Conference 1984, pp. 336-342.</bib>
<bib id="bib2"><number>[2] Edson</number>P. Ferlin, Heitor S. Lopes, Carlos R. Erig Lima and Maurício Perretto "Prada: a high-performance reconfigurable parallel architecture based on the dataflow model" International Journal of High Performance Systems Architecture, vol. 3, no. 1, pp. 41-55, 2011.</bib>
<bib id="bib3"><number>[3] Brian</number>W. Hollocks "Forty years of discrete event simulation" Journal of the Operational Research Society, vol. 57, no. 12, pp. 1383-1399.</bib>
<bib id="bib4"><number>[4] Andrew</number>F. Carpenter "An engine for the multi-level simulation of digital systems," Ph.D. dissertation, University of Manchester, 1985.</bib>
<bib id="bib5"><number>[5] Richard</number>Membarth, Frank Hannig, Jürgen Teich and Harald Köstler "Towards domain-specific computing for stencil codes in HPC" SC Companion: High Performance Computing, Networking Storage and Analysis. IEEE, 2012, pp. 1133-1138.</bib>
<bib id="bib6"><number>[6] Prashant</number>Singh Rawat, Miheer Vaidya, Aravind Sukumaran-Rajam, Mahesh Ravishankar, Vinod Grover, Atanas Rountev, Louis-Noël Pouchet and Ponnuswamy Sadayappan "Domain-specific optimization and generation of high performance GPU code for stencil computations" Proc IEEE, vol 106, no 11, pp 1902-1920.</bib>
<bib id="bib7"><number>[7] Robert</number>D. Groot and Patrick B. Warren "Dissipative particle dynamics: Bridging the gap between atomistic and mesoscopic simulation" Journal of chemical physics, vol. 107, no. 11, pp. 4423-4435, 1997.</bib>
<bib id="bib8"><number>[8] Matthew</number>F. Naylor, Simon W. Moore and David B. Thomas "Tinsel: a manythread overlay for FPGA clusters" 29th International Conference on Field Programmable Logic and Applications. IEEE, 2019, pp. 375-383.</bib>
<bib id="bib9"><number>[9] Andrew</number>D. Brown, Mark L. Vousden, Alexander D. Rast, Graeme M. Bragg, David B. Thomas, Jonny R. Beaumont, Matthew F. Naylor and Andrey M. Mokhov "POETS: Distributed event-based computing-scaling behaviour" Proc ParCo'19, Prague.</bib>
<bib id="bib10"><number>[10] P.J.</number>Hoogerbrugge and Johannes M. Koelman "Simulating microscopic hydrodynamic phenomena with dissipative particle dynamics" Europhysics Letters, vol. 19, no. 3, pp. 155-160, Jun 1992.</bib>
<bib id="bib11"><number>[11] Pep</number>Espanol and Patrick Warren, "Statistical mechanics of dissipative particle dynamics" Europhysics Letters, vol. 30, no. 4, p. 191, 1995.</bib>
<bib id="bib12"><number>[12] Andrew</number>D. Brown "Event-driven computing", keynote address, ASAP'16, Imperial College London, 2016</bib>
<bib id="bib13"><number>[13] Steve</number>B. Furber, David R. Lester, Luis A. Plana, Jim D. Garside, Eustace Painkras, Steve Temple and Andrew D. Brown "Overview of the spinnaker system architecture" IEEE T Computers, vol. 62, no. 12, pp. 2454-2467, 2012.</bib>
<bib id="bib14"><number>[14] Andrew</number>D. Brown, Steve B. Furber, Jeffrey S. Reeve, Jim D. Garside, Kier J. Dugan, Luis A. Plana and Steve Temple "SpiNNaker - programming model" IEEE T Computers, 64, no 6, June 2014, pp 1769-1782 ISSN: 0018-9340 doi:10.1109/TC.2014.2329686.</bib>
<bib id="bib15"><number>[15] Andrew</number>D. Brown, Rob Mills, Kier J. Dugan, Jeff S. Reeve and Steve B. Furber "Reliable computation with unreliable computers", IEE Computers and Digital Techniques, doi: 10.1049/iet-cdt.2014.0110</bib>
<bib id="bib16"><number>[16] Saroj</number>A. Shambharker "A study on setting processor or CPU affinity in multi-core architectures for parallel computing", Int J Science & Research, ISSN 2319-7064 pp 1987-1990, vol. 4, no 5, 2015</bib>

<bib id="bib17"><number>[17] Brian</number>W. Kernighan and Shen Lin "An efficient heuristic procedure for partitioning graphs" Bell system technical journal, vol. 49, no. 2, pp. 291-307, 1970.</bib>
<bib id="bib18"><number>[18] Benedict</number>R. Gaster, Lee Howes, David R. Kaeli, Perhaad Mistry and Dana Schaa "Heterogeneous computing with OpenCL" Morgan Kaufmann 2012 ISBN 978-0-12-387766-6</bib>
<bib id="bib19"><number>[19] Pep</number>Espanol and Patrick B. Warren, "Statistical mechanics of dissipative particle dynamics", Europhys Letters, 30, no 4, 1995</bib>
<bib id="bib20"><number>[20] Roger</number>W. Hockney and James W. Eastwood "Computer simulation using particles" IOP publishing 1988, ISBN-10 0-85274-392-0</bib>
<bib id="bib21"><number>[21] Daan</number>Frenkel and Berend Smit "Understanding molecular simulation" Academic Press, 2002, ISBN-13 978-0-12-267351-1</bib>
<bib id="bib22"><number>[22] Milos</number>Krstic, Eckhard Grass, Frank K. Gürkaynak and Pascal Vivet "Globally Asynchronous, Locally Synchronous circuits: overview and out-look" IEEE Design & Test of Computers, vol. 24, no. 5, pp. 430-441, Sept-Oct. 2007, doi: 10.1109/MDT.2007.164.</bib>
<bib id="bib23"><number>[23] Pierre</number>L'Ecuyer and Richard Simard: "TestU01: a software library in ANSI C for empirical testing of random number generators" ACM Trans Mathematical Software, vol. 33, no 22, 2007.</bib>
<bib id="bib24"><number>[24] Frank</number>Jensen, "Introduction to computational chemistry". Wiley 2017.</bib>
<bib id="bib25"><number>[25] Nicos</number>S. Martys and Raymond D. Mountain "Velocity Verlet algorithm for dissipative-particle-dynamics-based models of suspensions" Physical Review E, vol. 59, no. 3, p. 3733, 1999.</bib>
<bib id="bib26"><number>[26] Hao</number>Wu, Junbo Xu, Shengfei Zhang and Hao Wen "GPU accelerated dissipative particle dynamics with parallel cell-list updating" IEIT J Adaptive and dynamic computing, vol 1, no 2, pp 33-42, 2011</bib>
<bib id="bib27"><number>[27] Keda</number>Yang, Zhiqiang Bai, Jiaye Su and Hongxia Guo "Efficient and large-scale dissipative particle dynamics simulations on GPU" Soft Materials, vol. 12, no. 2, pp. 185-196, 2014.</bib>
<bib id="bib28"><number>[28] Eugene</number>M. Izhikevitch and Gerald M. Edelman "Large-scale model of mammalian thalamocortical systems", Proc Nat Acad Sci USA vol. 105, no. 9 pp 3593-3598, 2008</bib>
<bib id="bib29"><number>[29] Robert</number>D. Groot "Electrostatic interactions in dissipative particle dynamics - simulation of polyelectrolytes and anionic surfactants", J. Chem. Phys 118-11265 (2003) doi 10.1063/1.1574800</bib>
<bib id="bib30"><number>[30] Sriram</number>Pemmaraju and Steven Skiena "Computation discrete mathematics" ISBN-13 978-0521806862</bib>
<bib id="bib31"><number>[31] Henri</number>Vogt "Lecons sur le resolution algebraique equations", 1895, republished as ISBN-13 978-1148316321</bib>
<bib id="bib32"><number>[32] Matthew</number>Sacker, Andrew D. Brown, Andy J. Rushton and Peter R. Wilson, "A behavioural synthesis system for asynchronous circuits", IEEE Transactions on VLSI, vol 12 no 9 2004 pp 978-994, doi 10.1109/TVLSI.2004.832944</bib>
<bib id="bib33"><number>[33] Daniel</number>J.D. Milton, Andrew D. Brown, Mark Zwolinski and Peter R. Wilson, "Behavioural synthesis utilising dynamic memory constructs", IEE Computers & Digital Techniques vol 151 no 3 2004 pp 252-264 doi 10.1049/ip-cdt:20040241</bib>
<bib id="bib34"><number>[34] Andrew</number>D. Brown, Daniel J.D. Milton, Andy J. Rushton and Peter R. Wilson, "Behavioural synthesis utilising recursive definitions", IET Computers & Digital Techniques 2012 pp 1-8 doi: 10.1049/iet-cdt.2012.0006</bib>
<bib id="bib35"><number>[35] Julian</number>C. Shillcock and Reinhard Lipowsky, "Tension-induced fusion of bilayer membranes and vesicles", Nature Materials 4(3) pp 225-228 Mar 2005</bib>
<bib id="bib36"><number>[36] Julian</number>C. Shillcock and Reinhard Lipowsky, "Equilibrium structure and lateral stress distribution of amphiphilic bilayers from dissipative particle dynamics simulations", J. Chem. Phys 117(10) pp 5048-5061 Sept 2002</bib>
<bib id="bib37"><number>[37] J.</number>Castagna, X.Gui, M. Seaton, A. O'Cais "Towards extreme scale dissipative particle dynamics simulations using multiple GPGPUs", Computer Physics Communications, doi 10:1016/j.cpc.2020.107159</bib>
<bib id="bib38"><number>[38] Matthew</number>F. Naylor, Simon W. Moore, David B. Thomas, Jonathan R. Beaumont, Shane T. Fleming, Mark L. Vousden, Theo Markettos, Thomas Bytheway and Andrew D. Brown, "General hardware multicasting for fine-grained message-passing architectures", Proc 29th Euromicro International Conference on Parallel, Distributed and Network-based Processing, PDP 2021, 126-133, https://doi.org/10.1109/PDP52278.2021.00028</bib>
<bib id="bib39"><number>[39] Julian</number>C. Shillcock, David B. Thomas, Jonathan R. Beaumont, Mark L. Vousden and Andrew D. Brown, "Coupling bulk phase separation of disordered proteins to membrane domain formation in molecular simulations on a bespoke computer fabric", Membranes, 12(1) :17 https://doi.org/10.3390/membranes12010017</bib>
<bib id="bib40"><number>[40] Ashur</number>Rafiev, Jordan Morris, Fei Xia, Alex Yakovlev, Matthew Naylor, Simon Moore, David Thomas, Graeme Bragg, Mark Vousden and Andrew Brown, "Practical distributed implementations of very large scale Petri net simulations": in ToPNoC XVI, LNCS 13220, pp112-139, 2022 https://doi.org/10.1007/978-3-662-65303-6_6.</bib>
<bib id="bib41"><number>[41] Alexander</number>Rast, Mahyar Shahsavari, Graeme M. Bragg, Mark Vousden, David Thomas and Andrew Brown. "A Hardware/Application Overlay Model for Large-Scale Neuromorphic Simulation". 2020 International Joint Conference on Neural Networks (IJCNN). 19-24 July 2020. DOI: 10.1109/IJCNN48605.2020.9206708</bib>
<bib id="bib42"><number>[42] Mahyar</number>Shahsavari, Jonny Beaumont, David Thomas and Andrew Brown, "POETS: A parallel cluster architecture for Spiking Neural Network" Jul 2021, International Journal of Machine Learning and Computing. 11, 4, p. 281-285</bib>
<bib id="bib43"><number>[43] Ashur</number>Rafiev, Jordan Morris, Fei Xia, Rishad Shafik, Alex Yakovlev, Ole-Christoffer Granmo, and Andrew Brown, "Visualization of Machine Learning Dynamics in Tsetlin Machines" in International Symposium on the Tsetlin Machine (ISTM), IEEE, 2022</bib>
<bib id="bib44"><number>[44] Jordan</number>Morris, Ashur Rafiev, Fei Xia, Rishad Shafik, Alex Yakovlev and Andrew Brown, "An Alternate Feedback Mechanism for Tsetlin Machines on Parallel Architectures" in International Symposium on the Tsetlin Machine (ISTM), IEEE, 2022</bib>
<bib id="bib45"><number>[45] Ashur</number>Rafiev, Alex Yakovlev, Ghaith Tarawneh, Matthew Naylor, Simon Moore, David Thomas, Graeme Bragg, Mark Vousden, and Andrew Brown. "Synchronization in Graph Analysis Algorithms on the POETS Many-core Architecture". IET Computers & Digital Techniques **16** pp 71-88 2022. DOI: 10.1049/cdt2.12041</bib>
<bib id="bib46"><number>[46] Mahyar</number>Shahsavari, David Thomas, Andrew Brown and Wayne Luk "Neuromorphic Design Using Reward-based STDP Learning on Event-Based Reconfigurable Cluster Architecture" ICONS 2021: International Conference on Neuromorphic Systems July 2021 pp 1–8 doi 10.1145/3477145.3477151</bib>

<bib id="bib47"><number>[47] Mark</number>L. Vousden, Jordan Morris, Graeme M. Bragg, Jonathan R. Beaumont, Ashur Rafiev, Wayne Luk, David B. Thomas, and Andrew D. Brown. "Event-Based High Throughput Computing". IET Computers and Digital Techniques. (Awaiting referees comments.)</bib>
<bib id="bib48"><number>[48] Jordan</number>Morris, Ashur Rafiev, Graeme Bragg, Mark Vousden, David Thomas, Alex Yakovlev and Andrew Brown "An event driven approach to genotype imputation on a RISC-V FPGA cluster" IEEE Transactions on Computational Biology and Bioinformatics 2022. (Awaiting referees comments)</bib>
<bib id="bib49"><number>[49] Matthew</number>F. Naylor, Simon W. Moore, David B. Thomas, Jonny R. Beaumont, Shane Fleming, Mark L. Vousden, Theo Markettos, Tom Bytheway and Andrew Brown "General hardware multicasting for fine-grained message-passing architectures" Mar 2021, Proceedings - 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2021. p. 126-133 doi 10.1109/PDP52278.2021.00028</bib>
<bib id="bib50"><number>[50] Matthew</number>Naylor, Simon Moore, Andrey Mokhov, David Thomas, Jonny Beaumont, Shane Fleming, Theo Markettos, Tom Bytheway and Andrew Brown "Termination detection for fine-grained message-passing architectures" Proc ASAP 2020. Hannig, F., Navaridas, J., Koch, D. & Abdelhadi, A. (eds.). IEEE, pp 17-24 8 p. 9153260</bib>
<bib id="bib51"><number>[51] George</number>Karypis and Vipin Kumar "A fast and high quality multi-level scheme for partitioning irregular graphs", SIAM J on Scientific Computing, vol 30 no 1 pp 359-392 1999</bib>
<bib id="bib52"><number>[52] Mark</number>Vousden, Graeme McLachlan Bragg, and Andrew Brown. "Asynchronous Simulated Annealing on the Placement Problem: A Beneficial Race Condition". Journal of Parallel and Distributed Computing. Vol 169 pp 242-251 2022</bib>

\begin{CCSXML}

<ccs2012>
<concept>
<concept_id>10010583.10010786.10010787</concept_id>
<concept_desc>Hardware~Analysis and design of emerging devices and systems</concept_desc>
<concept_significance>500</concept_significance>
</concept>
<concept>
<concept_id>10010520.10010521.10010528.10010536</concept_id>
<concept_desc>Computer systems organization~Multicore architectures</concept_desc>
<concept_significance>500</concept_significance>
</concept>
<concept>
<concept_id>10010147.10010169.10010170</concept_id>
<concept_desc>Computing methodologies~Parallel algorithms</concept_desc>
<concept_significance>500</concept_significance>
</concept>
<concept>
<concept_id>10010147.10010169.10010170.10003824</concept_id>
<concept_desc>Computing methodologies~Self-organization</concept_desc>
<concept_significance>500</concept_significance>
</concept>
</ccs2012>
\end{CCSXML}
\ccsdesc[500]{Hardware~Analysis and design of emerging devices and systems}
\ccsdesc[500]{Computer systems organization~Multicore architectures}
\ccsdesc[500]{Computing methodologies~Parallel algorithms}
\ccsdesc[500]{Computing methodologies~Self-organization}