



POETS: An Event-driven Approach to Dissipative Particle Dynamics

ANDREW D. BROWN, Department of Electronics and Computer Science, University of Southampton, UK

JONATHAN R. BEAUMONT, Department of Electronic Engineering, Imperial College London, South Kensington Campus, London, UK

DAVID B. THOMAS, Department of Electronics and Computer Science, University of Southampton, UK

JULIAN C. SHILLCOCK, Blue Brain Project, Ecole Polytechnique Federale Lausanne, Lausanne, Switzerland

MATTHEW F. NAYLOR, Computer Laboratory, University of Cambridge, UK

GRAEME M. BRAGG and **MARK L. VOUSDEN**, Department of Electronics and Computer Science, University of Southampton, UK

SIMON W. MOORE, Computer Laboratory, University of Cambridge, UK

SHANE T. FLEMING, Department of Computer Science, University of Swansea, UK

7

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. Partially Ordered Event Triggered System (POETS)—the focus of this article—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 article, 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 that 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

This work was supported by EPSRC Grant EP/N031768/1 (POETS). (J. C. Shillcock was supported by funding to the Blue-Brain 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).

Authors’ addresses: A. D. Brown (corresponding author), D. B. Thomas, G. M. Bragg, and M. L. Vousden, Department of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK; emails: adb@ecs.soton.ac.uk, d.b.thomas@southampton.ac.uk, gmb@ecs.soton.ac.uk, m.vousden@soton.ac.uk; J. R. Beaumont, Department of Electronic Engineering, Imperial College London, South Kensington Campus, London SW7 2AZ, UK; email: j.beaumont@imperial.ac.uk; J. C. Shillcock, Blue Brain Project, Ecole Polytechnique Federale Lausanne, 2149 Batiment A1, Station 19, CH-1015 Lausanne, Switzerland; email: julian.shillcock@epfl.ch; M. F. Naylor and S. W. Moore, Computer Laboratory, University of Cambridge, CB3 0FD, UK; emails: {matthew.naylor, s.w.moore}@cl.cam.ac.uk; S. T. Fleming, Department of Computer Science, University of Swansea, UK; email: s.t.fleming@swansea.ac.uk.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions.acm.org).

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

2329-4949/2023/06-ART7 \$15.00

<https://doi.org/10.1145/3580372>

thread baseline comparator and scaling behaviour that is almost constant. The results are validated against a “conventional” implementation.

CCS Concepts: • **Hardware** → **Analysis and design of emerging devices and systems** • **Computer systems organization** → **Multicore architectures** • **Computing methodologies** → **Parallel algorithms; Self-organization**;

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

ACM Reference format:

Andrew D. Brown, Jonathan R. Beaumont, David B. Thomas, Julian C. Shillcock, Matthew F. Naylor, Graeme M. Bragg, Mark L. Vousden, Simon W. Moore, and Shane T. Fleming. 2023. POETS: An Event-driven Approach to Dissipative Particle Dynamics. *ACM Trans. Parallel Comput.* 10, 2, Article 7 (June 2023), 32 pages.

<https://doi.org/10.1145/3580372>

1 INTRODUCTION

High-performance computers get ever more powerful, year after 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 article, 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 article describes a computing architecture designed to address not a particular *domain* but a large, though specific, compute *pattern*. **Partially Ordered Event Triggered System (POETS)** is an *event-based* computing architecture. Like **Message Passing Interface (MPI)** programs, multiple processes are executed simultaneously (in POETS they are called threads) communicating by messages (in POETS they are called packets). Unlike MPI, POETS packets are small (64 bytes), fixed size, hardware brokered, and fast due to the absence of a software infrastructure, and thus the overhead of message choreography is avoided. Similarly 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 dataflow 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 dataflow 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 the phase¹ space of (at the very minimum) hundreds of thousands of particles, and the corresponding computing costs are massive.

In this article, we present the following:

- 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 computation-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 twofold: absolute speed and near constant scaling (see Figure 15).

In more detail, the article consists of eight 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 10,000 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 include neural simulations (massively non-isotropic application graphs with device degrees from 1 to $\sim 10^5$) [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

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

as a sheet-anchor for this article, we tile three-dimensional (3D) 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 synchronisation of the handlers—the final solution is distributed throughout the local memories of all the devices and may be exfiltrated for conventional visualisation.

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, then programs will perform well. If the software architect strays from this design intent, then 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

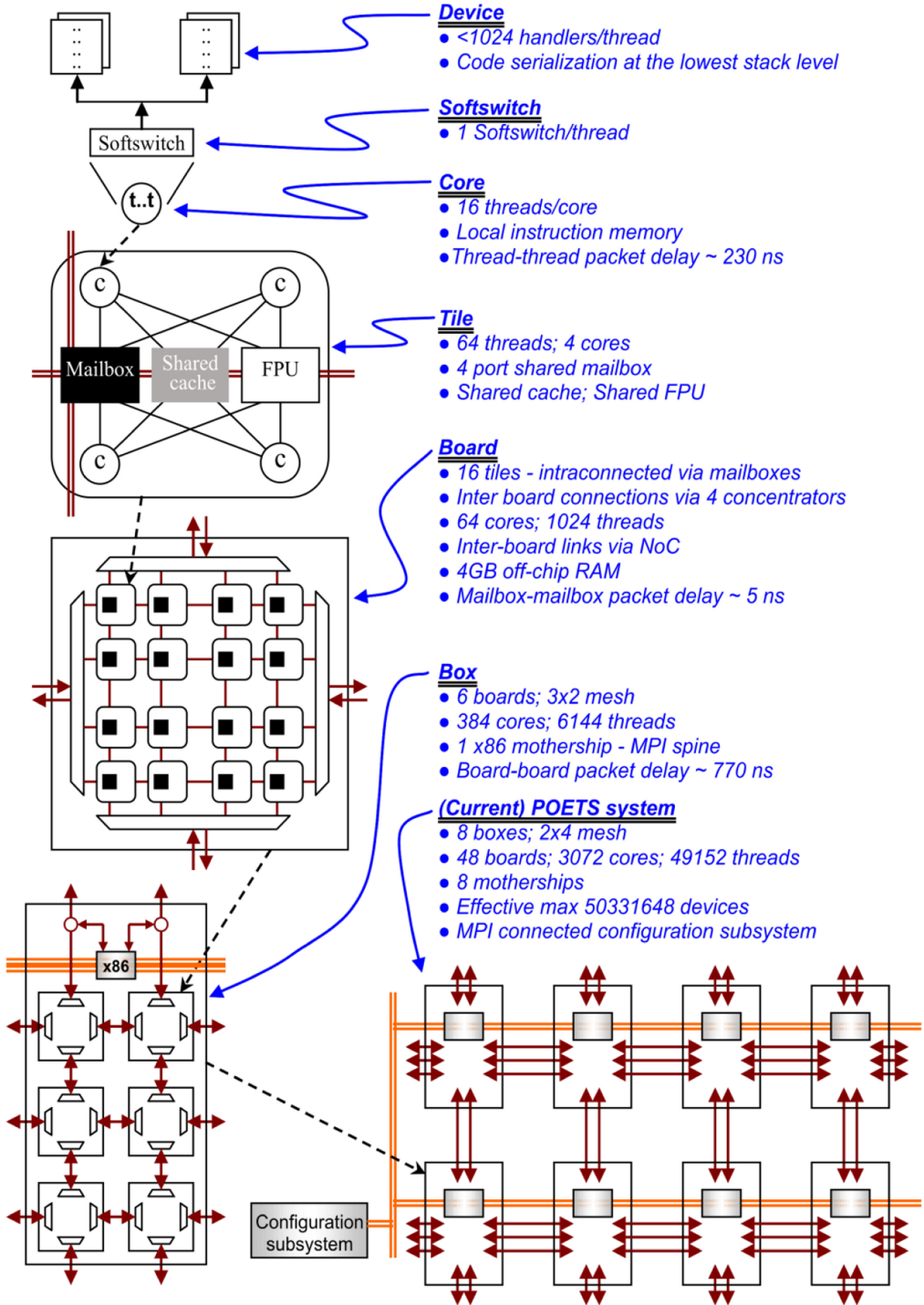


Fig. 1. The hierarchy of the POETS hardware from device handler to complete (current) system.

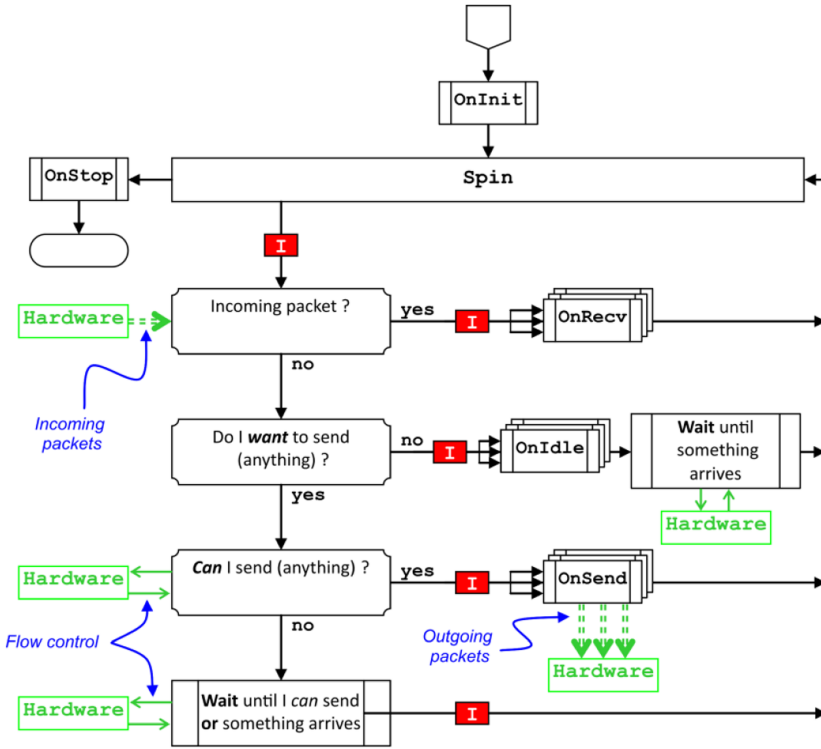


Fig. 2. Abstract SoftSwitch (a spinning device multiplexer) model.

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 ($<1,024$) 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.

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 4 Gbytes (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 4×4 grid. Each board has four 10 Gbps links that 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 Reference [38].

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 2×4 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 serialisation provided by the softswitches allows application graphs of tens of millions of vertices—much larger systems are possible, but the serialisation 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, serialisation 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, then serialisation 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. Serialisation 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.

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

²The 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.

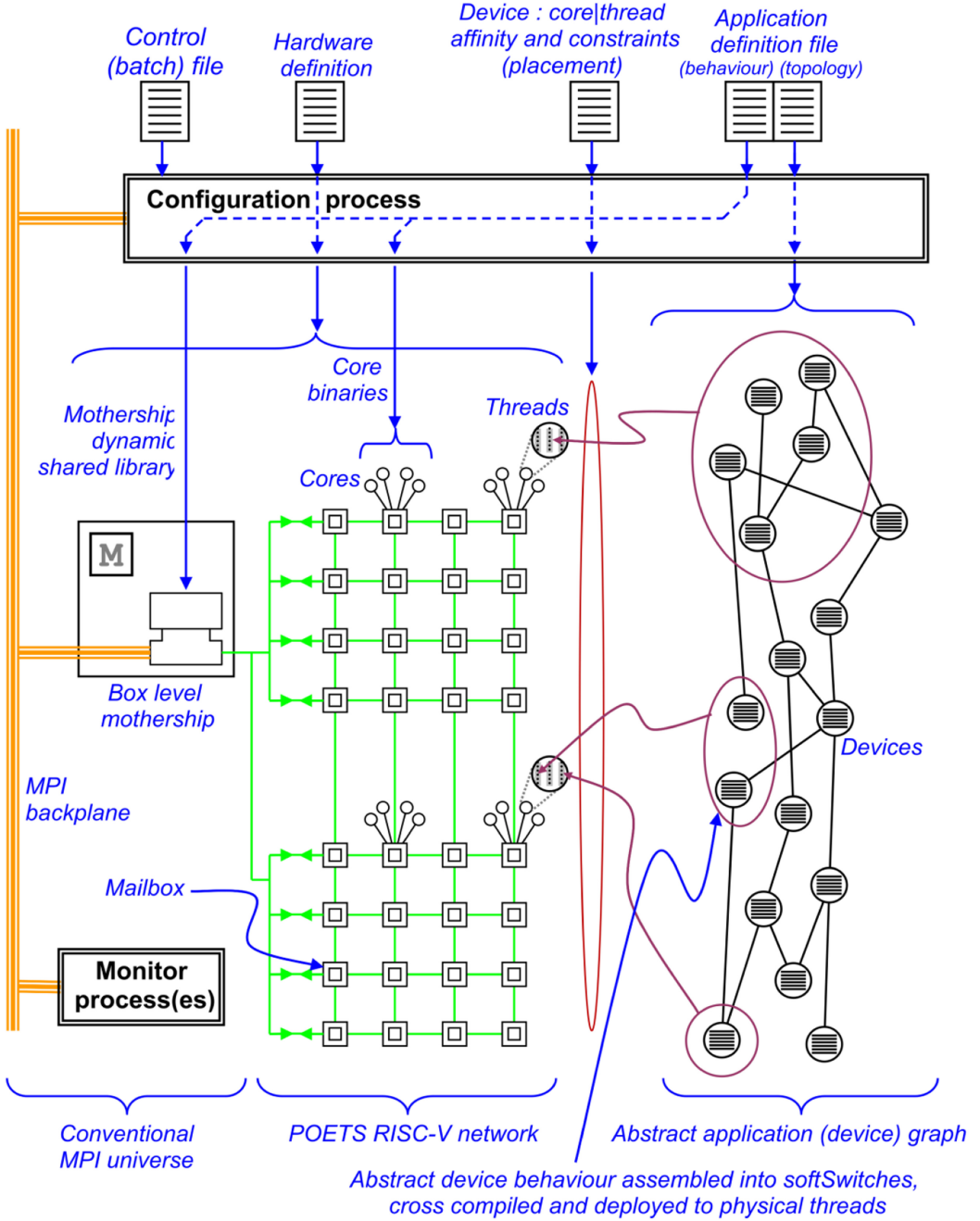


Fig. 3. Overall system layout.

packet consumption, locally or globally, – then 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 [13–15]. POETS takes a middle route: The handlers *request*

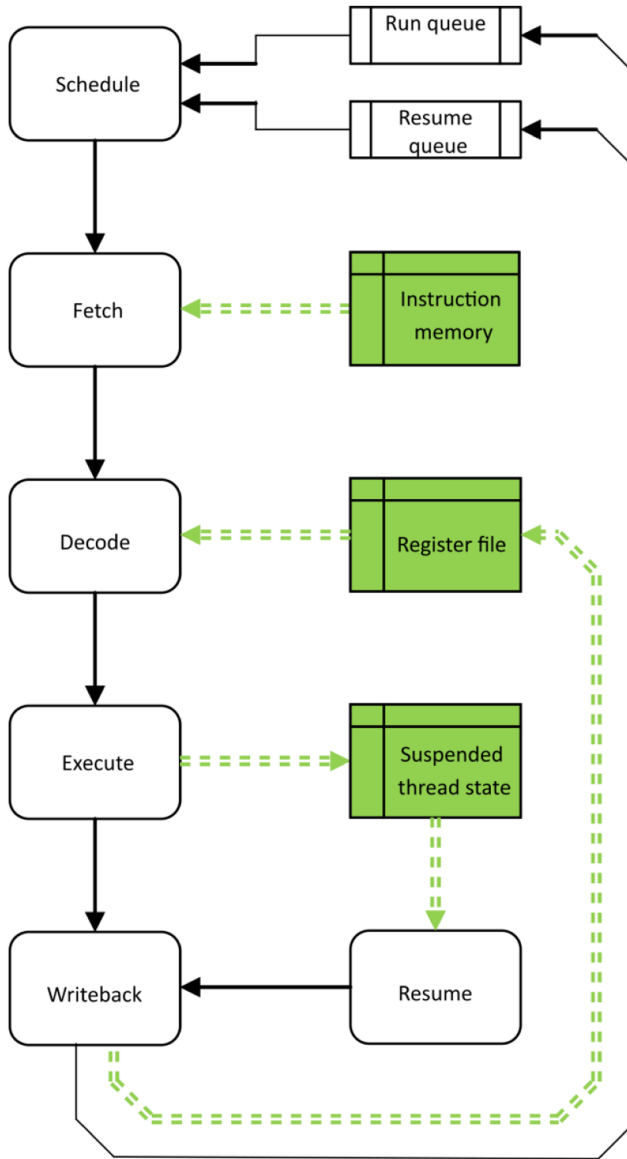


Fig. 4. POETS core pipeline: ≤ 1 instructions/thread present at any time, resulting in a compact, fast core with high throughput when $|\text{threads}| > |\text{stages}|$.

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.

2.2.1 Variations on the Hardware. The principal point of this article is to describe the behaviour of DPD on an event-triggered architecture, in particular the (near constant) scaling behaviour

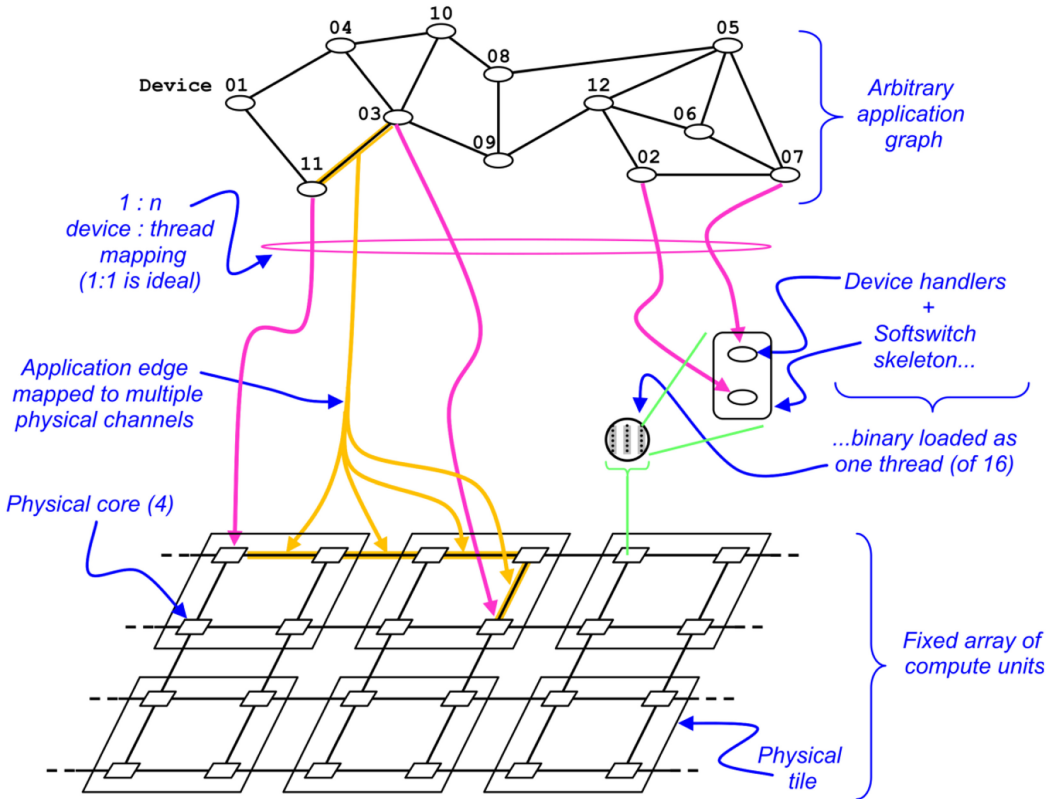


Fig. 5. The arbitrary device application graph is mapped to the regular underlying POETS hardware.

(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 [13–15], 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.

2.3 Configuration

The hardware is a fixed (notwithstanding the intrinsic reprogrammability of the underlying 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 four 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)).

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 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.

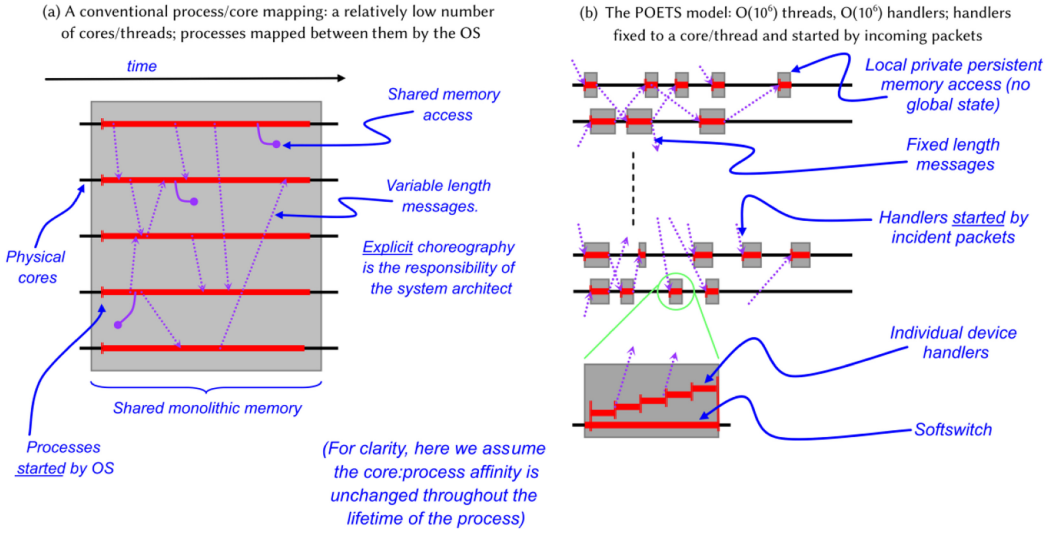


Fig. 6. Conventional execution model (a) and the POETS execution model (b).

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), then 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 article) 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

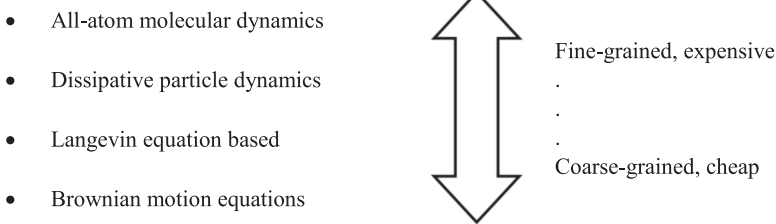
Designing software to run on massively parallel microsystems such as POETS, where there is no “overseer” operating system, the communications system is non-deterministic (from the perspective of the motherships), and there is little internal visibility or debug is challenging. Sniffing

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 these data are 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 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, then 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 after 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 computing 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 References [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 timescales 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

³A *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.

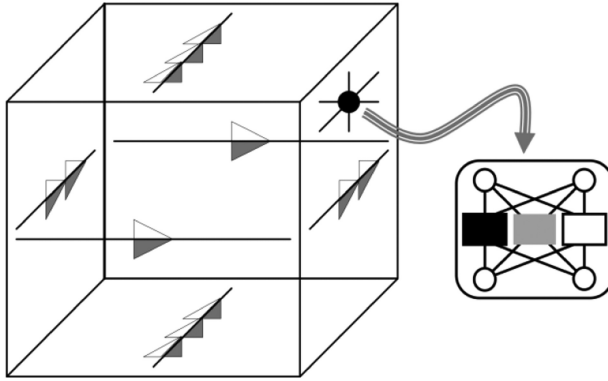


Fig. 7. Triple torus application graph mapped to the compute stack.

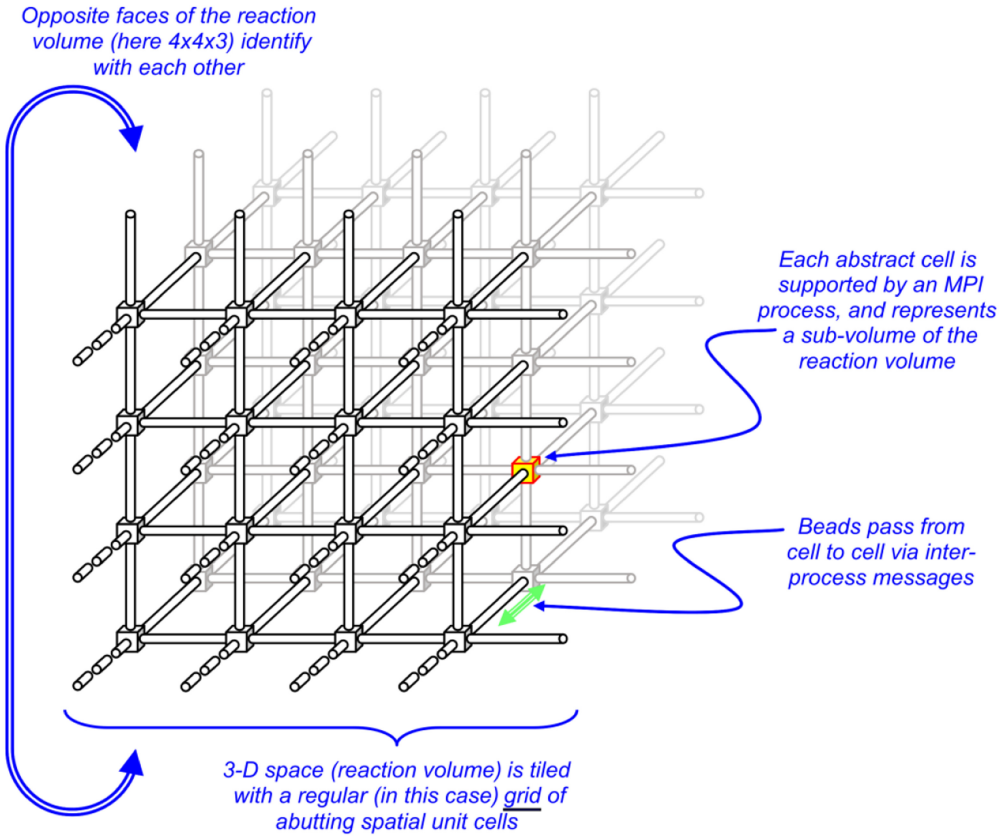


Fig. 8. DPD data layout in a parallel environment.

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 (Figures 7 and 8).

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

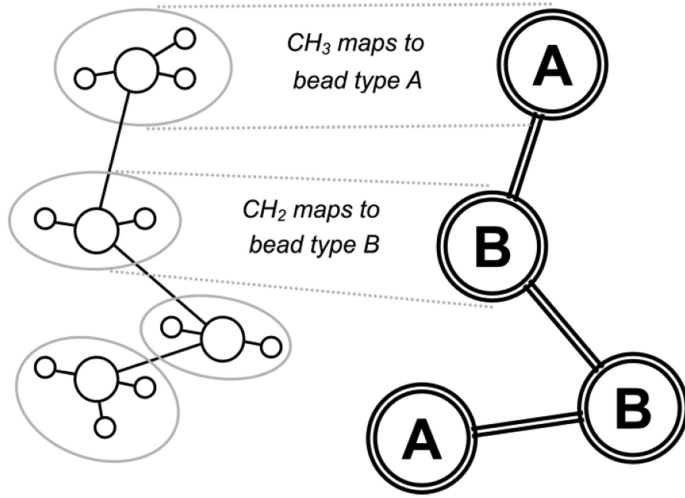


Fig. 9. The structure of linear butane (14 atoms: $\text{CH}_3\text{CH}_2\text{CH}_2\text{CH}_3$) maps to just four DPD beads A-B-B-A.

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.

Each bead i has an associated point position vector \mathbf{r}_i within the simulation volume and a vector \mathbf{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 that 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 (two-body)
- Bead-bead torsional chemical bonds (three- and four-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 article is on how the data representing the system and their 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 article.

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 r_0 [7]. This is captured by the function that defines the force between two beads i and j separated by the Euclidean distance $r_{ij} = r_i - r_j$,

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

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

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

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

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

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

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

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,

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

The coefficients γ and σ are not independent; numerical details may be found in Reference [7]. ξ 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 Reference [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 per cube. Each processing cube will have a *halo* (or *ghost*) layer immediately inside its boundary, 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 (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. To correctly calculate the pairwise 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,

$$T(t) = (k_B N_f)^{-1} \sum_i^N m v_i^2(t). \quad (5)$$

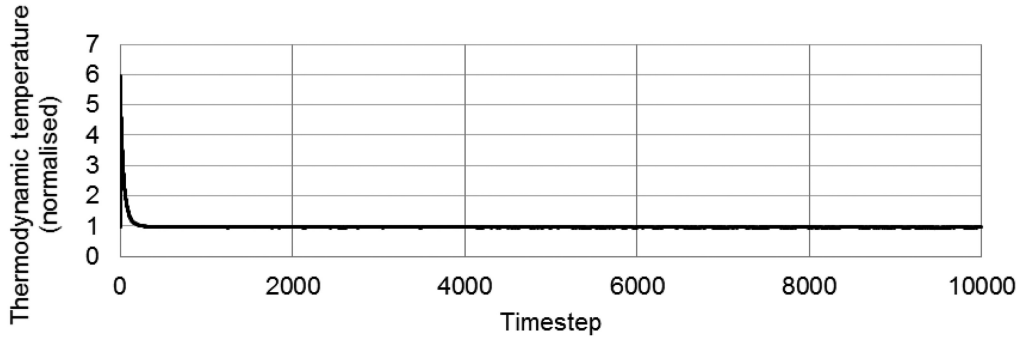
- (5) On setup, a “stop time” is broadcast to all cells; when every cell reaches that simulation (irrespective of the wall-clock time), the analysis terminates.

5 IMPLEMENTING DPD IN A POETS ENVIRONMENT

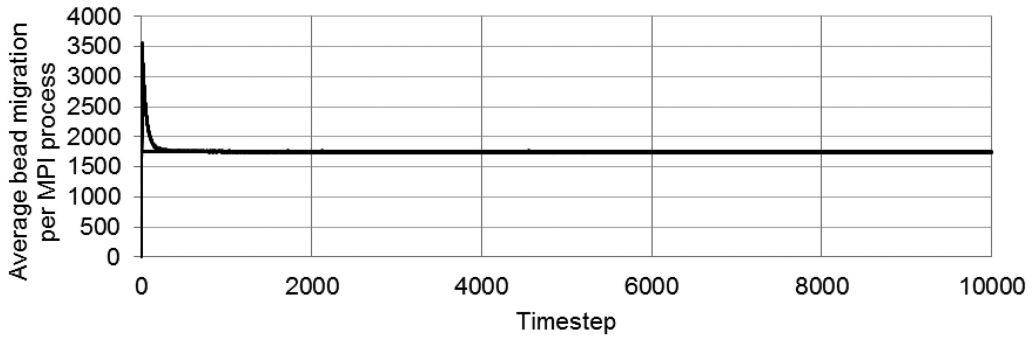
This algorithmic process is implemented in an event-driven graph-based environment in almost exactly the same way (Figure 8), and the differences lie in the numeric parameters of the simulation. Again, with periodic boundary conditions, our application graph consists of a three-dimensional (triple) torus. 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

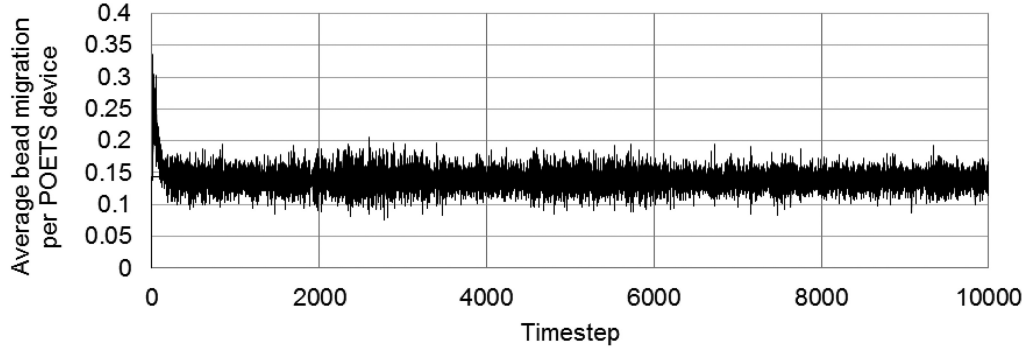
With POETS, the design intent is that we have available not a handful of processes but tens of thousands of devices (ultimately millions—the system is designed to be scalable). The absolute number of beads necessary to perform sensible experiments is obviously a function of the SuS, but rather than have 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



(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)

Fig. 10. Reasonable behavior: temperature and bead migration.

one bead. The cell size is chosen to be equal to 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).

The DPD problem used in this article is the phase separation of three immiscible fluids, simulated over a period of 10,000 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 ($8 \times 8 \times 8$ 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 $\sim 10^4$ beads.

Figure 10(c) shows the average bead transit density for the equivalent POETS simulation: 10^6 ($100 \times 100 \times 100$) cells, each hosting two to three beads. The *absolute* numbers are very low and hence the average is noisy.

5.2 Managing Causality

Managing the passage of simulated time requires more care. On a platform where inter-process synchronisation 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 synchronisation 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, then 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

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

⁴Without 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.

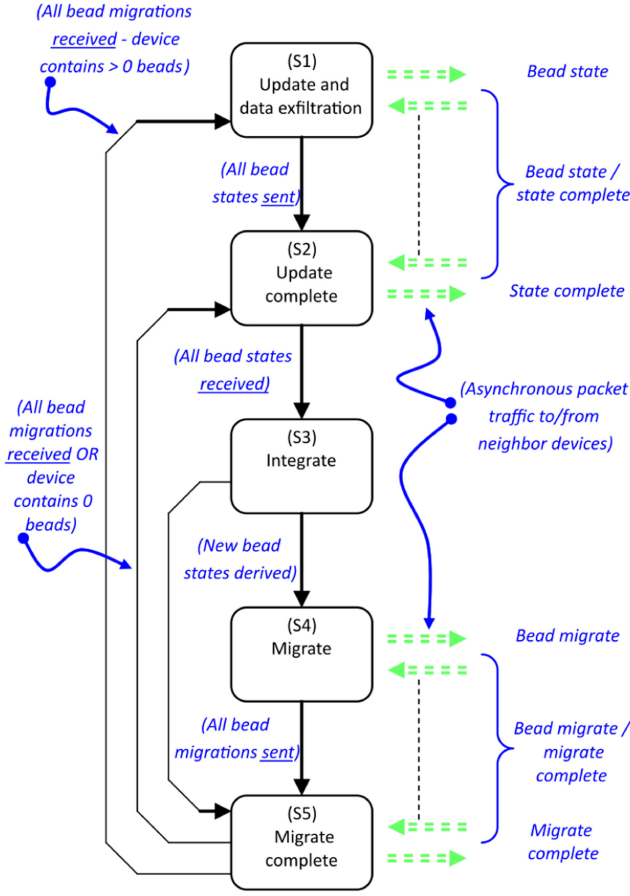


Fig. 11. State automaton realizing a device.

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 synchronisation 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 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 $1 \cdot h$ is a significant strength—it provides automatic fine grained localised load balancing in space and wall-clock 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 synchronisation primitive would not be able to do. The final RNG function is

$$H(i, j, t) = (id(i)^{S(t)} * id(j)) + (id(j)^{S(t)} * id(i)). \quad (6)$$

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

⁵Informal 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.

physical hardware. In our hardware architecture the mailbox–mailbox (intra-board) communication is very low, of the order of 5 ns 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 2×3 board mesh in a 2×4 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 $\sim 9.2 \mu\text{s}$. 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 $10\text{-}\mu\text{s}$ worst-case delay is much less than the 10 ms per timestep that we see in practice, so given the GALS approach taken there appears to be a lot of headroom 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 $29 \mu\text{s}$ 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 and 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

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

(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 forward 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 5,000,000; the relative abundance of the three species is 60:30:10 in every run.

6.2 Reasonable Behaviour?

We are integrating equations that contain a non-deterministic component (Equation (4)) on a message-passing hardware platform that itself has no global synchronisation. 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 that 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?

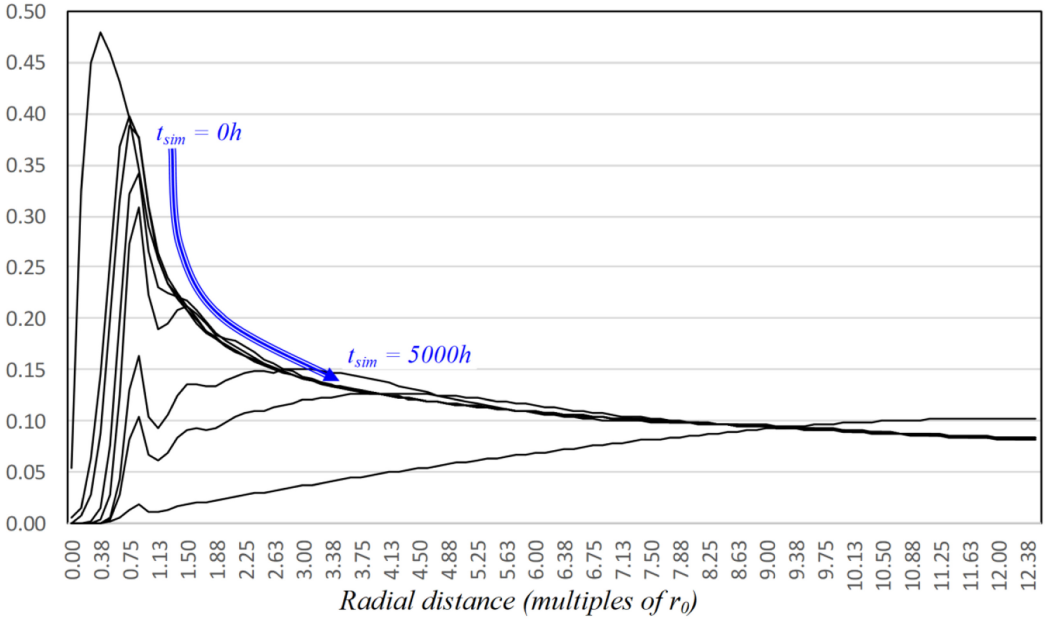


Fig. 12. Radial distribution function changing in the course of the simulation.

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(c)) 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, however, 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 article; 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

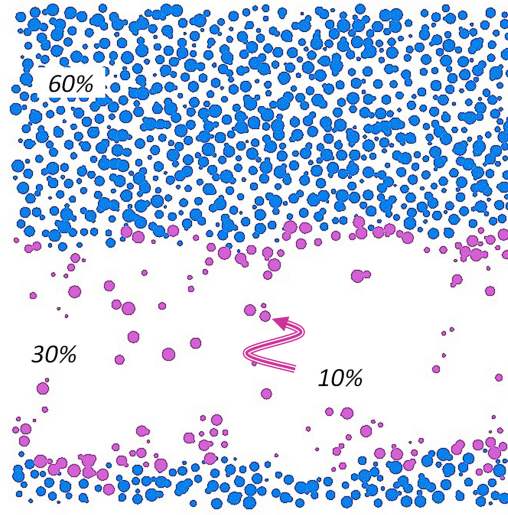


Fig. 13. Emergent behavior: beads separating (spontaneously) into layers (30% "white" beads omitted for clarity).

species in the experimental setup. It is consistent with the results generated from References [35, 36] and hence indirectly with physical experiments.

Indication 4: Bit-level consistency. We have a sequential software simulator that 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 timesteps (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.

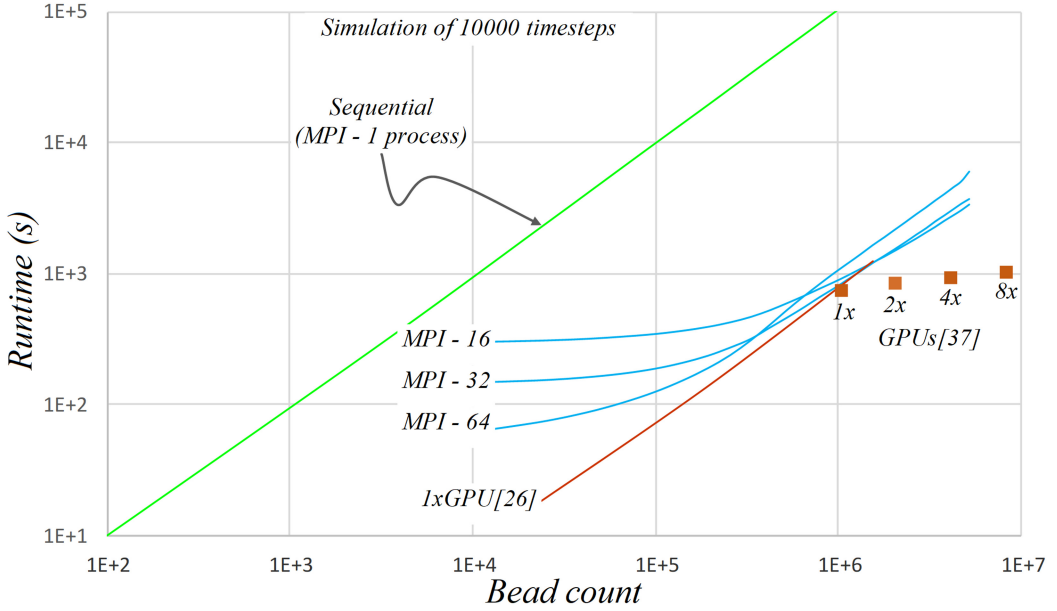


Fig. 14. Runtime vs. bead count on COTS architectures.

6.3 Wall-clock Performance

The focus of this work is on scaling and wall-clock speed. The results presented in this section (Figures 14 and 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 10,000).

6.3.1 COTS Platforms and 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 parallelisation, quite quickly these are overwhelmed by the cost of serialisation 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 3D structure of the GPU. However, once an application moves away from this ideal match, a GPU implementation will become less competitive.

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 one, two, three, and eight 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.

6.3.2 POETS 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

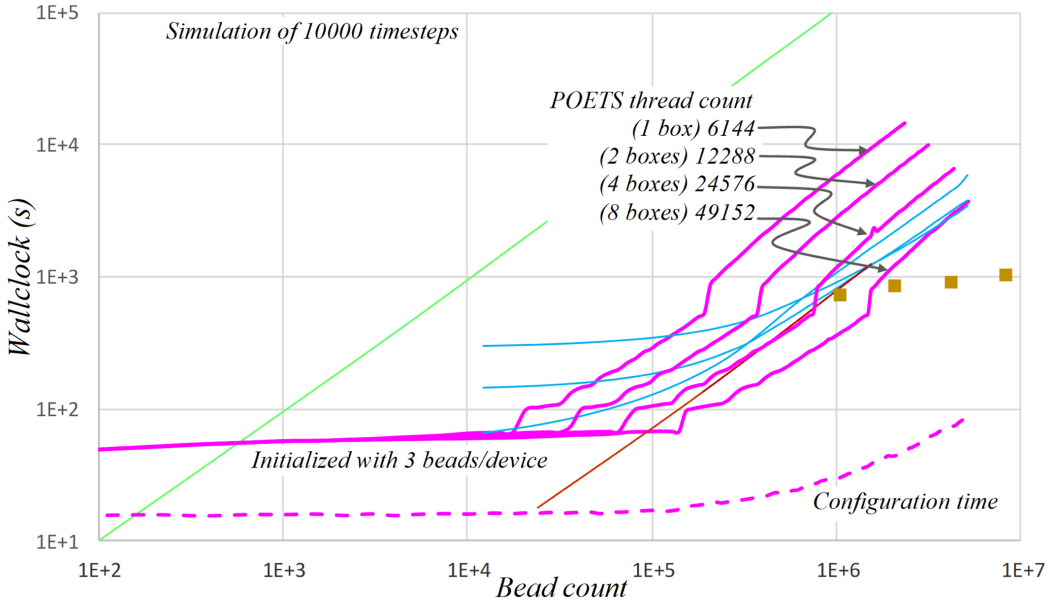


Fig. 15. Runtime vs. bead count on POETS; results superposed on the COTS results of Figure 14.

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.

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 other's 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 serialisation 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 serialisation 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

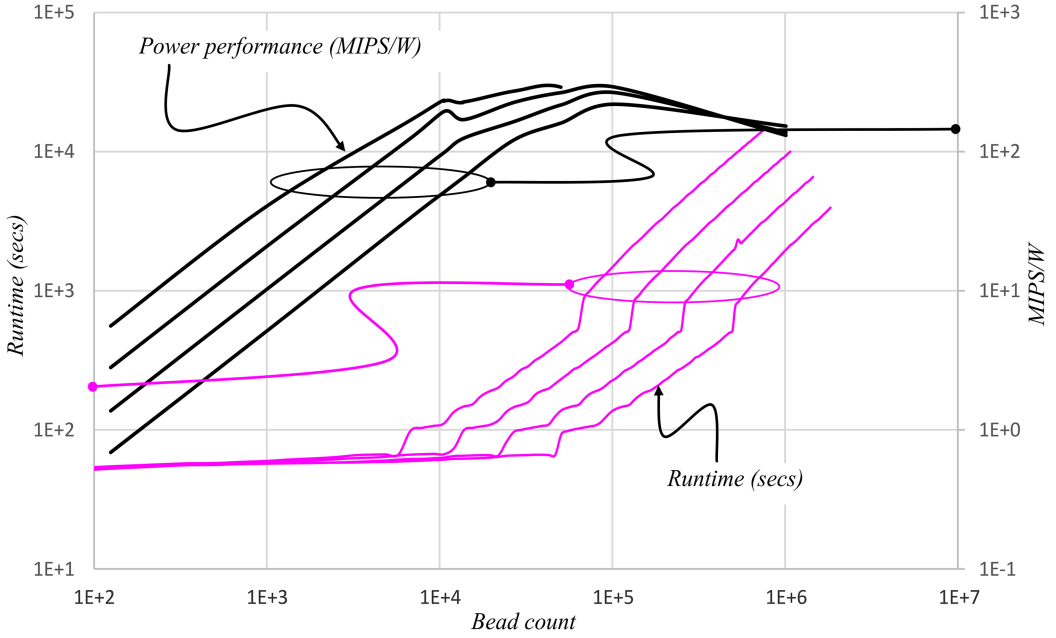


Fig. 16. Power consumption vs bead count.

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 stabilisation startup time). Nevertheless, it is useful to look at the energy–compute relationships. Each compute FPGA dissipates around 50 W, and ignoring the “infrastructure” energy costs (motherships and associated interface circuitry in Figure 1), Figure 16 (right 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 $4 \times$ 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 utilisation peaks at 97%, which is very high for such a fine-grained workload. 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 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 per 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 Total Dissipated Power of each system: The eight-box POETS system dissipates around 2,400 W, the MPI system dissipates around 500 W, and a single Piz Daint GPU dissipates around 250 W. 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 second-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, then 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 article.

This methodology can be utilized in other domains, such as 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 [40–42] that the scaling relationships (cf. Figure 15) 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 eight-box system, we achieved 126 M neuron steps/wall-clock seconds with 2.1 B spikes (packets) delivered/wall-clock second. 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) [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 include

- 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), and
- reactive analogue circuit simulation (work in progress).

An overview may be found in Reference [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 article, 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 $\times 100$; this can be improved almost arbitrarily by adding more hardware—the scaling curve is simply pushed to the right. (Optimistic estimates⁶ 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 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

⁶Tuning 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 $20\text{--}30r_0$ on a side. POETS has $O(1)$ beads/device, which suggests that a further $O(20^3)$ speedup should be feasible.

humans explore a problem: If justifying the compute budget *a priori* is no longer a necessary burden, then 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 toward 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.

ACKNOWLEDGMENTS

The codebase is available at github.com/Osprey-DPD/osprey-dpd, github.com/POETSII/dpd-baremetal, and github.com/POETSII/tinsel.

REFERENCES

- [1] Michael E. Glazier and Anthony P. Ambler. 1984. Ultimate: A hardware logic simulation engine. In *Proceedings of the 21st Design Automation Conference (DA'84)*. 336–342.
- [2] Edson P. Ferlin, Heitor S. Lopes, Carlos R. Erig Lima, and Mauricio Perretto. 2011. Prada: A high-performance reconfigurable parallel architecture based on the dataflow model. *Int. J. High Perf. Syst. Arch.* 3, 1 (2011), 41–55.
- [3] Brian W. Hollocks. 2006. Forty years of discrete event simulation. *J. Operat. Res. Soc.* 57, 12 (2006), 1383–1399.
- [4] Andrew F. Carpenter. 1985. *An Engine for the Multi-level Simulation of Digital Systems*. Ph.D. Dissertation, University of Manchester.
- [5] Richard Membarth, Frank Hannig, Jürgen Teich, and Harald Köstler. 2012. Towards domain-specific computing for stencil codes in HPC. In *SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE, 1133–1138.
- [6] Prashant Singh Rawat, Miheer Vaidya, Aravind Sukumaran-Rajam, Mahesh Ravishankar, Vinod Grover, Atanas Rountev, Louis-Noël Pouchet, and Ponnuswamy Sadayappan. 2018. Domain-specific optimization and generation of high performance GPU code for stencil computations. *Proc. IEEE* 106, 11 (2018), 1902–1920.
- [7] Robert D. Groot and Patrick B. Warren. 1997. Dissipative particle dynamics: Bridging the gap between atomistic and mesoscopic simulation. *J. Chem. Phys.* 107, 11 (1997), 4423–4435.
- [8] Matthew F. Naylor, Simon W. Moore, and David B. Thomas. 2019. Tinsel: A manythread overlay for FPGA clusters. In *Proceedings of the 29th International Conference on Field Programmable Logic and Applications*. IEEE, 375–383.
- [9] Andrew 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. In *Proceedings of the International Conference on Parallel Computing (ParCo'19)*.
- [10] P. J. Hoogerbrugge and Johannes M. Koelman. 1992. Simulating microscopic hydrodynamic phenomena with dissipative particle dynamics. *Europhys. Lett.* 19, 3 (Jun 1992), 155–160.
- [11] Pep Espanol and Patrick Warren. 1995. Statistical mechanics of dissipative particle dynamics. *Europhys. Lett.* 30, 4 (1995), 191.
- [12] Andrew D. Brown. 2016. Event-driven computing. In *Proceedings of the IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP'16)*.
- [13] Steve B. Furber, David R. Lester, Luis A. Plana, Jim D. Garside, Eustace Painkras, Steve Temple, and Andrew D. Brown. 2012. Overview of the spinnaker system architecture. *IEEE Trans. Comput.* 62, 12 (2012), 2454–2467.
- [14] Andrew D. Brown, Steve B. Furber, Jeffrey S. Reeve, Jim D. Garside, Kier J. Dugan, Luis A. Plana, and Steve Temple. 2014. SpiNNaker—Programming model. *IEEE Trans. Comput.* 64, 6 (June 2014), 1769–1782. DOI: [10.1109/TC.2014.2329686](https://doi.org/10.1109/TC.2014.2329686)

- [15] Andrew D. Brown, Rob Mills, Kier J. Dugan, Jeff S. Reeve, and Steve B. Furber. 2015. Reliable computation with unreliable computers. *IEE Comput. Digit. Techn.* 9 (2015), 230–237. <https://doi.org/10.1049/iet-cdt.2014.0110>
- [16] Saroj A. Shambharker. 2015. A study on setting processor or CPU affinity in multi-core architectures for parallel computing. *Int. J. Sci. Res.* 4, 5 (2015), 1987–1990.
- [17] Brian W. Kernighan and Shen Lin. 1970. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Techn. J.* 49, 2 (1970), 291–307.
- [18] Benedict R. Gaster, Lee Howes, David R. Kaeli, Perhaad Mistry, and Dana Schaa. 2011. *Heterogeneous Computing with OpenCL*. Morgan Kaufmann.
- [19] Pep Espanol and Patrick B. Warren. 1995. Statistical mechanics of dissipative particle dynamics. *Europhys. Lett.* 30, 4 (1995).
- [20] Roger W. Hockney and James W. Eastwood. 1988. *Computer Simulation Using Particles*. IOP Publishing.
- [21] Daan Frenkel and Berend Smit. 2002. *Understanding Molecular Simulation*. Academic Press.
- [22] Milos Krstic, Eckhard Grass, Frank K. Gürkaynak, and Pascal Vivet. 2007. Globally asynchronous, Locally Synchronous circuits: Overview and out-look. *IEEE Des. Test Comput.* 24, 5 (Sept.-Oct. 2007), 430–441. DOI : [10.1109/MDT.2007.164](https://doi.org/10.1109/MDT.2007.164)
- [23] Pierre L'Ecuyer and Richard Simard. 2007. TestU01: A software library in ANSI C for empirical testing of random number generators. *ACM Trans. Math. Softw.* 33, 22 (2007).
- [24] Frank Jensen. 2017. *Introduction to Computational Chemistry*. Wiley.
- [25] Nicos S. Martys and Raymond D. Mountain. 1999. Velocity Verlet algorithm for dissipative-particle-dynamics-based models of suspensions. *Phys. Rev. E* 59, 3 (1999), 3733.
- [26] Hao Wu, Junbo Xu, Shengfei Zhang, and Hao Wen. 2011. GPU accelerated dissipative particle dynamics with parallel cell-list updating. *IEIT J. Adapt. Dynam. Comput.* 1, 2 (2011), 33–42.
- [27] Keda Yang, Zhiqiang Bai, Jiaye Su, and Hongxia Guo. 2014. Efficient and large-scale dissipative particle dynamics simulations on GPU. *Soft Mater.* 12, 2 (2014), 185–196.
- [28] Eugene M. Izhikevitch and Gerald M. Edelman. 2008. Large-scale model of mammalian thalamocortical systems. *Proc. Natl. Acad. Sci. U.S.A.* 105, 9 (2008), 3593–3598.
- [29] Robert D. Groot. 2003. Electrostatic interactions in dissipative particle dynamics - simulation of polyelectrolytes and anionic surfactants. *J. Chem. Phys.* 118-11265 (2003). DOI : [10.1063/1.1574800](https://doi.org/10.1063/1.1574800)
- [30] Sriram Pemmaraju and Steven Skiena. 2003. *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. Cambridge University Press, Cambridge.
- [31] Henri Vogt. 1895. *Leçons sur la résolution algébrique des équations*. Nony, Paris.
- [32] Matthew Sacker, Andrew D. Brown, Andy J. Rushton, and Peter R. Wilson. 2004. A behavioural synthesis system for asynchronous circuits. *IEEE Trans. VLSI* 12 9 (2004), 978–994. DOI : [10.1109/TVLSI.2004.832944](https://doi.org/10.1109/TVLSI.2004.832944)
- [33] Daniel J. D. Milton, Andrew D. Brown, Mark Zwolinski, and Peter R. Wilson. 2004. Behavioural synthesis utilising dynamic memory constructs. *IEE Comput. Digit. Techn.* 151, 3 (2004), 252–264. DOI : [10.1049/ip-cdt:20040241](https://doi.org/10.1049/ip-cdt:20040241)
- [34] Andrew D. Brown, Daniel J. D. Milton, Andy J. Rushton, and Peter R. Wilson. 2012. Behavioural synthesis utilising recursive definitions. *IET Comput. Digit. Techn.* 6, 6 (2012), 362–369. DOI : [10.1049/iet-cdt.2012.0006](https://doi.org/10.1049/iet-cdt.2012.0006)
- [35] Julian C. Shillcock and Reinhard Lipowsky. 2005. Tension-induced fusion of bilayer membranes and vesicles. *Nat. Mater.* 4, 3 (March 2005), 225–228. DOI : <https://doi.org/10.1038/nmat1333>
- [36] Julian C. Shillcock and Reinhard Lipowsky. 2002. Equilibrium structure and lateral stress distribution of amphiphilic bilayers from dissipative particle dynamics simulations. *J. Chem. Phys.* 117, 10 (September 2002), 5048–5061. DOI : <https://doi.org/10.1063/1.1498463>
- [37] J. Castagna, X. Gui, M. Seaton, and A. O'Cais. 2020. Towards extreme scale dissipative particle dynamics simulations using multiple GPGPUs. *Comput. Phys. Commun.* 251 (2020), 107159. DOI : [10.1016/j.cpc.2020.107159](https://doi.org/10.1016/j.cpc.2020.107159)
- [38] Matthew 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. 2021. General hardware multicasting for fine-grained message-passing architectures. In *Proceedings of the 29th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP'21)*. 126–133. <https://doi.org/10.1109/PDP52278.2021.00028>
- [39] Julian C. Shillcock, David B. Thomas, Jonathan R. Beaumont, Mark L. Vousden, and Andrew D. Brown. 2021. Coupling bulk phase separation of disordered proteins to membrane domain formation in molecular simulations on a bespoke computer fabric *Membranes* 12, 1 (2021), 17. <https://doi.org/10.3390/membranes12010017>
- [40] Ashur Rafiev, Jordan Morris, Fei Xia, Alex Yakovlev, Matthew Naylor, Simon Moore, David Thomas, Graeme Bragg, Mark Vousden, and Andrew Brown. 2022. Practical distributed implementations of very large scale Petri net simulations. In *Transactions on Petri Nets and Other Models of Concurrency XVI, Lecture Notes in Computer Science*, Vol. 13220, 112–139. https://doi.org/10.1007/978-3-662-65303-6_6.
- [41] Alexander Rast, Mahyar Shahsavari, Graeme M. Bragg, Mark Vousden, David Thomas, and Andrew Brown. 2020. A hardware/application overlay model for large-scale neuromorphic simulation. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'20)*. DOI : [10.1109/IJCNN48605.2020.9206708](https://doi.org/10.1109/IJCNN48605.2020.9206708)

- [42] Mahyar Shahsavari, Jonny Beaumont, David Thomas, and Andrew Brown. 2021. POETS: A parallel cluster architecture for Spiking Neural Network. *Int. J. Mach. Learn. Comput.* 11, 4 (July 2021), 281–285.
- [43] Ashur Rafiev, Jordan Morris, Fei Xia, Rishad Shafik, Alex Yakovlev, Ole-Christoffer Granmo, and Andrew Brown. 2022. Visualization of machine learning dynamics in Tsetlin machines. In *Proceedings of the International Symposium on the Tsetlin Machine (ISTM'22)*. IEEE, Los Alamitos, CA.
- [44] Jordan Morris, Ashur Rafiev, Fei Xia, Rishad Shafik, Alex Yakovlev, and Andrew Brown. 2022. An alternate feedback mechanism for tsetlin machines on parallel architectures. In *Proceedings of the International Symposium on the Tsetlin Machine (ISTM'22)*. IEEE, Los Alamitos, CA.
- [45] Ashur Rafiev, Alex Yakovlev, Ghaith Tarawneh, Matthew Naylor, Simon Moore, David Thomas, Graeme Bragg, Mark Vousden, and Andrew Brown. 2022. Synchronization in graph analysis algorithms on the POETS Many-core architecture. *IET Comput. Digit. Techn.* 16 (2022), 71–88. DOI : [10.1049/cdt2.12041](https://doi.org/10.1049/cdt2.12041)
- [46] Mahyar Shahsavari, David Thomas, Andrew Brown, and Wayne Luk. 2021. Neuromorphic design using reward-based STDP learning on event-based reconfigurable cluster architecture. In *Proceedings of the International Conference on Neuromorphic Systems (ICONS'21)*. 1–8. DOI : [10.1145/3477145.3477151](https://doi.org/10.1145/3477145.3477151)
- [47] Mark L. Vousden, Jordan Morris, Graeme M. Bragg, Jonathan R. Beaumont, Ashur Rafiev, Wayne Luk, David B. Thomas, and Andrew D. Brown. 2023. Event-based high throughput computing: A series of case studies on a massively parallel softcore machine. *IET Comput. Digit. Tech.* 17, 1 (2023), 29–42. <https://doi.org/10.1049/cdt2.12051>
- [48] Jordan Morris, Ashur Rafiev, Graeme Bragg, Mark Vousden, David Thomas, Alex Yakovlev, and Andrew Brown. 2022. An event driven approach to genotype imputation on a RISC-V FPGA cluster. (unpublished).
- [49] Matthew F. Naylor, Simon W. Moore, David B. Thomas, Jonny R. Beaumont, Shane Fleming, Mark L. Vousden, Theo Markettos, Tom Bytheway, and Andrew Brown. 2021. General hardware multicasting for fine-grained message-passing architectures. In *Proceedings of the 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP'21)*. 126–133. DOI : [10.1109/PDP52278.2021.00028](https://doi.org/10.1109/PDP52278.2021.00028)
- [50] Matthew Naylor, Simon Moore, Andrey Mokhov, David Thomas, Jonny Beaumont, Shane Fleming, Theo Markettos, Tom Bytheway, and Andrew Brown. 2020. Termination detection for fine-grained message-passing architectures. In *Proceedings of the IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP'20)*, F. Hannig, J. Navaridas, D. Koch, and A. Abdelhadi (Eds.). IEEE, 17–24.
- [51] George Karypis and Vipin Kumar. 1999. A fast and high quality multi-level scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* 30 1 (1999), 359–392.
- [52] Mark Vousden, Graeme McLachlan Bragg, and Andrew Brown. 2022. Asynchronous simulated annealing on the placement problem: A beneficial race condition. *J. Parallel Distrib. Comput.* 169 (2022), 242–251.

Received 21 March 2022; accepted 12 January 2023