



POETS

Event-based parallel computing - partially ordered event-triggered systems

Final report for EPSRC Programme Grant EP/N031768/1

*Andrew Brown
David Thomas
Wayne Luk
Simon Moore
Alex Yakovlev*

*Mark Vousden
Graeme Bragg
Jordan Morris
Ashur Rafiev
Coral Westoby
Tim Todman*

October 2022



Event-based parallel computing - partially ordered event-triggered systems (POETS)

EPSRC Programme Grant EP/N031768/1

Report to the Advisory Board, October 2022

Foreword

The final meeting of the Advisory Board for the EPSRC programme grant *Event-based parallel computing - partially ordered event-triggered systems (POETS)* will take place on 21st October 2022. The Advisory Board has been set up in fulfilment of an EPSRC requirement; we welcome the opportunity to receive external perspectives on the aims, ambitions and execution of this research, and to reflect upon the outcomes of our efforts.

COVID was disruptive, as it was for most activities, but POETS probably suffered less than its peers. With the exception of the hardware activities at Cambridge, most staff were able to work remotely.

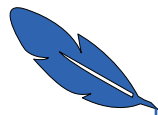
The big loss was the outreach and publicity program; for two years, the POETS team attended almost no meetings, nor gave any external seminars. We are grateful to EPSRC for allowing us to vire the resources allocated for this to other ends, most notably the acquisition of more compute hardware.

The previous Advisory Board report outlined the underlying technology and aspirations of the project; here we build on this, and further, we outline where we hope to take the technology in the future.

Meeting structure

The Advisory Board meeting itself will take place over one day - 21 October 22. Most members of the board will be arriving the previous evening (20 October - dinner is available), others will arrive on the morning of the 21st: we have a full agenda for the day, so we will start at 09:00 on the 21st. The meeting will close at approximately 17:00, after the Advisory Board feedback to the Investigators.

The presentations will follow the approximate stack trajectory of hardware-middleware-application, the sessions being led by the PIs at the various four institutions, but the details will be provided by the individual research staff, reflecting the cross-institution and collaborative nature of the work undertaken.



Investigators

Andrew Brown UoS
Alex Yakovlev UoN
Simon Moore UoC
David Thomas UoS
Wayne Luk IC

Research staff

Mark Vousden UoS
Graeme Bragg UoS
Mahyra Shahsavari ICL
Tim Todman ICL
Ashur Rafiev UoN
Jordan Morris UoN
Tom Bytheway UoC

External collaborators

Julian Shillcock
Jonny Wray
Ole-Christoffer Granmo
Eric Moreno
Hamza Javed

EPFL
eTherapeutics
University of Agder, Norway
California Institute of Technology
CERN, Switzerland



POETS Advisory Board meeting

20-21 October 2022

Agenda
Imperial College, London

20 October 2022

- *Afternoon/evening: Arrival*

18:30 Pre-dinner drinks

19:00 Dinner

21 October 2022

- *Breakfast*

09:00 Welcome, introductions, housekeeping

09:10 The Big Picture - the structure of the day

09:30 Management - challenges and cooperation, the impact of COVID

10:00 The underpinning hardware

- *10:50 Coffee*

11:05 The software stack

- *12:05 Lunch*

13:05 Applications (I)

- *14:45 Tea*

15:00 Applications (II)

15:40 Impact

16:10 Closing comments

16:25 Advisory board private meeting

16:40 Advisory board provide feedback to the PIs; meeting closes



Joining instructions

1. If you are staying at the Strathmore Hotel on 20 October 2022

Strathmore Hotel, 41 Queen's Gate Gardens, London SW7 5NB.

Tel: +44 (0) 20 7584 0512

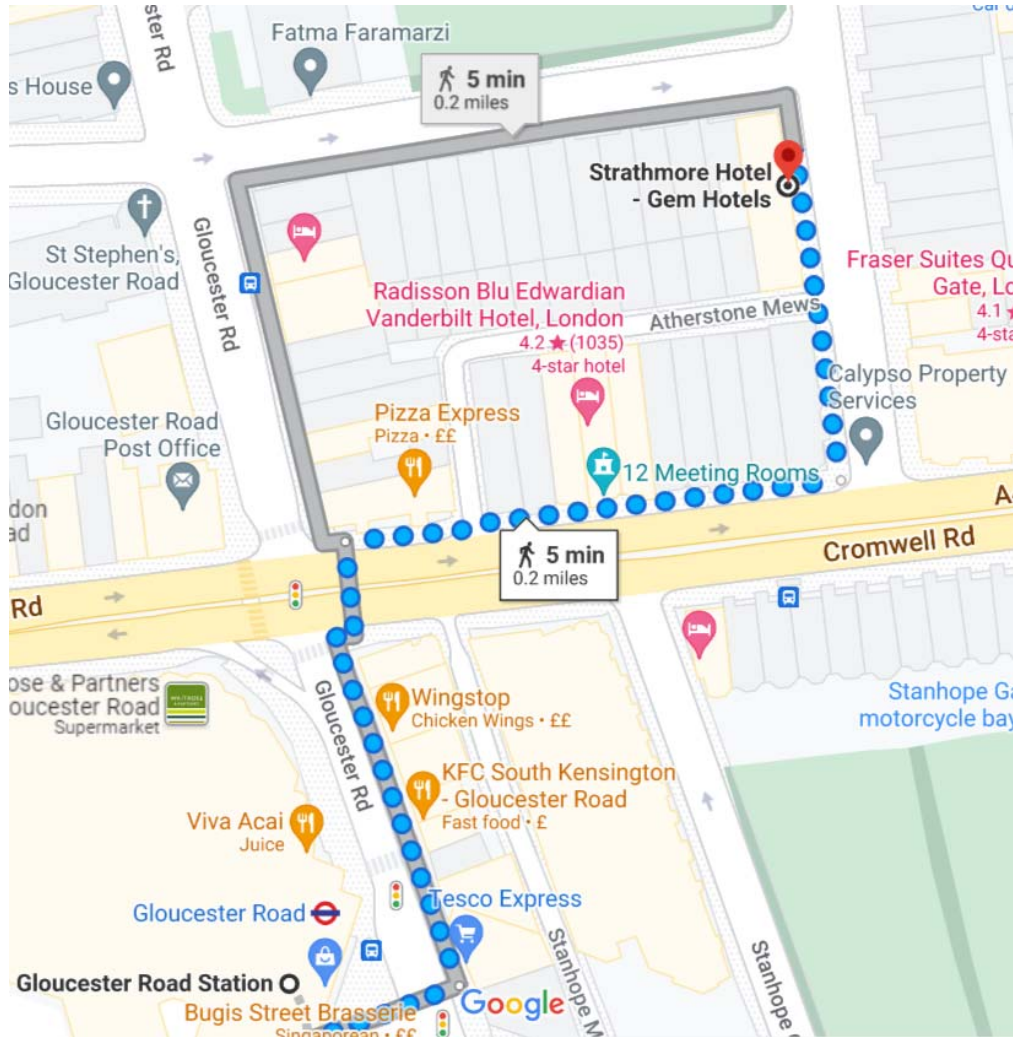
Email: strathmore@gemhotels.com

<https://www.gemhotels.com/hotels-london/strathmore-hotel>

Check-in: 14:00, Check-out: 11:00 (rooms are prepaid)

Breakfast included, 07:00-10:00 (our meeting on 21 October starts at 09:00)

5 minutes walk from Gloucester Road Underground Station

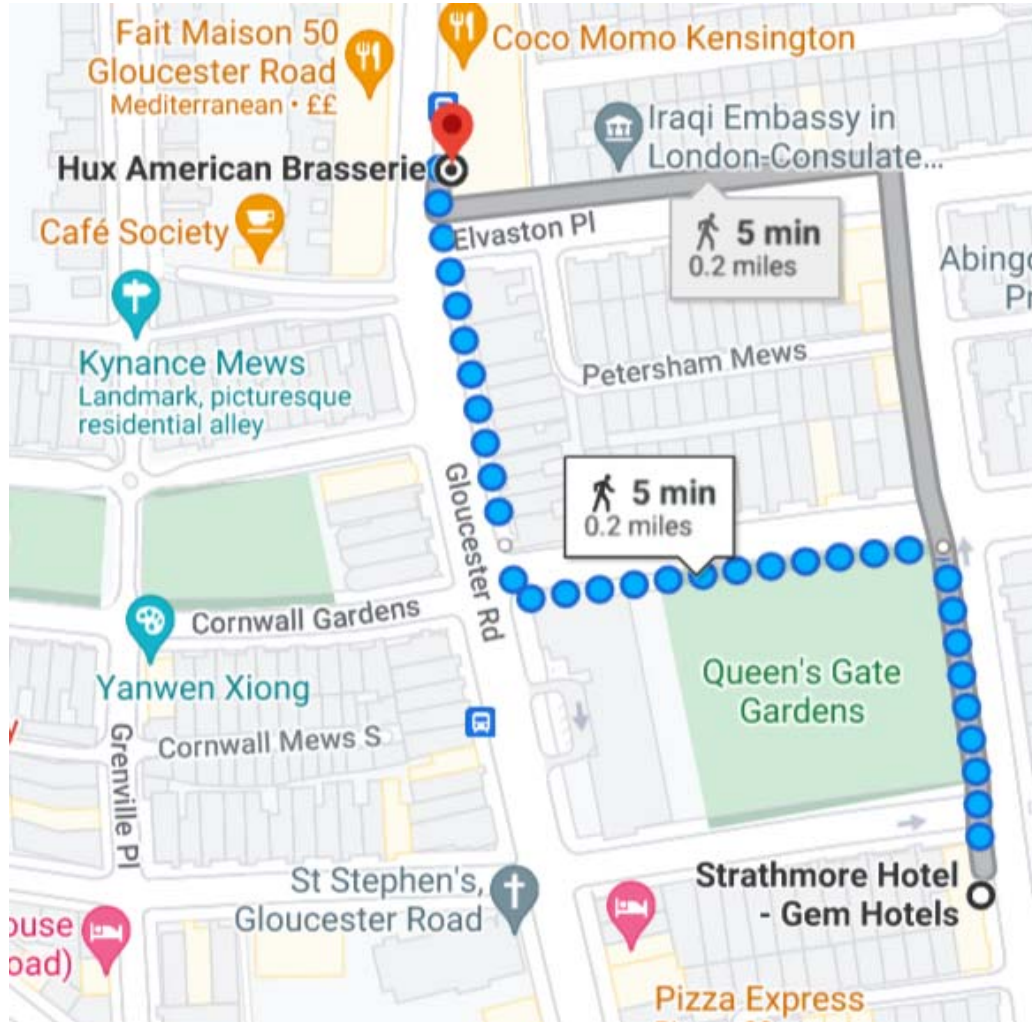




2. Dinner on 20 October 2022: 6:30pm at HUX American Brasserie

HUX American Brasserie, 25-35 Gloucester Road

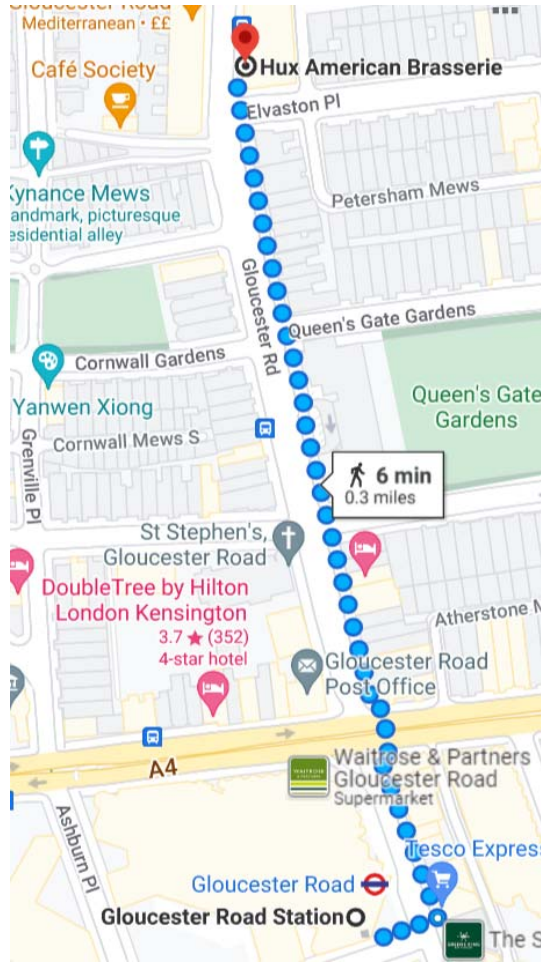
5 minutes walk from the Strathmore Hotel:





HUX American Brasserie, 25-35 Gloucester Road

6 minutes walk from Gloucester Road Underground station:



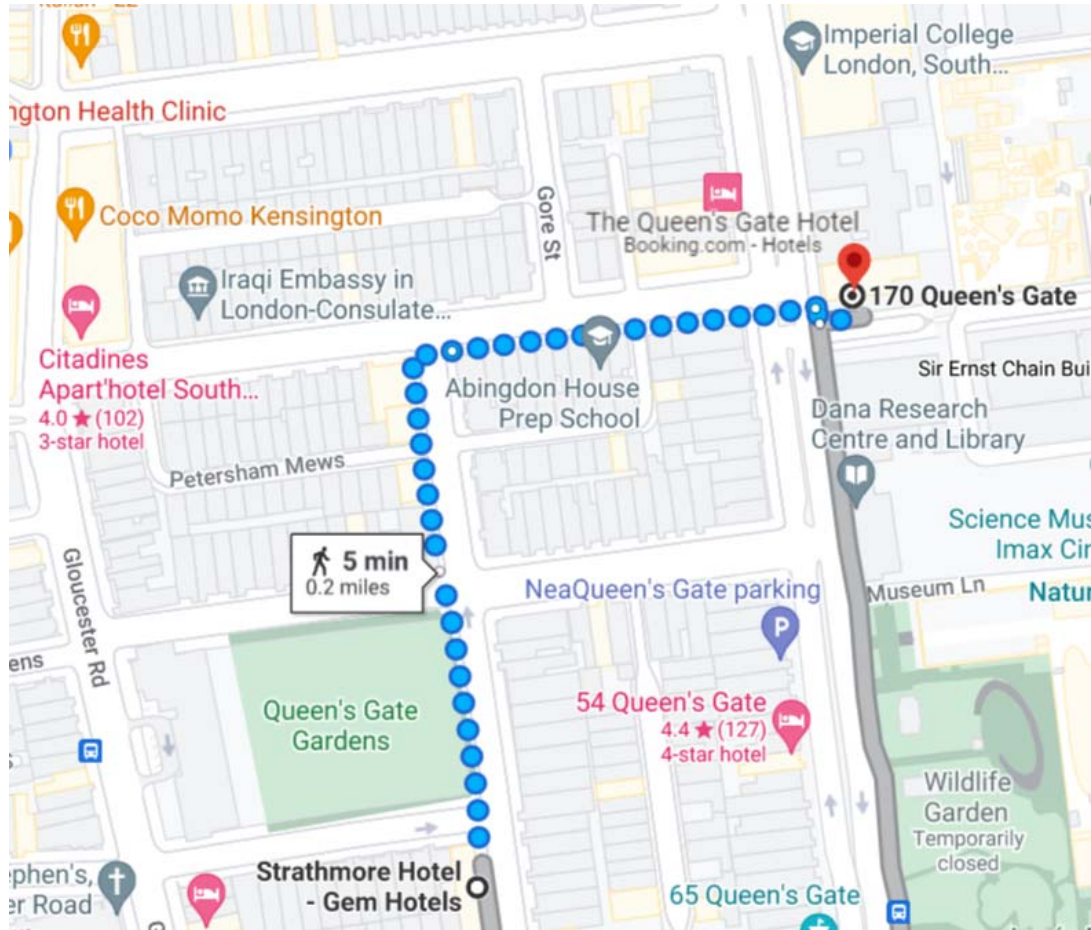


3. Meeting on 21 October: 09:00

Solar Room, 170 Queen's Gate

<https://www.imperialvenues.co.uk/south-kensington/170-queens-gate/the-solar-room/>

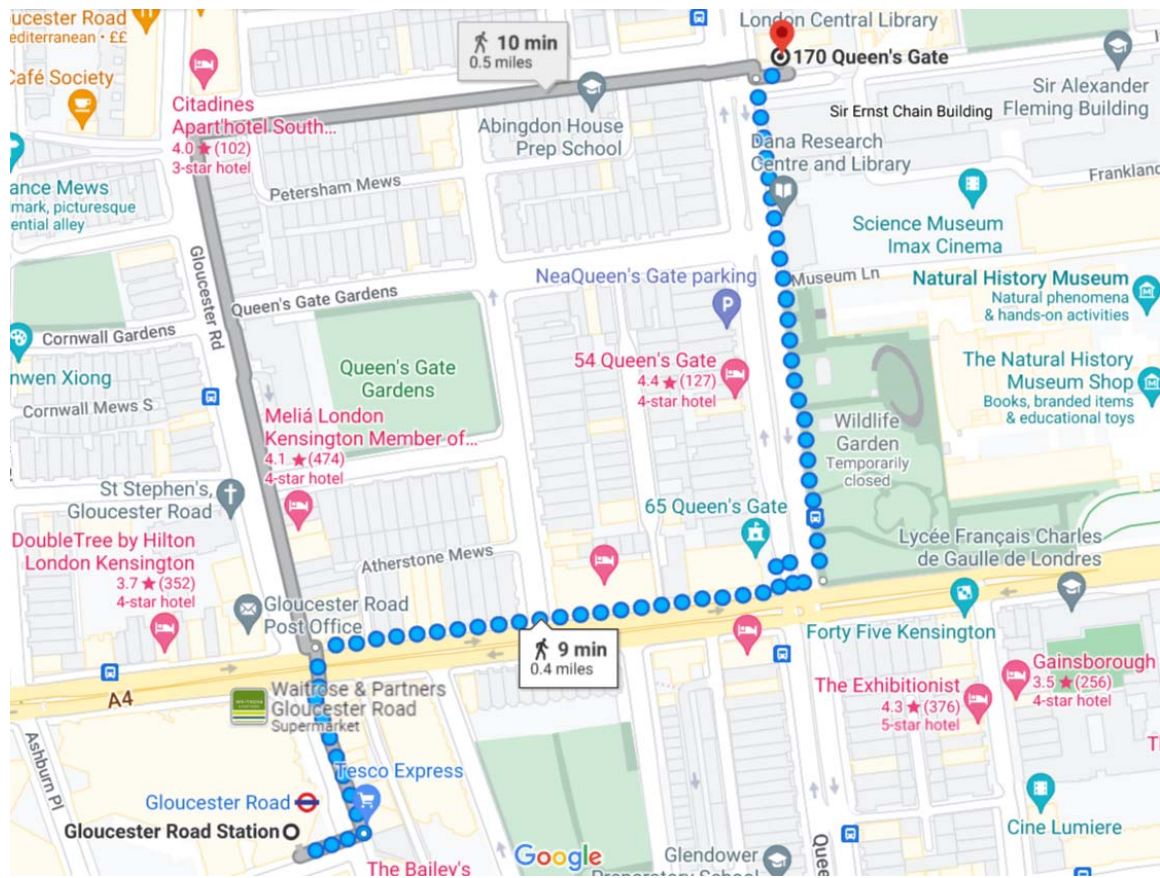
5 minutes walk from Strathmore Hotel:





9 minutes walk from the Gloucester Road Underground Station:

Solar Room, 170 Queen's Gate



No formal dinner has been arranged for the evening of 21 October, but the PI would be delighted to take any project members to dinner, should they so wish.

If there are any problems, please call Wayne on 07434 659001 or Tim on 07796 496388.



POETS - report to the Advisory Board

October 2022

1. Introduction

- 1.1 Staffing
- 1.2 Expenditure

2. The big picture

- 2.1 The problems with the status quo
- 2.2 Event-based compute - what do we get?
- 2.3 How does it work ?
- 2.4 The packet storm
- 2.5 How do we know this is a good idea ?

3. Physical hardware

- 3.1 Outline
- 3.2 Component delays and COVID-19
- 3.3 Tinsel overlay
- 3.4 Tinsel v2.0 for Stratix 10
- 3.5 Custom enhancements
- 3.6 Beyond POETS: using the second-generation hardware

4. The software stack

- 4.1 Introduction
- 4.2 Mode of (abstract) operation
- 4.3 System configuration
 - 4.3.1 The Orchestrator mapper
 - 4.3.2 The Orchestrator Composer
- 4.4 Tools infrastructure & ecosystem
 - 4.4.1 Libraries
 - 4.4.2 Simulation
 - 4.4.3 Debugging and Verification
- 4.5 Visualisation - data & traffic fluxes

5. Domain specific applications - I

- 5.1 Introduction
- 5.2 Graph traversal
 - 5.2.1 Drug discovery
 - 5.2.2 Seismic raytracing
- 5.3 Genomic imputation
 - 5.3.1 Methodology
- 5.4 Micromagnetic modelling
 - 5.4.1 The problem
 - 5.4.2 Results



- 5.5 Reactive systems
 - 5.5.1 Demonstration circuits
 - 5.5.2 Application
- 5.6 Partial discharge
 - 5.6.1 The POETS model
- 5.7 Large-scale Petri Net simulations
- 5.8 Machine learning using Tsetlin Machines
- 5.9 High-energy physics
- 5.10 Machine learning on a national scale
 - 5.10.1 SIR: a simplified epidemiological model
 - 5.10.2 Graph convolutional networks: a machine learning approach for irregular problems

6. Domain specific applications - II

- 6.1 Drug discovery
- 6.2 Dissipative Particle Dynamics
 - 6.2.1 Baseline simulation algorithm
 - 6.2.2 Angle bond support
 - 6.2.3 Integration
 - 6.2.4 Optimisation
 - 6.2.5 Generalisation
 - 6.2.6 Exploration

7. Future technical plans

- 7.1 SONNETS
 - 7.1.1 Motivating example: national-level financial risk analysis
 - 7.1.2 Building a stack
 - 7.1.3 Challenges we want to solve
 - 7.1.4 Event-triggered modelling
 - 7.1.5 Summary
- 7.2 ENNUI
- 7.3 ARIA

8 Closing remarks

Appendix A: Reactive systems

- A.1 Underlying mathematics - limitations
- A.2 The circuit
- A.3 Data layout
- A.4 The packet handlers
- A.5 Sanity check

Appendix B: Publications



1. Introduction

This report represents the state of the POETS programme grant at the end of the final year activities. The introduction provides a precis of staffing and staff movements, comments about the impact of COVID, and an overview of the expenditure.

The bulk of the report is devoted to technical outcomes. The layout follows the traditional report stack: underpinning hardware (section 3), followed by the generic software infrastructure (section 4), and our explorations of the application space (sections 5 and 6). We conclude (section 7) with an outline of our ambitions for this technology in the future.

1.1 Staffing

Some churn in research staff is inevitable during the course of a project of this duration, although informally, we suggest that this is probably less than that experienced by a lot of research projects.

At Southampton, we employed three research fellows. Alex Rast joined us one year into the project, and left in January 2020 to take up a permanent post on the academic staff of Oxford Brookes University. Mark Vousden started in August 2018, over two years after the formal start, and left in June 2022, to take up a permanent post on the academic staff at the University of Southampton. Graeme Bragg joined us in November 2018, and left to become a permanent Teaching Fellow at Southampton in January 2022. David Thomas accepted a post as Chair of Computer Engineering and joined Southampton from Imperial at the beginning of 2022. It proved impossible to recruit further competent staff for the remaining few months of the contract, and as the principal researchers (Andrew Brown, David Thomas, Mark Vousden and Graeme Bragg) were all still within the Department of Electronics, and the hours funded by EPSRC, it was decided to distribute the workstreams accordingly amongst what were now full-time employees of the University. Kath Kerr (the POETS administrator) left in the spring of 2022.

We note with regret that Dr Jeff Reeve, who retired and left the POETS project in July 2018, passed away on 27 July 2019.

At Cambridge, Matt Naylor (research associate) started at the beginning of the project and has led the POETS hardware work, leaving in January 2021. Martijn Bakker joined us for 7 months from October 2018 (research assistant). Tom Bytheway (research assistant) has worked part time on POETS since March 2019. Theo Marketos (senior research associate) has been used for a few months to provide expertise for the physical build, in particular high frequency board-to-board communications and power management. He Li started as a Research Associate in December 2019 and left in September 2020. Coral Westoby started as a Research Assistant in March 2020 and has worked until the end of the project.

At Newcastle, Lead Investigator Andrey Mokhov left the employment of Newcastle University at the end of August 2019 for industry, and so surrendered his role in POETS. Prof Alex Yakovlev took on the role of Newcastle Lead Investigator, expanding his involvement with the project. Dr Ghaith Tarawneh was a part time research associate on the project from May 2019, leaving the University a year later to take up an industrial post. Dr Jordan Morris and Dr Ashur Rafiev joined the project in May and January 2019 respectively, as well as Dr Fei Xia, from the late 2021, and all have remained with the project throughout.

At Imperial, Shane Fleming (who had been with POETS as a research associate for almost two years) left the university in January 2019 for a role in industry. Jonny Beaumont



remained as a researcher till May 2021, then left for a role in a startup. Mahyar Shahsavari left the project after two years in Dec 2021 to take up an academic position in Radboud University in the Netherlands. Tim Todman remains with the project till the end.

2.1 Expenditure

The *overall* expenditure was reasonably consistent with the projected budget at the start of the program; Southampton Finance will be reconciling the final expenditure in the next few months. The only notable hiatus was in the outreach and public engagement programs, which was generously funded by EPSRC at the outset and remained significantly underspent. EPSRC were extremely sympathetic and allowed us to vire resources from the outreach budget to acquire further hardware for the project; our aspirations in this direction are described in section 7.

Of a total initial budget of £6,226,629.02 the *final* over/(under)spend from each partner is approximately

Partner University	(under) / over spend
Southampton	(£40000)
Cambridge	£2000
Newcastle	£3000
Imperial	£10000



2. The big picture

2.1 The problems with the status quo

Moore's Law has given us a doubling of logic density every eighteen months or so for over half a century. It has enabled microelectronics to move from a narrow professional niche into the hands and pockets of every consumer on the planet. However, growth at this rate is not sustainable in nature - in any field - and as process geometries continue to shrink towards the scale of the atom, we face the emergence of fundamental limits which the scaling of current methodologies can no longer overcome; increasingly, far ranging architectural changes are required to utilise the potential of the technology. Many exist, but several major challenges can be identified:

- Power dissipation: it is not possible to power all parts of a state-of-the-art chip at the same time (the *dark-silicon* problem). Simply put, you cannot get the electrical power in and the consequent heat out fast enough to stop the system melting. It has been demonstrated that multiple small processors are correspondingly much more power efficient than fewer large ones, so the deployment of massive cohorts of small processors is an obvious way to push this particular problem down the road.
- As process geometries continue to shrink, issues of reliability and robustness inevitably emerge. In a system of millions of cores, it is unrealistic to expect 100% functionality 'out of the box'; equally, cores will inevitably fail over the lifetime of the system. A POETS system is capable of inconspicuous self-monitoring (and possible recovery) and *graceful performance degradation* in the face of core or communication fabric failure¹.
- A traditional argument against moving to large numbers of cores is the *relative* cost of computation and communication, currently around 1000x in favour of computation - moving data (messages) traditionally is very (and non-linearly) expensive. Unlike a lot of physical system parameters, this multiplier is unlikely to change much in forthcoming decades because it is derived from thermodynamics, not fabrication technology. You can no longer (reasonably) separate the core from the communications: ***the "computer" is the network***. POETS deals with this issue by 'embedding' the cores in a hardware communications fabric (which is truly parallel) and in which the messages are small and of fixed size (a few bytes). With POETS machines, the burden of high-level message choreography is completely removed: systems trade cores against complexity in both compute and communications. Much work is required to understand how to optimise the scheduling of workloads on such machines, but the nature of the task is changing: in the past, a large application was distributed 'evenly' over a few processors and much effort went into scheduling to keep all of the processor resources busy; today, the nature of the cost function is different: *processing is effectively a free resource*.
- The automatic (high-level) parallelisation of general-purpose codes remains a 'holy grail' of computer science, but fine-grain parallelisation is frequently signposted by the underlying mathematics. The problem has been in the past that solutions emerging naturally from the numerical solution technique do not map well (that is, cheaply) onto existing architectures. The POETS architecture changes this: the operational 'sweet spot' lies in the regime of low-level, high density message processing.

¹ A.D. Brown, R. Mills, K.J. Dugan, J.S. Reeve and S.B. Furber "Reliable computation with unreliable computers", IEE Computers and Digital Techniques, 2015, **9**, no 4, pp 230-236, doi: 10.1049/iet-cdt.2014.0110



- Traditional architectures massively over-constrain problem definition. The essence of an algorithm is the overall data-flow, not the precise ordering of the program source. Single-thread optimising compilers are extremely complex systems, and parallel optimisation is not common. Forcing system designers to define behaviour in terms sympathetic to humans (i.e. sequentially) is a very bad place for a compiler to start.

The problem is not *creating* large cohorts of processors, but signposting how they might be productively *used* to perform or enable the sort of analyses that users of big compute demand.

2.2 Event-based compute - what do we get?

POETS trades thread count against speed; each thread *reacts* to data as soon as it becomes available. If data is 'old', a new value can be calculated when fresh data arrives *much faster than the housekeeping necessary to work out if a data item is invalid or not.*

Applications able to benefit from POETS compute will have a complexity scaling curve similar to that shown here - as our problem size grows, more and more threads are used to address small parts of it - the amount of work done by an *individual* thread does not increase, and the communications is entirely hardware, and - from the execution time point of view - almost instantaneous.

POETS is a compute architecture designed to address not a particular application domain, but a large, but specific, compute *pattern*. POETS is an *event-based* computing architecture.

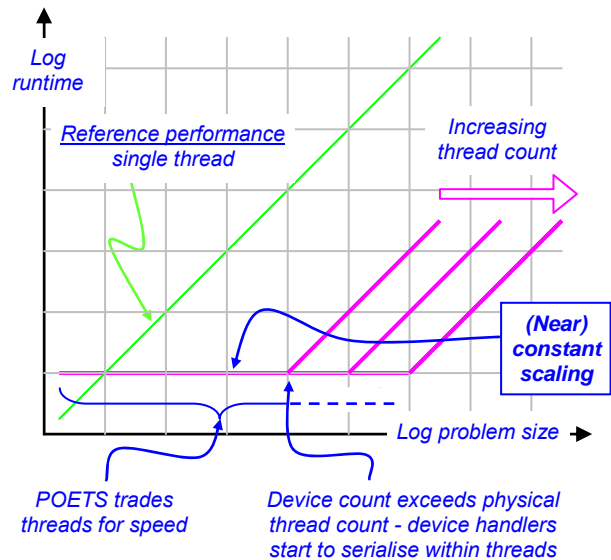


Figure 2.2.a: Idealised POETS performance - (near) constant scaling.

A POETS compute architecture offers (near) constant scaling with problem size.

- Like Message Passing Interface (MPI) programs, multiple processes are executed simultaneously (in POETS, threads) communicating by messages (packets).
 - Unlike MPI, POETS packets are typically small (~64 bytes), fixed size, hardware brokered, and extremely fast due to the absence of a software infrastructure - communication is genuinely parallel at the hardware level; two packet trajectories in different parts of the machine will never interact (and so never get in each other's way).
- Similar to GPGPU systems, the POETS hardware architecture contains many (millions) of computing units.
 - Unlike GPGPUs, these are fully-fledged distributed cores (in our case RISC-V) that scale naturally to large numbers of chips optimised for communication. The communication speed attributes of GPGPUs decay quickly if the topology of the problem does not map cleanly to the (mandatory, fixed) layout of the processing capabilities, and does not scale well to more than a few boards anyway. POETS is largely unconcerned about the packet traffic pattern.



POETS is ideally suited for compute problems that can be addressed by decomposing the problem domain into a large (not necessarily regular) mesh of nodes, and allowing these nodes to communicate state changes *asynchronously* with their neighbours²³.

2.3 How does it work ?

The POETS message-passing architecture is subtly different from other message based systems. 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** (aka **events**): examples being neural simulations and state changes in finite element methods - other examples are discussed later. The application graph can represent the physical topology of the system under simulation (in which case the topology is arbitrary) or we can tile some (arbitrary dimension) space, and use each device to represent the behaviour of the system in that particular unit space cell.

- ▶ Tiling three-dimensional space with conventional cubes allows each cube to interact with its (26) nearest neighbours. The application graph in this case is thus highly regular (degree 26), but this is irrelevant to the functioning of the system.

Each device maintains a small, independent **state**. 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 **events**. An event is delivered to a device (by the EPIC hardware infrastructure); the appropriate handler is awoken by the arrival of the event, and it executes. During the course of this 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 asynchronous and event-driven.*

*A POETS compute trajectory exploits the ability of a large number of processors to **self-organise**, via a self-timed **packet storm**, such that the desired physical solution appears as an **emergent property** of the system once it has returned to quiescence.*

2.4 The packet storm

Events triggering handlers which (may) emit subsequent events can lead quickly to a **packet storm**. This is not as dangerous as it sounds: the network (hardware) contains a 'self-throttling' mechanism that prevents it becoming overloaded. The effect of this is that it goes as fast as it can, all the time - it's just that "as fast as it can" may vary, depending on congestion. Figure 2.4.a shows the packet flux over the compute fabric during the course of a computation. (More details of the solution of a first-order differential heat equation are provided later.) In outline, the system-under-simulation (here a rectangular grid) is initialised with random temperature values. In general, obviously, each grid point will not then be in thermal equilibrium with its neighbours. It sends the value of its temperature to its neighbours, each of which will adjust its own temperature to arrive at equilibrium with (what it perceives to be) the value of its surroundings. Then - if it *is* a new temperature - it broadcasts this new value to all of *its* neighbours.

² A.D. Brown "Event-driven computing" keynote talk, ASAP 2016, Imperial College London

³ A.D. Brown, D.B. Thomas, J.S. Reeve, G. Tarawneh, A. de Gennaro, A.A. Mokhov, M. Naylor and T.J. Kazmierski, "Distributed event-based computing", International conference on parallel computing, ParCo'17, Bologna, September 2017.

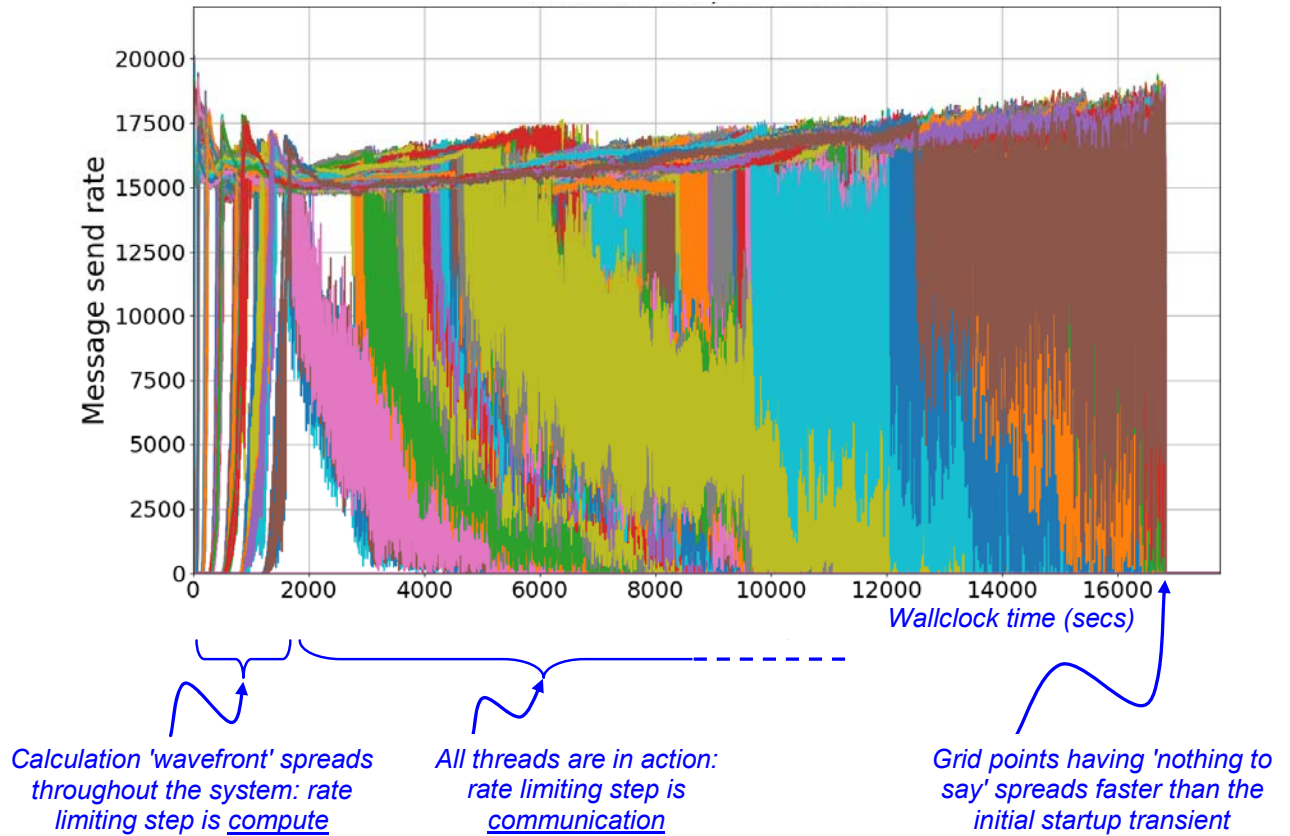


Figure 2.4.a: Canonical heat equation solution trajectories.

All the grid points are acting simultaneously, and the result is that, very quickly, a *packet storm* arises, with each grid point reacting *as soon as it can* to information about temperature changes from its neighbours.

As abruptly as it starts, the grid of temperature values discovers (and *on the basis of local information alone*) that it is in thermal equilibrium, and so all packet traffic ceases.

Qualitatively, a 'computation wavefront' ripples through the fabric. It is possible (although not obvious from figure 2.4.a) to identify 'computational reflections' in the activity of the system.

*The POETS perspective to this is simply to let everything go as fast as it can (subject to maintaining causality), with the assumption that the cost of compute that is either duplication and/or based on stale data is **less** than the cost of explicitly managing the compute trajectory so that all compute is useful - cores are free.*

2.5 How do we know this is a good idea ?

In any complex computational system, at any point in time, there will be a rate limiting step. There must be at least one, or the system would go faster. The notion that *two* (or more) steps might be equally limiting is fragile, at best. Much effort in machine architecture design goes into pre-emptively identifying potential bottlenecks and accelerating (or avoiding) them.

We have demonstrated the (almost) constant scaling behaviour outlined above in a variety of application domains - quantitative behaviour is presented in sections 6 and 7.



3. Physical hardware

At the previous Advisory Board, we had completed construction of our first-generation POETS cluster of eight 4U boxes, each holding seven Stratix-V FPGAs and a modern x86 CPU⁴. These Stratix-V FPGAs had been purchased on a previous EPSRC funded project and were reused to provide an early hardware target for software built by the rest of the consortium.

3.1 Outline

One FPGA in each box serves as a bridge between the x86 (standard PCIe) and the remaining six worker FPGAs (10G Ethernet). Worker FPGAs are connected together via 10G Ethernet (four 10G ports per FPGA) and custom PCIe backplanes (two ports of eight lanes), resulting in a 3D topology. The boxes occupy a single 40U rack in University of Cambridge Department of Computer Science and Technology's machine room. This system



is accessible over the Internet (SSH). The setup is quite reliable, except for the occasional replacement of an FPGA fan (we've done around four fan replacements in total over the past three years). FPGA temperatures and fan health are all monitored continuously, with automated emails being sent when an issue is detected. This hardware has been widely used by the project team and has provided results for many publications.

Figure 3.1.a: First generation Stratix-V based server (stack of two server boxes shown).

Since then, we have gone beyond the original grant proposal to develop a second-generation POETS cluster, consisting of ten 4U boxes, but with each server box improved in the following ways. Now there are eight Stratix-10 FPGAs per box, each of which has a dedicated PCIe connection to the host x86 CPU, eliminating the need for a bridge FPGA. FPGA fans have been replaced

with heat sinks, relying solely on heavy duty fans within the 4U case for active cooling. Compared to the Stratix-V FPGAs, the Stratix 10s contain:

- 4x more general-purpose logic resources;
- thousands of hard floating-point units;
- 10x higher inter-FPGA bandwidth (4x 100G Ethernet ports per FPGA);
- 4x DRAM bandwidth (two extra DIMMs and DDR4 rather than DDR3).

Later in this report, we present early results from this second-generation POETS cluster. Due to COVID-19 impacting work on the second-generation system and having a highly detrimental effect on the supply chain, we were unable to complete the validation and refinement process needed before the system could be deployed to the whole project team.

⁴ M. Naylor, S. W. Moore and D. Thomas, "Tinsel: A Manythread Overlay for FPGA Clusters," *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 2019, pp. 375-383, doi: 10.1109/FPL.2019.00066.



3.2 Component delays and COVID-19

The Stratix-10 FPGAs were announced before we started the POETS project, but Intel (the manufacturer) had major issues manufacturing these parts and major customers like Microsoft Azure received parts first. One solution would have been to have purchased the previous generation of FPGA: the Stratix-V, which were available. However, we already had a set of Stratix-V FPGAs from an earlier project to it made sense to reuse these for a first-generation system and seek to purchase Stratix-10s for a second-generation system. If we had not had these Stratix-V FPGAs in stock, we would have purchased them on this grant.

We were only able to secure Stratix-10 FPGA boards by May 2021, much later than anticipated and during the pandemic when we were unable to get physical access. We also suffered a four-month delay obtaining the base server boxes to put the FPGAs in.

The DE10Pro card containing the Stratix-10 FPGAs needs an I/O expansion card to get full connectivity (Ethernet, USB, etc.). The expansion card available from the DE10Pro manufacturer was the wrong form factor to fit in a server case. Fortunately, we did have access to an early version of the Stratix-10 FPGA board which allowed us to make progress on our version of the I/O expansion card. This work was conducted by RA Tom Bytheway who worked part-time on this project. We went through the following revisions:

- 2019-12-16 - Initial board layout in original horizontal USB-C configuration.
- 2020-01-17 - First revision of the flex connector.
- 2020-07-29 - Test flex PCB to connect original Terasic HPS board via flex.
- 2020-09-10 - Revised test flex with additional stiffeners.
- 2020-09-23 - Switched to first vertical layout test with an on-board connection to the FPGA board deprecating the initial flex PCB based design.
- 2020-11-17 - First fully revised for vertical PCB layout with final arraignment of onboard connectors for IO.
- 2020-12-04 - Revised board with termination points on all ULPI lines to allow for impedance adjustment.
- 2021-04-12 - Side experiment in HPS board with USB eliminated and only ethernet and UART for reduced part count and PCB size.
- 2021-04-12 - First six-layer board with USB and Ethernet Phy chips reversed in position to allow for extremely fastidious ULPI and RGMII routing. **This board was fully functional.** This version of the board was ready when the large batch of Stratix-10 boards arrived, but we could no longer source many of the parts used.

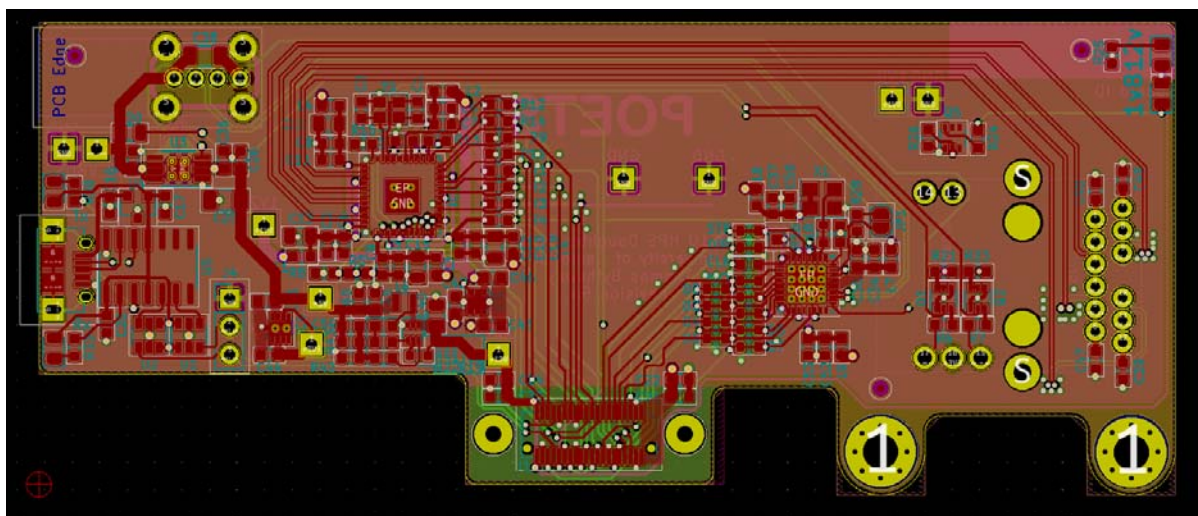


Figure 3.2.a: Final I/O expansion PCB for the Stratix-10 DE10Pro board.



- 2021-10-14 – We had to swap to alternative parts and redesign the board to mitigate the semiconductor shortage and supply issues.

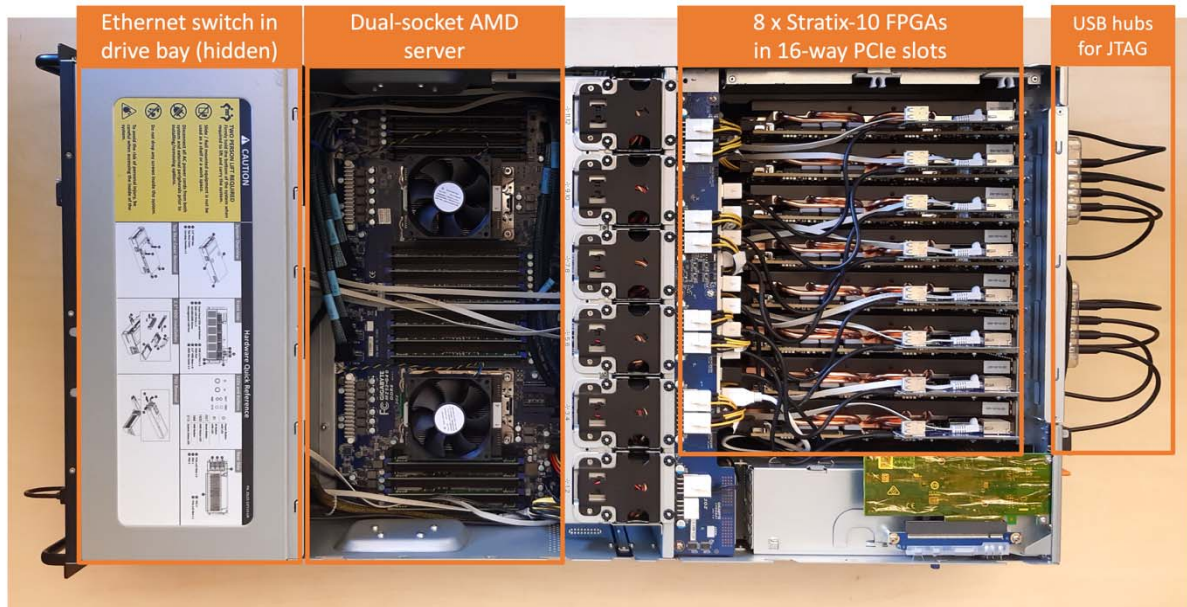


Figure 3.2.b: Stratix-10 server build, showing the 8 installed DE10Pro cards and the I/O expansion PCBs.

3.3 Tinsel overlay

At the previous Advisory Board, we had a manythread RISC-V overlay (Tinsel), specially designed for the POETS project, running on the first-generation POETS hardware. In an 8-box configuration, it provided:

- 3,072 RISC-V (RV32IMF) cores (49,152 hardware threads), clocking over 200MHz.
- Inter-core communication network combining a 2D network-on-chip within each FPGA with reliable links between FPGAs (reliability is achieved using a custom protocol on top of Ethernet).
- Architectural support for message-passing between hardware threads, with guaranteed delivery.
- Cached access to 96x 2GB DDR3 DRAMs and 192x 8MB QDRII+ SRAMs (around 2TB/s of memory bandwidth in total).
- Single-precision floating point support.
- HostLink API for command-and-control of the Tinsel overlay from each x86 CPU over PCI Express.
- A novel hardware-accelerated synchronisation barrier primitive, based on Safra's termination detection algorithm, which is ideally suited to fine-grained message-passing systems.
- Hardware support for "local" multicasting, reducing network bandwidth significantly in the case where a message needs to be sent to multiple colocated destinations (i.e. destinations that are located close to each other on the NoC). This proved very beneficial to the POETS dissipative particle dynamics implementation, enabling excellent scaling behaviour⁵.

⁵ M. Naylor *et al.*, "General hardware multicasting for fine-grained message-passing architectures," *2021 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2021, pp. 126-133, doi: 10.1109/PDP52278.2021.00028.

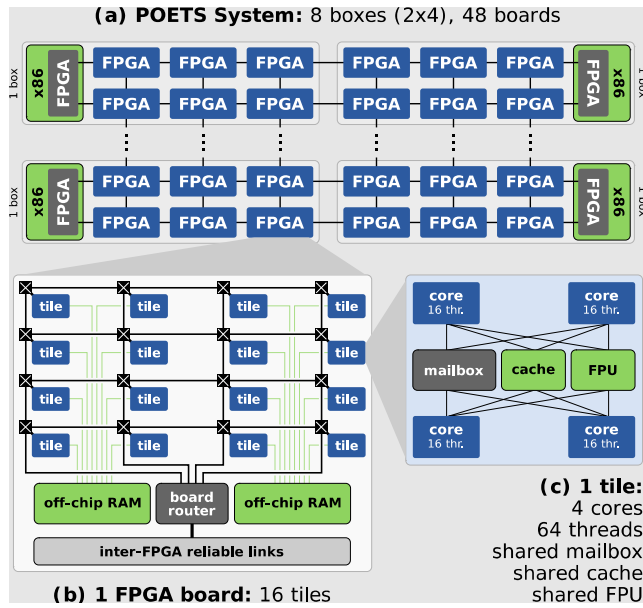


Figure 3.3.a: High-level architecture of the Tinsel overlay.

We published a paper about Tinsel at FPL 2019⁴. The paper compared distributed vertex-centric graph processing systems, one running on a Xeon cluster and the other running on our POETS cluster. Results were favourable for the POETS approach.

Since then, we have published a paper about our synchronisation barrier primitive to ASAP 2020⁶. The paper explains the semantic and efficiency advantages of the primitive compared to conventional MPI barriers and demonstrates how to implement it efficiently in hardware.

We have also extended our work on hardware multicasting to support distributed destinations as well as colocated destinations, publishing a paper about it at PDP 2021. The paper explains how we support general multicasting in POETS, with guaranteed and precise delivery of messages to any number of arbitrary destinations, fully offloadable to hardware.

With these developments, Tinsel reached its main functionality targets for the POETS project and has been operating unmodified on the first-generation hardware for the past two years.

⁶ M. Naylor *et al.*, "Termination detection for fine-grained message-passing architectures," *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2020, pp. 17-24, doi: 10.1109/ASAP49362.2020.00012.

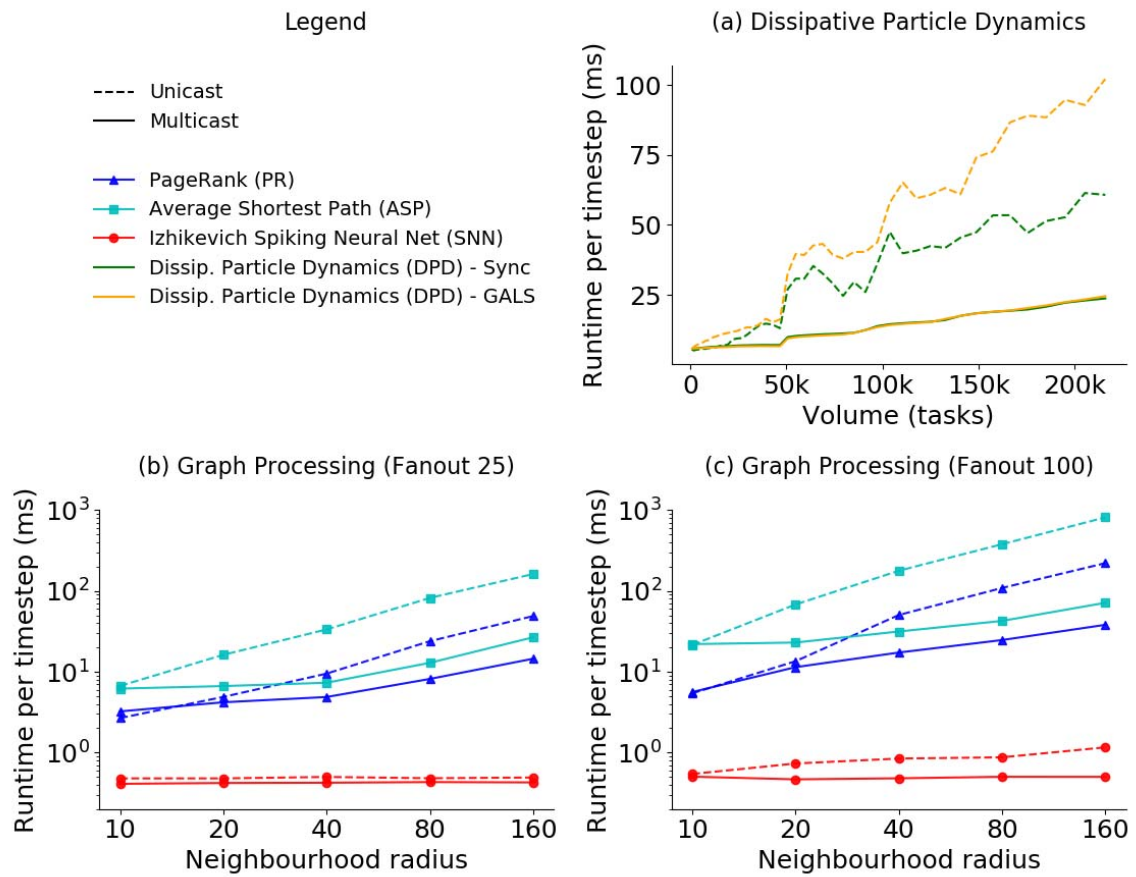


Figure 3.3.b: The performance benefits of general hardware multicasting.

3.4 Tinsel v2.0 for Stratix 10

Matt Naylor designed the original Tinsel overlay for Stratix-V FPGAs. Having completed this exciting and highly successful project, Matt wanted to change research direction and moved to a different research project. We hired a replacement RA who, after 9 months, failed to pass his probationary period. We then hired Coral Westoby, who began work on the Tinsel port and refinement for Stratix 10.

The second-generation POETS cluster is constructed using the same architecture but using higher performance components throughout. The DE10 cluster uses a significantly larger number of newer and more capable FPGA boards, with a much faster network and is hosted by high-performance host CPUs.

	Gen 1 (Stratix-V)	Gen 2 (Stratix-10)	Relative change
FPGA Cluster size	48	64	4/3x
Network connectivity	2d square grid	2d Torus	Reduced worst-case path length
Networking bandwidth	10GbE	100GbE	10x
FPGA Fabric Size	622k	2753k	4.4x
FPGA Fabric Frequency	215MHz	200-300Mhz	
FPGA DSP Slices	256	5760	22.5x
FPGA On-chip Memory	57Mbit	244Mbit	4.2x
SRAM	8Mbyte	N/A	
DRAM Capacity Total	4Gb	32Gb	4x
DRAM Bandwidth	2x DDR3	4x DDR4	4x



For each tinsel core, the DE10's provide 4.4x as many logic elements and 4.2x the on-chip memories, used for the individual processing cores and caches within Tinsel, allowing the design for the manycore fabric to remain relatively unchanged. In contrast, the Stratix 10 has over 5x the number of DSP slices per logic element. This allows for greater floating-point throughput per core and presents a different optimisation target for Tinsel.

The 100GbE networking offers much greater bandwidth, at a comparable latency to the DE5 cluster. However, the data paths run at both slightly higher clock rates and at much wider widths and developing a network of cores capable of using this bandwidth is challenging. In order to determine the optimal design target for the DE10's, microbenchmarks to explore the design space were used. We primarily looked at the relationship of Integer and Floating-point execution unit count and design clock frequency.

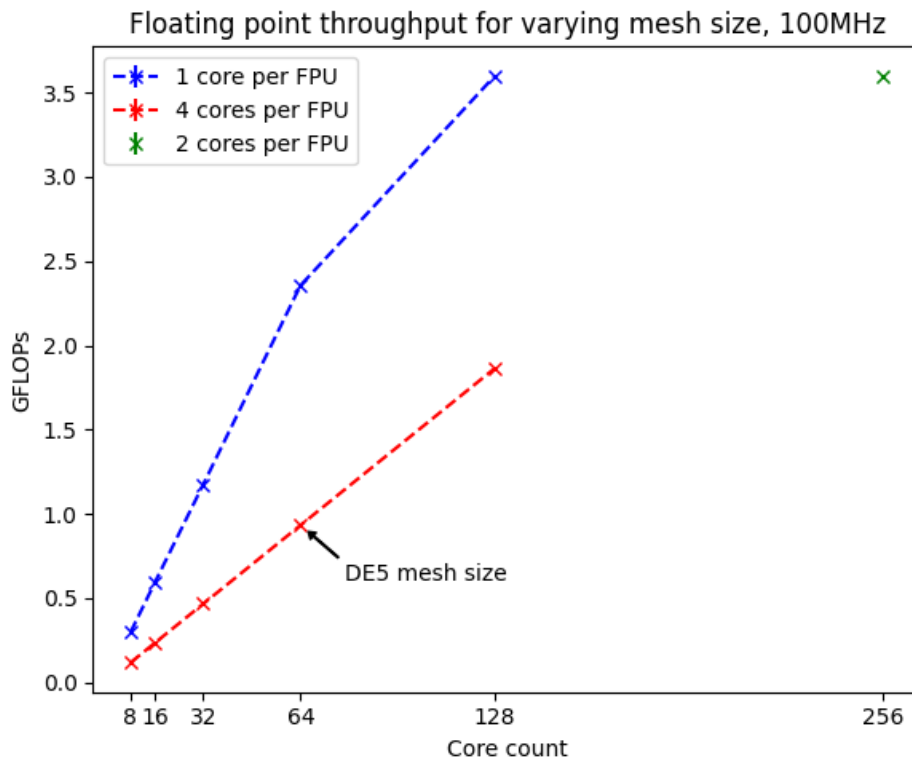


Figure 3.4.b: Floating point throughput for a single DE10 board at a fixed fabric frequency of 100MHz. Error bars are smaller than the plot markers. Dashed lines link points for ease of viewing and do not indicate intermediate results. The Tinsel design shows good scaling for single-board floating point throughput to designs much larger than the original target.

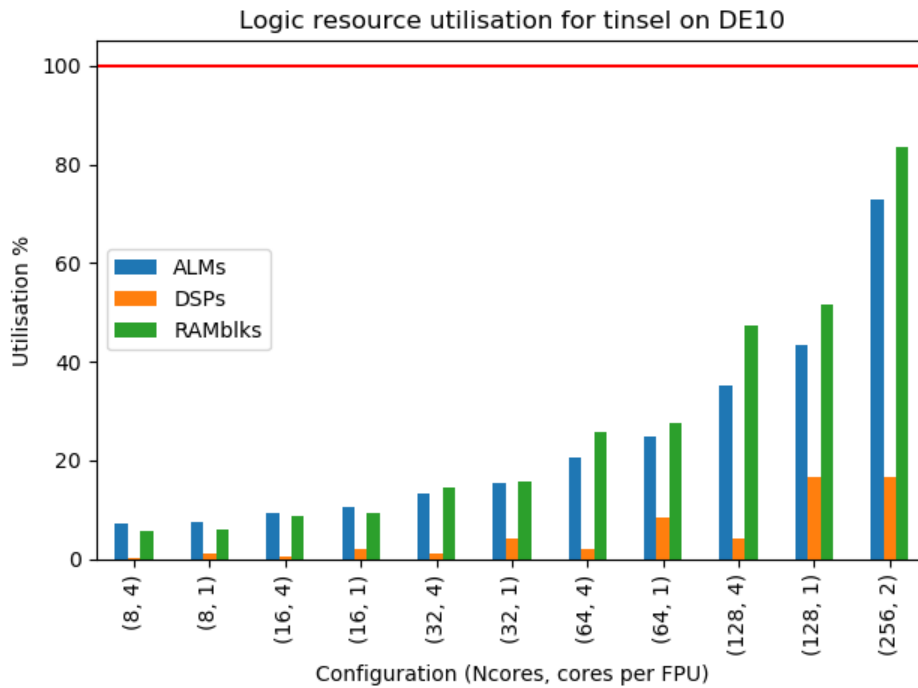


Figure 3.4.c: Logic utilisation for a range of tinsel designs on the DE10. The design (64, 4) is the largest design that fits on the DE5.

In a single board comparison, the DE10 offers a significant uplift in FP performance. Tinsel is a highly parameterizable design and providing a dedicated FP unit per core provides a proportional improvement in throughput on a microbenchmark.

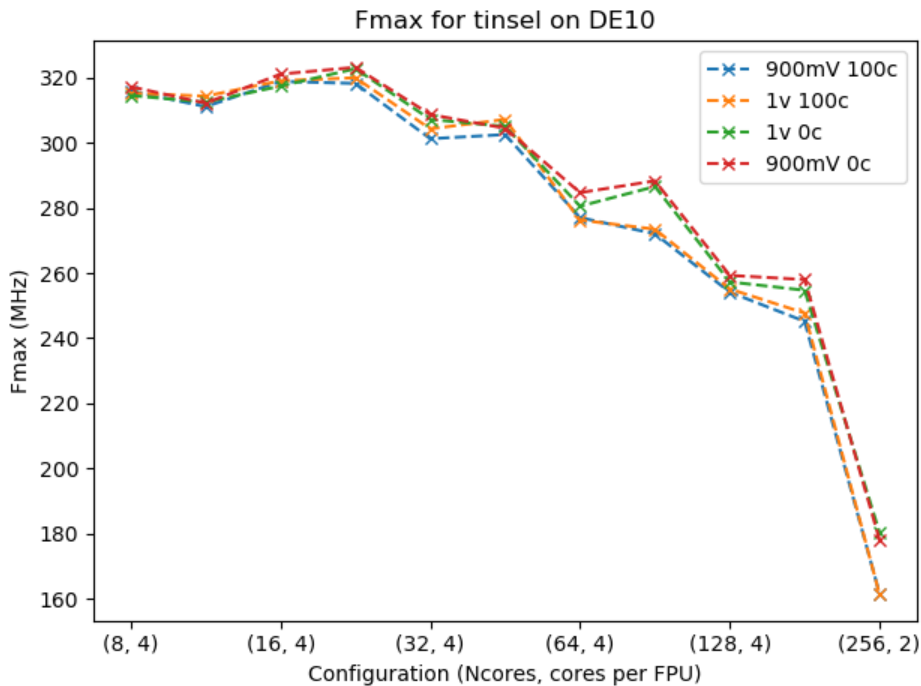


Figure 3.4.d: Achievable performance for the varying mesh sizes for Tinsel on the DE10. Dashed lines link points for ease of viewing and do not indicate intermediate results.

For the largest designs, Fmax is limited by very high fanout nets from each mailbox to the core cluster. Parameterisation limitations prevented us from expanding the mailbox



network size from the DE5 baseline.

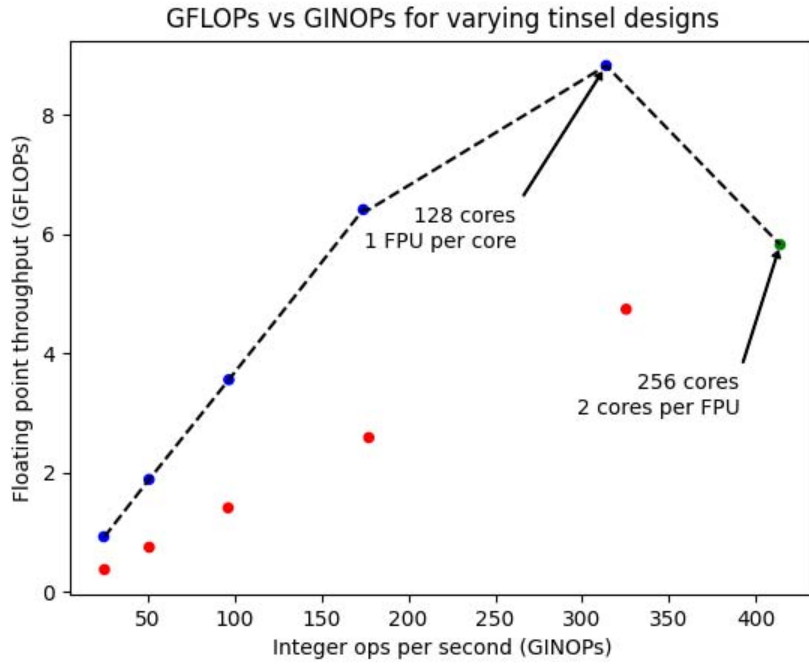


Figure 3.4.e: Pareto frontier for varying configurations of Tinsel on the DE10. The highest performing design for floating point is the 128 core, 1 core per FPU configuration. Adding additional cores improves theoretical Integer throughput, but the reduction in Fmax significantly harms Floating point.

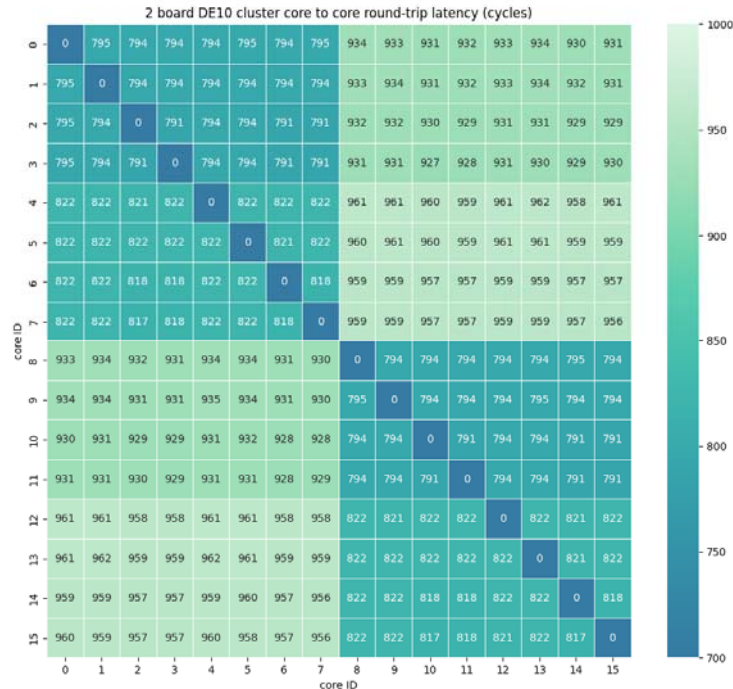


Figure 3.4.f: Core to Core benchmark of messaging latency. Measurements were performed on a 2 board DE10 cluster using an 8-core tinsel configuration, using 2 mailboxes per board. Boards were interconnected with a point-to-point 100G Ethernet connection.



3.5 Custom enhancements

The Tinsel overlay is based on the RISC-V core. To improve performance, we have developed various optimisations, including custom enhancement to Tinsel such as custom instructions, custom accelerators and custom vectorisation, and the Simodence softcore for exploring reconfigurable SIMD instructions.

Tinsel cores achieve good performance on multiple applications, but they process data in fixed widths (e.g. 32 bits) with a fixed instruction set, and can be beaten on computation-intensive applications by conventional FPGA datapath designs without using instructions.

To address this, we develop three kinds of custom enhancements to Tinsel: *custom accelerators*, *custom instructions*, and *custom vectorisation*. Custom accelerators are pipelines or state machines that attach to the Tinsel network, implementing a fixed function. Custom instructions are added to one or more Tinsel processors, using RISC-V instruction extensions. They have complementary strengths: instructions have a low, fixed latency, while accelerators have variable latency due to network communications. Furthermore, custom vectorisation involves new custom vector instructions to improve parallelism. Finally, beyond Tinsel, we have explored **reconfigurable SIMD instructions** based on the Simodence, a new RISC-V core we developed for this research.

Custom instructions. They extend the Tinsel instruction set (RV32IMF) with domain-specific instructions. We examine both integer and floating-point instructions: integer instructions take one cycle, and involve customising the Tinsel pipeline in four places; floating-point instructions use the Tinsel FPU and hence can take more than one cycle; they involve customising the Tinsel pipeline in 12 places. We customize the RISC-V GNU toolchain to call custom instructions using compiler intrinsics.

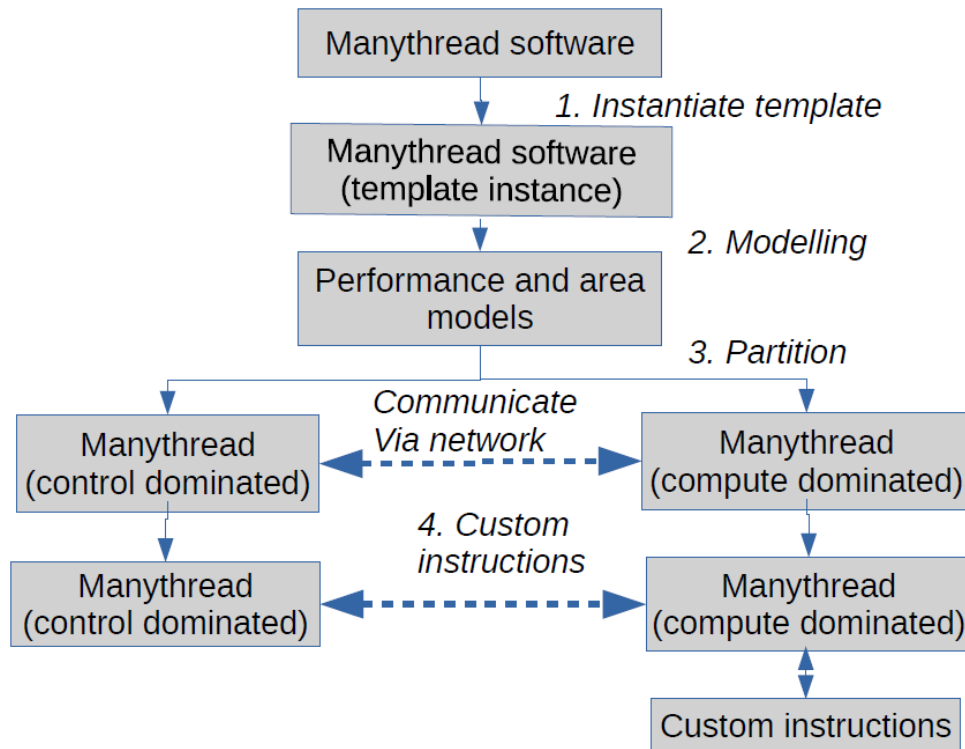


Figure 3.5.a: Design flow for custom instructions.

For a given FPGA, since custom instructions take extra area that could be used for additional Tinsel processors, we provide an analysis of the area-performance tradeoff for both throughput and latency-dominated applications, and a methodology to choose between



custom instructions. Our published work shows speedups for several applications including DPD (dissipative particle dynamics): a custom floating-point square root saves 95 cycles per call, giving about 16% speedup overall.

Figure 3.5.a shows our four-step approach to improving performance using custom instructions, starting from manythread software (i.e. a Tinsel application). We use straightforward but indicative performance and area models to evaluate different custom instruction candidates.

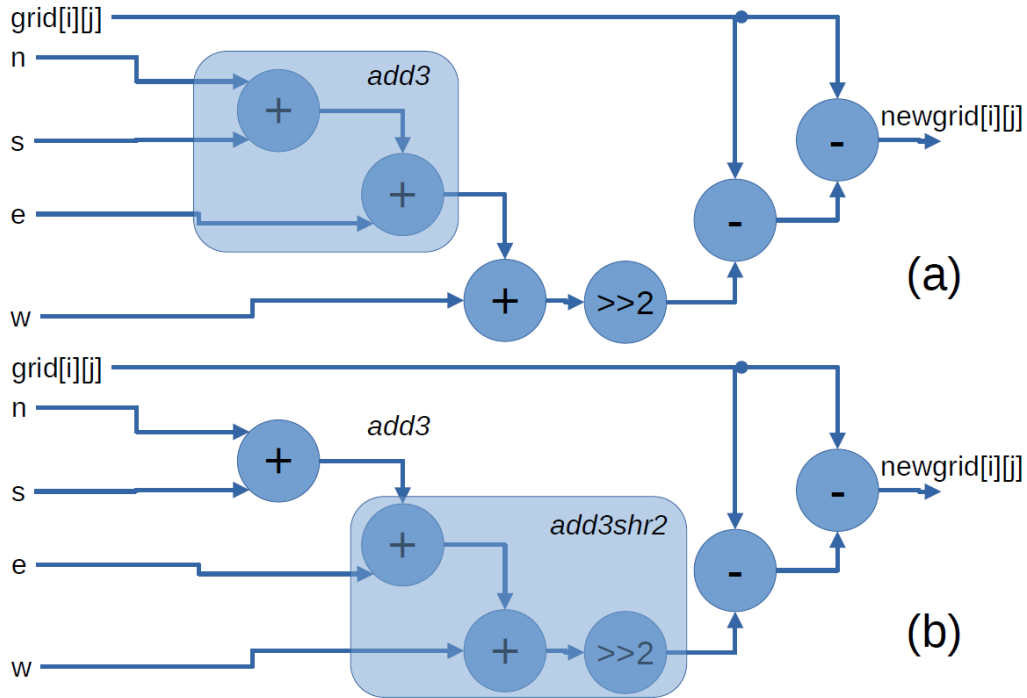


Figure 3.5.b: Two custom instruction candidates for the Heatplate application: (a) a three-input add; (b) three input add and shift.

As an example, Figure 3.5.b shows two different custom instruction candidates for the Heatplate application, which performs core temperature calculation. Option (b) saves one cycle over option (a) by incorporating the shift into the custom instruction. In this example the instructions are limited to three inputs; we have implemented up to six input instructions within Tinsel.

The table below shows cycle counts and number of processors that can be instantiated, both before and after custom instructions are introduced. In each case the gain (reduced cycle count) exceeds the cost (reduced number of processors), so there is a net gain.



Benchmark	Cycles		Gain	Processors		Cost	Net Gain?
	C	C'	C/C'	N	N'	N/N'	$\frac{C}{C'} > \frac{N}{N'}$
Nbody	479	381	1.26	16	15	1.07	Yes
Heatplate	704	576	1.22	16	16	1.0	Yes
DPD	748	647	1.16	16	15	1.07	Yes
Dot Product	10	9	1.11	16	16	1.0	Yes
IAF	79	68	1.17	16	14	1.14	Yes
Pagerank	49	38	1.29	16	14	1.14	Yes

Cycle counts before (C) and after (C') custom instructions, with cost in terms of number of processors that can be instantiated without (N) and with (N') custom instructions, for applications including gravitational N-body (Nbody) and neural network (IAF).

Custom accelerators. These accelerators are network-attached. They input and output network flits: a flit is a subfield of a packet. This decouples them from the Tinsel cores and mailboxes, simplifying their design. Custom accelerators are currently written in SystemVerilog, but can be written in any language that compiles to that: some high-level synthesis tools can compile from C or OpenCL to this design, thus enabling the use of custom accelerators by users without hardware design skills.

We consider both stateful and stateless custom accelerators. **Stateful accelerators** replace a Tinsel application with the custom hardware equivalent, while **stateless accelerators** attach pipelines with no user-visible state to the network. We adapt the analysis for custom instructions to choose between different custom accelerator options, and use a similar design flow to accelerate manycore Tinsel applications with custom accelerators.

Since custom instructions and accelerators are complementary approaches to enhancing Tinsel performance, we develop a combined approach to choose between them. Custom instructions are better suited to applications where the custom hardware is frequently used, while custom accelerators may be better suited to applications where custom instructions would go unused on many processors, and the network overhead can be tolerated. We compare the approaches on several applications.

Design-space exploration. We show the potential of these custom enhancements by exploring various possibilities for the DPD (dissipative particle dynamics) application. A large majority of the run time in DPD is taken by the *force_update* function, which calculates the force between two interacting particles, or beads, and updates their positions and velocities accordingly. The function uses a distance check to test whether two beads are sufficiently near to interact; if they are close enough, a long sequence of code calculates the positions and velocities. Figure 3.5.c(a) shows the time taken in instructions and cycles – 15% of interactions use the full calculation, taking an extra 643 cycles, not including the square root. The average time is 239 cycles.

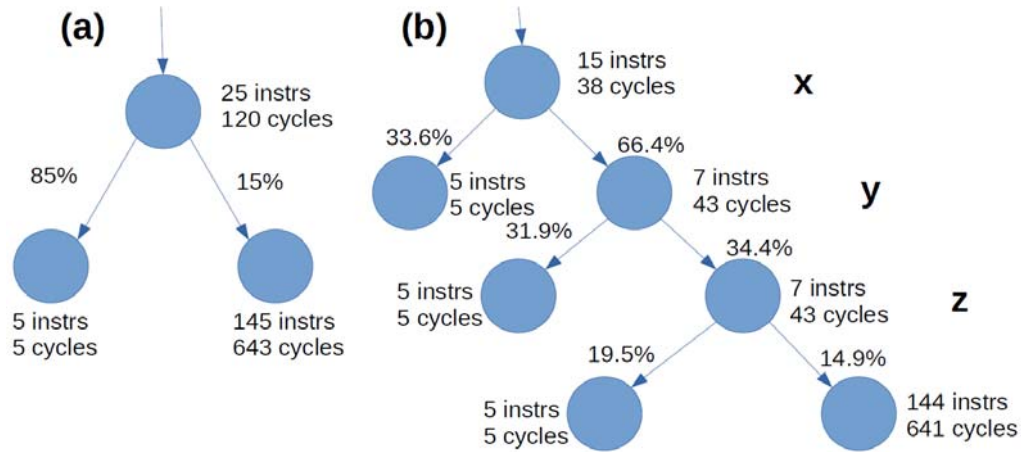


Figure 3.5.c: DPD distance tests. (a) Standard. (b) Separated by dimensions.

We consider the following hardware and software optimizations:

- custom floating-point square root, as above;
- custom upper-bound-square instruction;
- software optimization to separate distance test by dimensions.

Note that all interactions take 120 cycles for the distance test. To reduce this value, we separate the distance test by dimensions by calculating the squared distance in the x dimension, then the y dimension, and repeating the distance test after each dimension as shown in Figure 3.5.c(b). This reduces the average time to 199 cycles. Each dimension uses three floating-point operations to calculate the square of the difference, then accumulate. The square takes 11 cycles on the Stratix-V FPGA, so we develop an approximation to take the upper bound of the runtime of the square calculation, which can be done in 4 cycles in software, or one cycle in a custom instruction. The software takes 174 cycles, while the custom instruction takes 169 cycles. Figure 3.5.d shows the design space exploration with all three optimizations, reducing the average cycle count from 239 to 153, about 1.5 times.

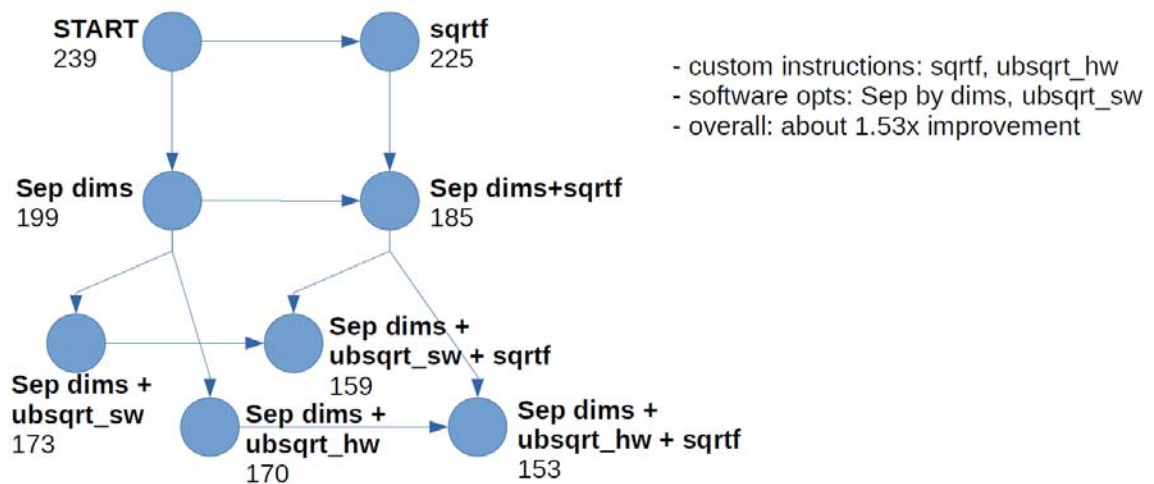


Figure 3.5.d: Design space exploration with two custom instructions (**sqrtf** and **uabsqrt_hw**) and software optimization to separate distance test into dimensions (Sep dims). Overall the average cycle count reduces from 239 to 153.

Custom vectorisation. The DPD application uses floating-point numbers, but it does not make full use of their range due to the limited distances and hence forces and velocities possible. We estimate the possible improvements due to vectorisation using a custom vector



format. Rather than using the standard RISC-V vector instructions which would require significant additional hardware resources, we consider a modified Tinsel design with longer data words: 39 or 48 bits, which would be interpreted as integer 3-vectors with 13 or 16 bits per component, respectively. The small data size per component means vector operations can take one cycle, which would involve customization of Tinsel’s ALU rather than its FPU. If the entire *force_update* calculation is mapped to this custom format, this would take an average of 60 cycles with one cycle per instruction. Compared to the average with floating-point numbers, this is about 4 times improvement – and the other customizations and optimizations could improve it even further. The cost is in extra area for the caches, assuming everything else (networking, mailboxes, FPU) remains 32 bits. Even if fewer Tinsels can be instantiated (say 25-50%), there should still be an overall improvement of 1.5 to 3 times.

Reconfigurable SIMD instructions. To enable further study of core customisation, we have developed Simodense⁷, a high-performance open-source RISC-V (RV32IM) softcore that support exploration of custom SIMD instructions on FPGAs. In order to maximise SIMD instruction performance, the design’s memory system is optimised for streaming bandwidth, such as very wide blocks for the last-level cache. This approach has been implemented in the Ultra96 platform, which features the Xilinx UltraScale+ ZU3EG device running at 150MHz. For custom instructions based on odd-even merge-sort (figure 3.5.e), over 12 times speedup can be achieved when compared with the softcore executing the quicksort library. In particular, the *c2_sort* custom instruction is able to sort a list of eight 32-bit elements in 6 cycles, while a sorting network implementation of only four 32-bit inputs in older Intel processors require 13 SIMD instructions and 26 cycles, resulting in 13 times reduction in instructions and 4.3 times reduction in cycles. For custom instructions based on prefix-sum, over 4 times speedup can be achieved.

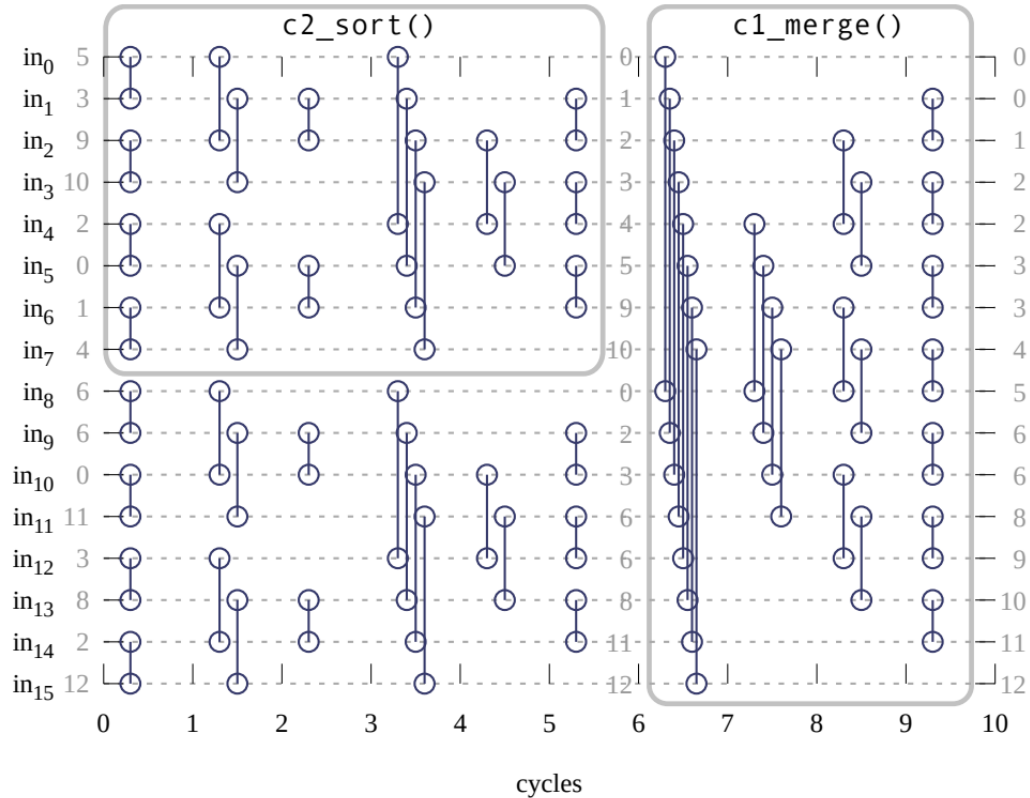


Figure 3.5.e: Two new custom instructions based on odd-even mergesort.

⁷ P. Papaphilippou, K. Paul H. J. and W. Luk, "Simodense: a RISC-V softcore optimised for exploring custom SIMD instructions," 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), 2021, pp. 391-397, doi: 10.1109/FPL53798.2021.00082.



Future work. There are various exciting possibilities for further work. Currently custom instructions, including custom vector and SIMD instructions, and custom accelerators, are largely manually derived. We would like to adapt this work to prior research on automatically deriving custom instructions from programs. There are further possibilities for demanding applications such as DPD to benefit from custom accelerators and instructions, by supporting their reuse in similar application domains. Finally, we have considered a largely uniform architecture with custom enhancements; future work would consider multiple heterogeneous processors with different customisations, and how the customizations can be adapted continuously to changing runtime requirements, such as when boundary layers form in the DPD application.

For further details, please see:

- FPT2020: Exploring performance enhancement of event-driven processor networks⁸
- ISVLSI2021: Custom enhancements to networked processor templates⁹
- FPL2021: Simodense: a RISC-V softcore optimised for exploring custom SIMD instructions⁸
- HEART2022: Non-deterministic event brokered computing¹⁰
- IEEE TCAS 2022: Custom instructions for networked processor templates¹¹
- Submitted for publication: Combining enhancements to event-driven processor networks

3.6 Beyond POETS: using the second-generation hardware

We plan to use the Stratix 10 infrastructure in other projects including the proposed follow-on project *SONNETS: Scalability Oriented Novel Networks of Event Triggered Systems*.

The Cambridge PI (Prof Simon Moore) has been researching secure processors since 2010, and the CHERI technology has become the basis for the Innovate UK Digital Security by Design program. Under this program, Cambridge receives funding directly and also through the following EPSRC subprojects:

- EP/V000292/1 - CHERI for Hypervisors and Operating Systems (CHaOS)
- EP/V000381/1 - CAPcelerate: Capabilities for Heterogeneous Accelerators
- EP/X015963/1 - Chrompartments: Hybrid Compartmentalisation for Web Browsers

We are starting to use the Stratix-10 infrastructure to provide support for many CHERI-RISC-V based systems to facilitate evaluation and benchmarking. We also have a US funded project – CHERI-BGAS – That is exploring CHERI protected partitioned global address spaces (PGAS) and will make use of the Stratix-10 reconfigurable infrastructure. So as with our previous batch of Stratix-V FPGAs, the Stratix-10 FPGAs will enable a great deal of research beyond POETS.

⁸ Tim Todman, David Thomas, Wayne Luk "Exploring performance enhancement of event-driven processor networks" 2020 International Conference on Field-Programmable Technology (ICFPT), doi: 10.1109/ICFPT51103.2020.00056

⁹ Tim Todman, Wayne Luk "Custom enhancements to networked processor templates" 2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), doi: 10.1109/ISVLSI51109.2021.00049

¹⁰ Andrew Brown, Tim Todman, Wayne Luk, David Thomas, Mark Vousden, Graeme Bragg, Wayne Luk, Jonny Beaumont, Tim Todman, Alex Yakovlev, Ashur Rafiev "Non-deterministic event brokered computing" International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies June 2022 Pages 84–86 <https://doi.org/10.1145/3535044.3535055>

¹¹ Tim Todman and Wayne Luk, "Custom Instructions for Networked Processor Templates" IEEE Transactions on circuits and systems: Express briefs, V69, No 7, JULY 2022 doi: 10.1109/TCSII.2022.3178389



4. The software stack

4.1 Introduction

From the project proposal:

*POETS - **Partial Ordered Event Triggered Systems** - technology is based on the idea of an extremely large number of small cores, embedded in a fast, hardware, parallel communications infrastructure - the **application network** communication is effected by small, fixed size, hardware data **packets** (a few bytes).*

The POETS project describes research to investigate and prototype a software methodology and associated hardware platform to realise the potential of this architecture.

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

Problem domains for which POETS is useful are those in which the problem description can be represented as an (arbitrary, abstract) **application graph**. This topology is *independent* of the topology of the underlying hardware POETS engine.

POETS leans heavily on graph theoretic concepts, and there are a lot of similar, albeit different concepts standing side-by-side here.

4.2 Mode of (abstract) operation

The **application graph** is a directed, tripartite graph: {**devices, channels, pins**} - see figure 4.2.a. Associated with each abstract device is (1) some localised, persistent (partial) state vector, and (2) a small set of software **handler routines**. To a zeroth approximation, each device can be modelled as a small, abstract, state machine, which communicates via asynchronous **messages**. Access *to* the state machine is via one or more **receive handlers**, and communication *from* the state machine via one or more **send handlers**. Devices do not have independent threads of control; they *react* to incoming messages and send out consequent messages when the communications infrastructure allows.

When quiescent, they can execute an **OnIdle handler**. This can - in principle - block the device, but the design intention is that any OnIdle activity is brief, so that reaction to incoming messages is not delayed unduly. This is exactly analogous to the OnIdle handlers in the Windows message loop.

Associated with each device is a **supervisor** - these will usually be connected to multiple devices. This is a process running on a conventional processor (with all the infrastructure that that provides), but it can be viewed as a special sort of device. Each device has at least one "supervisor input pin" (and associated handler) and at least one "supervisor output pin", but the actual connections between the device and supervisors are implicit, and instantiated by the Orchestrator. The supervisor(s) provide a meta-graph sitting over the application graph. The supervisors are created by the Orchestrator, using partial definitions provided by the Application.

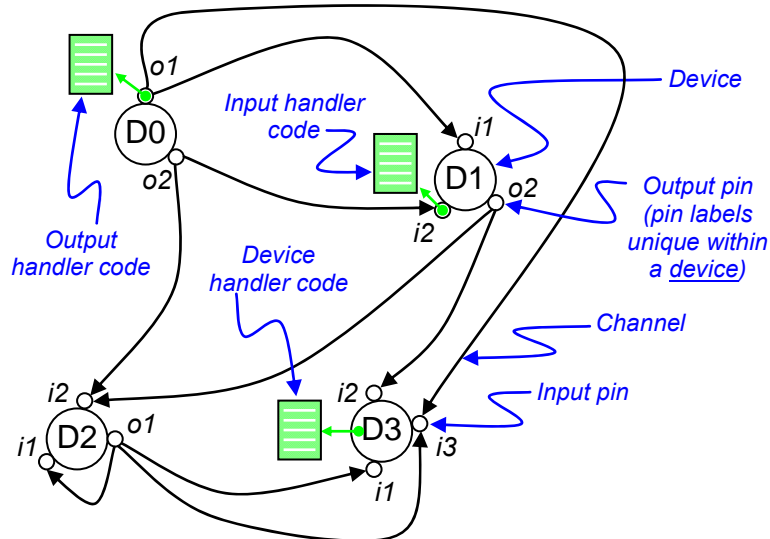


Figure 4.2.a: An abstract application graph (supervisor pins not shown).

The supervisor pins - by necessity - also have a handler, nominally provided by the Application. *Default* supervisor functionality is provided by the Orchestrator, but as this (the Orchestrator) has no domain-specific knowledge, the default functionality is by necessity extremely limited. It is invoked principally during setup and data exfiltration. It provides an asynchronous mechanism whereby the overseeing (conventional) processes can inject (and force the extraction of) information to/from the application graph. They can also be used to supply real time (external application) excitation signals to devices.

Communication between devices is effected by the transmission of *packets* - small (~64 byte) quanta of data, carried along the application graph edges (**channels**) by the underlying hardware: computation is achieved by means of a "self-organising" packet storm. Leaving aside for the moment the issues of how the process starts/stops, the coarse sequence of operations is

- Initially, every device is quiescent. However, any device can indicate whether it currently wants to send packets from any of its output pins, so some devices may start-up ready and willing to send a packet.
- A packet arriving at a device is fielded by the appropriate receive handler; the handler executes, and in doing so may modify the device state and optionally change the set of output pins on which it would like to send.
 - ▶ Devices cannot choose to block or delay the receipt of a message. However, only one handler (per device) will be active at any one time.
- When there is spare (hardware) network capacity, a device which wishes to send a packet will be given the opportunity to send. When the send handler executes, the device can *further* update its state and either: (1) prepare a single packet to be sent, or (2) cancel a request to send.
 - ▶ If a device needs to send multiple packets from a pin, it can leave the request to send on that pin enabled. Eventually spare network capacity will become available



again and the send handler will execute again, allowing multiple packets to be sent as necessary.

- ▶ Packet *launch* may (temporarily) block if the underlying network or target device is congested. As with any asynchronous network, a situation can always be contrived whereby packets are injected into the communication fabric faster than they are drained from it, thereby overwhelming the physical network. An important design issue is what might be done about this state of affairs? Dropping packets at the physical point of local congestion is easy to do, but brings the inherent drawback that neither the sender nor the (intended) recipient can know there is a problem. Any attempt to detect this and transmit the occurrence data anywhere simply adds to the network congestion. In POETS, the problem is pushed back (by the hardware) to the point best qualified to react sensibly to it: the message injection point (sender). If a packet wants/needs (algorithmically) to be sent, and the physical network cannot support the transmission, the putative sending device is informed and still has control, and can decide what to do (abandon the attempt, delay the attempt, or delay and try to send a modified packet later on). If there is currently no local message capacity, then the send handler will not get called, so any packets remain implicitly encoded in the device state - whilst a device might *want* to send a packet, it is not *allowed* to prepare and inject that packet until network capacity is available and delivery can be guaranteed.
- ▶ Once launched, both *unicast* and *multicast* packet delivery to devices in the application graph is *guaranteed*.
- ▶ Wallclock packet transit latency is non-deterministic and non-transitive.
- ▶ Any notion of simulated temporal fidelity must be carried explicitly by the packet payload. (Although wallclock time is available to the physical cores, this cannot possess the dynamic range of accuracy made possible by specifying it explicitly.)
- When a send or receive handler terminates (the design intention is that the execution will be "brief"), the device returns to quiescence, awaiting the arrival of subsequent packets (or the opportunity to send more packets).
- A device may send itself packets.
- Computation ends when (1) the packet storm ends, or (2) some higher-order injected command causes termination.
 - ▶ The solution may be encoded within the set of final (modified) device states distributed over the application graph, or may be output dynamically during execution as devices send messages to supervisor nodes.
 - ▶ Solution exfiltration (output) from the application graph is carried out by cooperation between the devices and the optionally application defined epilogue code within the Supervisors. The Supervisors assemble the distributed solution into some coherent form and hand it out to the Application.
- Application input is provided statically through the topology and configuration of the graph instance, and can also be injected dynamically in real time via the supervisor pins during execution.



The design intent is that handlers only achieve computation through the sending and receiving of packets. Devices do not have an independent thread of control, and only have the ability to compute or send packets when explicitly given access to network and compute resources - they *react* to packets arriving or the opportunity to send packets. There is no deterministic dialogue (although a deterministic conversation can always be forcibly overlaid within the constraints outlined here) because the packet choreography is non-deterministic.

In principle, there are no constraints on the structure of the application graph: it may contain an arbitrary number of devices, interconnected by an arbitrary number of channels. In practise there are hard limits to a few aspects of the graph - outlined later - but these are intended to be sufficiently remote that they will not impinge on the abstract principles of operation.

The concept of "design intent" is used several times in this document. POETS is intended to be used in a certain way, with the components having certain (relative) properties. It is possible - easy - to take POETS out of the region of design intent, and the system behaviour will degrade significantly as a consequence. Currently, there is no intrinsic defence against this - the domain expert user must be aware of the limitations of the system to get any kind of performance from it.

4.3 System configuration

The architectural hierarchy diagram we saw earlier in figure 3.3.a has a number of parameters, which we list in Table 4.1. For ease of understanding we have re-drawn the top levels in figure 4.3.2.a.

Each box is interconnected via an MPI spine, which also connects to a number of other subsystems, as shown in figure 4.3.2.b.

The configuration subsystem is known as the Orchestrator, which contains two principal components: **mapping** and **composing**.

4.3.1 The Orchestrator mapper

Broadly, the configuration subsystem (the Orchestrator) maps the abstract application graph (figure 4.2.a) onto the physical compute mesh (figure 4.3.2.a). The behaviour of this component is outlined in figure 4.3.2.c. Internally, it works on the same principles of an EDA place and route system: a variety of control parameters allow the placement to be evenly spread, packed tightly, configured such that the channels are uniform, and so on. The algorithms used are the traditional Kernighan-Lee and simulated annealing.



Layer	Contains
Device	<ul style="list-style-type: none"> • <1024 handlers/thread • Code serialization at the lowest stack level
Softswitch	<ul style="list-style-type: none"> • 1 Softswitch/thread
Core	<ul style="list-style-type: none"> • 16 threads/core • Local instruction memory • Thread-thread packet delay ~ 230 ns
Tile	<ul style="list-style-type: none"> • 64 threads; 4 cores • 4 port shared mailbox • Shared cache; Shared FPU
Board	<ul style="list-style-type: none"> • 16 tiles - intraconnected via mailboxes • Inter board connections via 4 concentrators • 64 cores; 1024 threads • Inter-board links via NoC • 4GB off-chip RAM • Mailbox-mailbox packet delay ~ 5 ns
Box	<ul style="list-style-type: none"> • 6 boards; 3x2 mesh • 384 cores; 6144 threads • 1 x86 mothership - MPI spine • Board-board packet delay ~ 770 ns
System	<ul style="list-style-type: none"> • 8 boxes; 2x4 mesh • 48 boards; 3072 cores; 49152 threads • 8 motherships • Effective max 50331648 devices • MPI connected configuration subsystem

Table 4.1: Hardware system parameters

4.3.2 The Orchestrator Composer

Alongside this, the composer generates the binary images that are executed by the core threads of figure 3.3.a. Recall that the application specifies only code *fragments* that are to be executed when events (packets) arrived/leave a device. These need to be assembled into piece of code that can be cross-compiled (under the aegis of the Orchestrator), downloaded to the hardware, and executed.

The Orchestrator supplies a software skeleton (known as the Softswitch) which it populates with the code fragments from the application definition, to produce an internally consistent binary image that can be loaded into a RISC-V thread (figure 4.3.2.d).

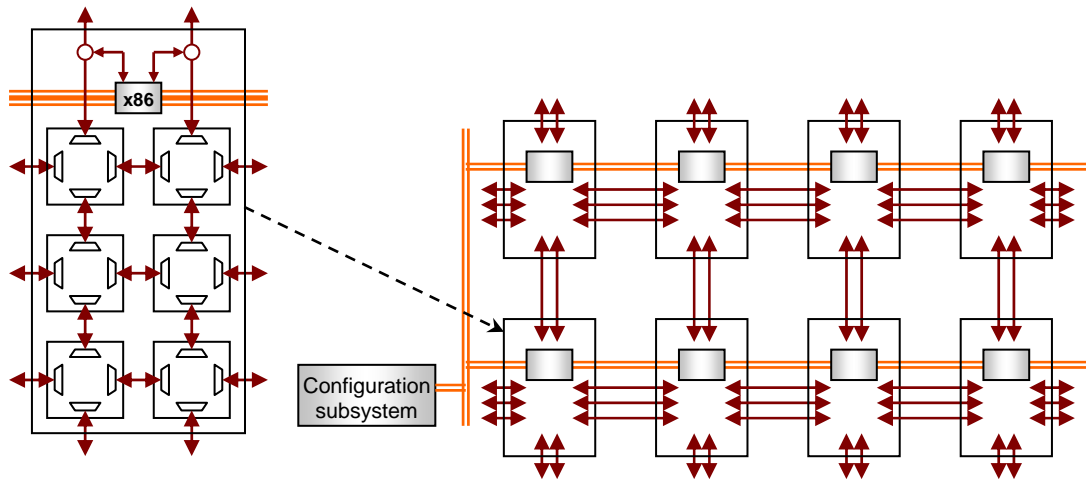


Figure 4.3.2.a: The hierarchy of the POETS hardware.

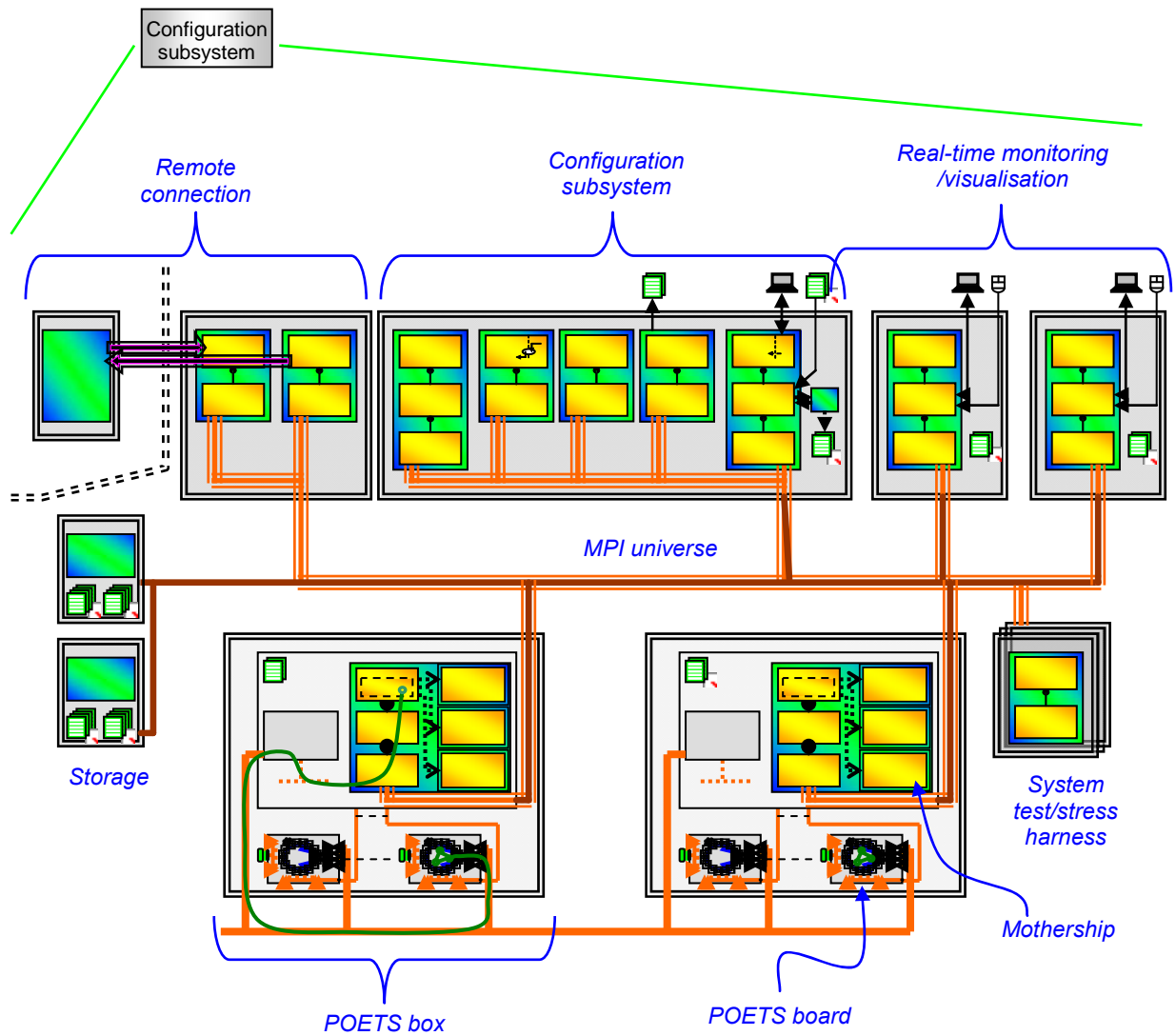


Figure 4.3.2.b: POETS infrastructure support - the Orchestrator.

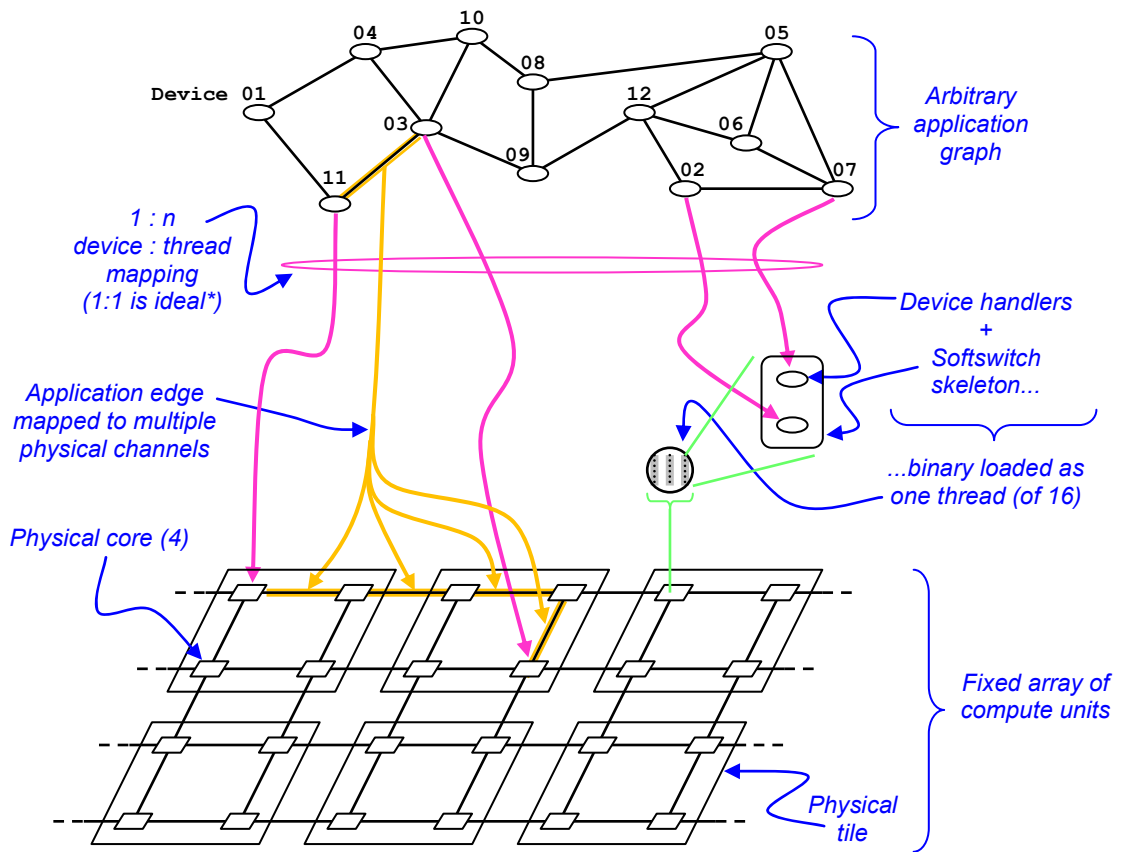


Figure 4.3.2.c: The arbitrary device application graph is mapped to the regular underlying POETS hardware.

* Sometimes multiplexing is useful for latency hiding¹².

¹² Mark Vousden, Graeme McLachlan Bragg, and Andrew Brown. "Asynchronous Simulated Annealing on the Placement Problem: A Beneficial Race Condition". Journal of Parallel and Distributed Computing. **169** pp 242-251 2022

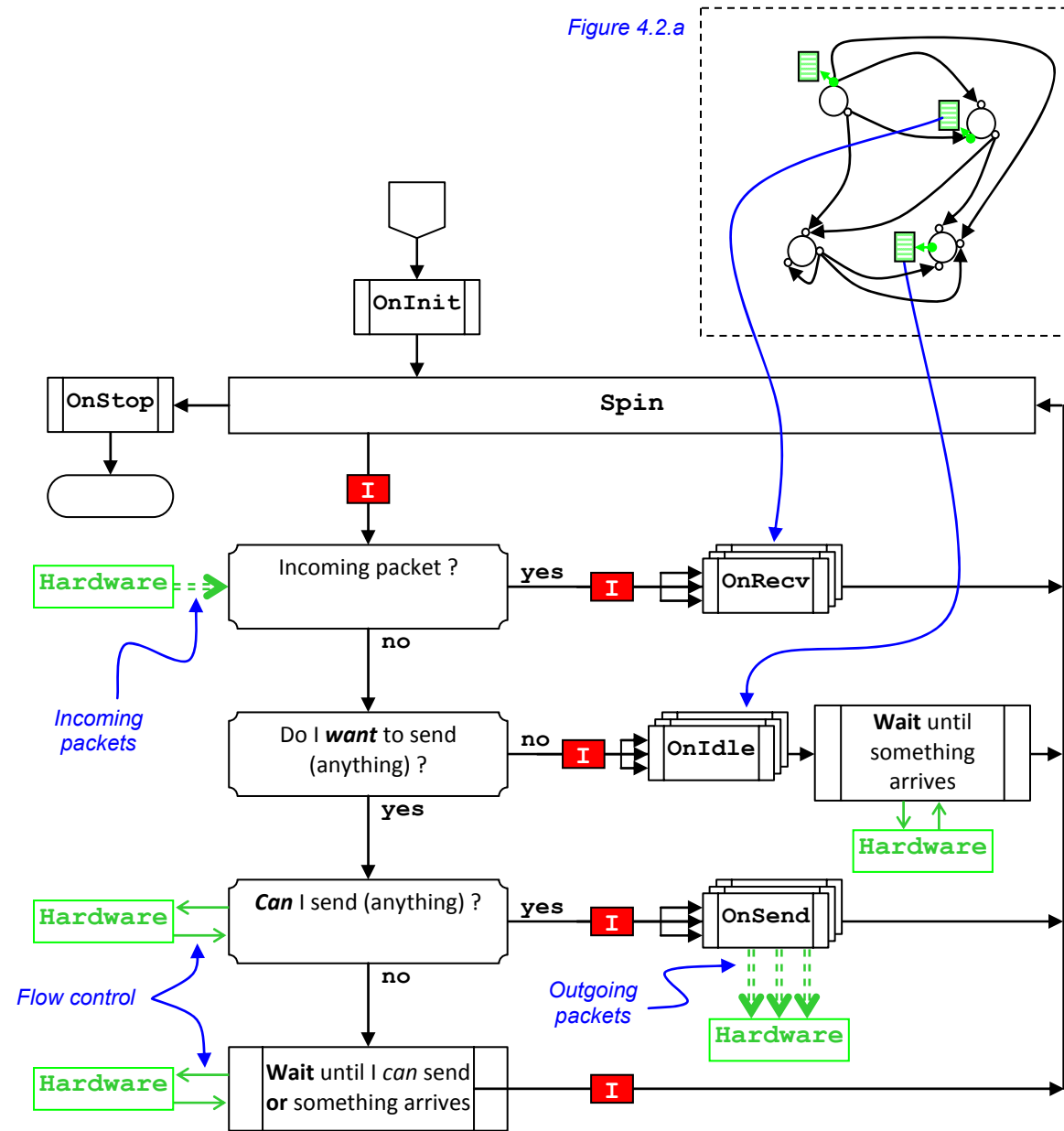
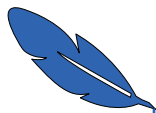


Figure 4.2.a

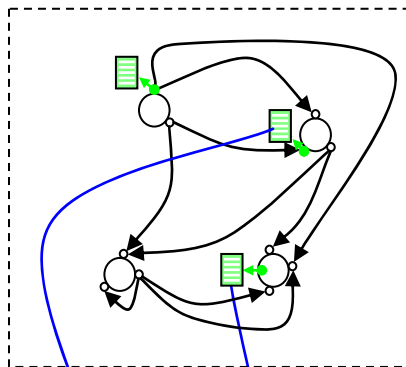
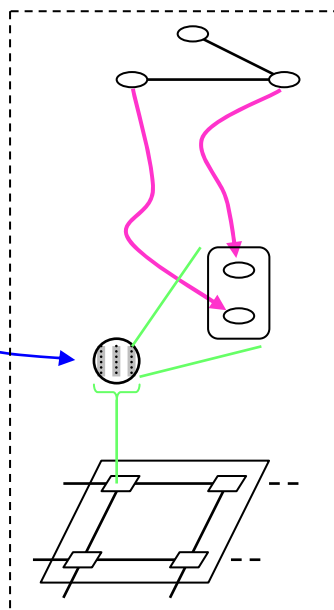


Figure 4.3.2.d: Abstract SoftSwitch (a spinning device multiplexer) model.

Figure 4.3.2.c





4.4 Tools infrastructure & ecosystem

To support development and verification of POETS applications we have developed an ecosystem of software simulators and verification tools. These tools complement and support the Orchestrator's hardware execution environment in several ways, including:

- Prototyping of new language features
- Libraries for generating application graphs
- High performance serialisation libraries for reading, writing, and type-checking application graphs
- Debugging of applications during development using simulators
- Verification of applications using model-checking and "aggressive" simulators
- Visualisation of application execution to support understanding
- High-performance execution environments to enable execution when the hardware is in use by others

These tools and infrastructure have been used directly by the researchers on the project and by 10 MSc and MEng students performing thesis projects on POETS, utilising the full spectrum of technologies: hardware and software. The tools have also been used indirectly by 80 MEng students as part of a 4th year coursework in High-performance Computing, allowing them to engage with the research even though we could not give them all direct access to the hardware.

Here we summarise the main tools developed over the project, looking both at the overall history, and developments since the last AB. We do not go into technical details, but instead focus on the eco-system and capabilities developed.

4.4.1 Libraries

While the application description language has been kept as simple as possible, it is still relatively complex to parse the applications and turn them into graphs. To avoid re-inventing the wheel each time, a set of shared libraries were developed to make it easier to read and write graphs, while also providing built-in error checking and high-performance. There are two main libraries that have been used:

- *Python retained mode* : when writing application generators and translators it is often necessary to build an in-memory graph representing the problem. This library provides a relatively high-performance python in-memory representation that works well up to ~1M nodes, with efficient streaming persistence to and from XML.
- *C++ streaming mode* : when writing generic tools and "production" application generators we need to work with graphs with up to ~100M nodes and ~1B edges. A streaming C++ API is used for this, which provides a single consistent API for both reading and writing graphs, and requires O(1) memory during parsing and generation in release mode.

Both libraries have remained stable as the language syntax and semantics have changed, with as few breaking changes as possible. Both APIs were first committed to git in June 2016 and have been maintained across 4 revisions to the XML syntax and extended to support additional hardware and language features as they arise.

4.4.2 Simulation

The POETS application language has simple semantics that support complex behaviour, so simulation of applications and emulation of hardware is needed for multiple purposes. The main simulation and emulation platforms now available include:



Epoch sim : This is the first simulation engine, developed in July 2016 during development of the application language and semantics. This pre-dated the Tinsel hardware and Orchestrator, and was used to create and develop initial applications. It remains the simulation engine of choice for developing new applications, as it provides simple “epoch” based semantics: simulation is implemented as a sequence of time epochs, and during each epoch every device gets the chance to send one message that is immediately delivered to all destination devices. This approach does not model true hardware concurrency well, but makes it easier for new developers (particularly MSc students) to start developing and understand applications.

This simulator has served as a key experimentation platform for trying out new ideas, particularly when looking at cross-project features. For example, the hardware idle feature was available in simulation before it was complete in hardware, and the various application language revisions were prototyped here before being implemented in the Orchestrator. Since the last AB we have used epoch_sim to explore different approaches to IO, adding support for external devices and supervisor devices. The simulation functionality has also been upgraded, adding the ability to delay messages – this is in response to the needs of MSc students, where the epoch-based semantics were a bit *too* simple.

Queue sim : Epoch sim has excellent debuggability, but is relatively slow, and inherently single-threaded. Testing applications requires the ability to simulate larger graphs for longer runs, particularly when hardware is not available. In August 2016 a multi-threaded simulator called queue sim was developed to fill this role, as neither the hardware nor orchestrator were yet available. This simulator has been consistently maintained, and still passes regression tests, but was obsoleted by other features – the introduction of the hardware idle feature required the introduction of a global lock, reducing queue sim performance, and leading to the development of POEMS (which will be discussed shortly).

Graph sim : Epoch sim is good for initial development, but is not very effective for refining applications and discovering more subtle problems related to concurrency and message ordering. In 2018 a new simulation engine was created called graph sim which is able to efficiently simulate multiple message orderings, including first-in first-out, last-in first-out, and random message delivery. This was developed in response to bugs found in applications running in hardware that could not be replicated in epoch sim due to its simple semantics. Graph sim remains actively maintained and has been used by multiple researchers and students to help identify more subtle problems in graphs.

POLiteSWSim : As well as the main XML application language, there is also a lower-level Tinsel-specific API called POLite which is used to explore hardware features before lifting them up to the application language. Initially the only way to run applications was in hardware, which made it difficult to debug POLite applications and perform continuous integration. In May 2021 a drop-in simulation back-end was developed, allowing POLite applications to be executed and debugged without access to hardware – this was motivated by the need to debug problems in DPD under POLite, but has also been applied to other applications.

POEMS : Changes to the language semantics meant that queue sim become obsolete as a high performance simulator. At the same time the availability of larger hardware and the full Orchestrator required the ability to simulate much larger graphs on more parallel machines. In July 2019 a completely new simulation engine called POEMS was created – the name POEMS comes from a potential hardware architecture considered by researchers at Cambridge, which inspired the software architecture of POEMS. This simulator is intended to be a full execution engine rather than a simulator, and has very little support for



debugging. It is designed to minimise cache traffic and maximise parallelism in modern multi-core processors, using internal batching and lock-free data-structures to scale to 64 or more cores. Optimisation has continued since the last AB in order to support debugging of large-scale DPD applications, with new message batching approaches allowing it to deliver more than 2 billion messages per second in a 64-core machine.

4.4.3 Debugging and verification

The simulators can be used to run applications, but their main purpose is to support development, through debugging and verification. Debugging and verification is supported through multiple techniques:

Assertions : one of the simplest techniques for understanding application behaviour is simply to assert device invariants at the beginning and end of each device handler. This is difficult to do in hardware, as the code needed to support assertions adds substantially to code size and impedes optimisation, so often it is impossible to fit assertions into Tinsel program memory. All the simulators support the use of assert and will immediately flag the point at which device invariants are broken – lower performance higher debuggability simulations (such as epoch sim) can also dump the exact state of the simulation at the time the assertion failed.

Logging : the XML application graph supports a function called `handler_log(level, msg, ...)`, which allows device handlers to call a printf like function. The level parameter controls the amount of information printed, with higher levels printing more messages. This is again very expensive in hardware, with `handler_log` having severe performance and program memory size penalties in Tinsel/Orchestrator. In simulation full logging is supported and the log level can be changed at run-time – the only tradeoff is that simulation will be slower. This provides another route to debugging applications, and is particularly useful for exploring the handler states that lead to an assertion failure.

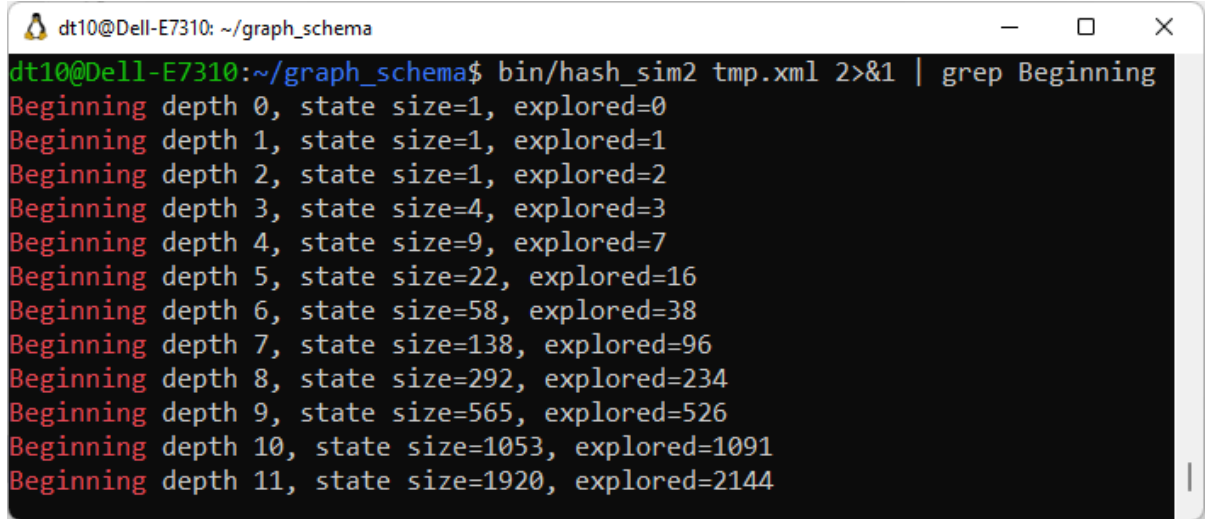
Tracing : sometimes logging is not enough, as it hides dependencies and causality within the execution – we need to know the true causal chain of events that led to an event, rather than the linearised history of logging. Both epoch sim and graph sim support application tracing, which captures the complete causal history of send and receive events, including all state changes and the messages in flight. This can then be examined by the developer to determine how and why a specific state was reached, following messages forwards and backwards. Figure 4.4.3.c shows event-traces for a heat equation application – on the left is the result for a FIFO message delivery order, and on the right a LIFO order. These event traces are very complex, and usually one of the last resorts for debugging, but in those cases are invaluable as they provide a level of detail that is simply impossible to recover from the hardware.

Model checking : as applications and hardware become more complex, we started to encounter situations where working applications failed on newer hardware, or they worked for one problem size then hung for another size, or just become non-deterministic. Extensive simulation never seemed to find the right execution order, but clearly something was going wrong. In July 2019 this was addressed with a new model checker, which was able to perform a breadth-first traversal of every possible execution path. Previous attempts to translate POETS applications into existing model checking languages such as TLA+ and Spin were partially successful, but required manual steps, and ran the risk of mis-translating the system.

To allow direct model-checking of applications, we exploited the formal POETS model of a graph of devices (finite-state-machines) where the exact simulation state is captured by the current device states, plus the state of all messages in flight. The existing simulation



infrastructure and parsers were used to create the ability to form a 128-bit hash over the entire state. This hash was designed to be associative and commutative with respect to message send and receive events, allowing the hash to be updated in $O(1)$ time, regardless of the problem size. When performing simulation we can evaluate the next hash value after each step, allowing an exhaustive breadth-first search through every possible simulation state.



```
dt10@Dell-E7310: ~/graph_schema
dt10@Dell-E7310:~/graph_schema$ bin/hash_sim2 tmp.xml 2>&1 | grep Beginning
Beginning depth 0, state size=1, explored=0
Beginning depth 1, state size=1, explored=1
Beginning depth 2, state size=1, explored=2
Beginning depth 3, state size=4, explored=3
Beginning depth 4, state size=9, explored=7
Beginning depth 5, state size=22, explored=16
Beginning depth 6, state size=58, explored=38
Beginning depth 7, state size=138, explored=96
Beginning depth 8, state size=292, explored=234
Beginning depth 9, state size=565, explored=526
Beginning depth 10, state size=1053, explored=1091
Beginning depth 11, state size=1920, explored=2144
```

Figure 4.4.3.a: Start of model checking.

Figure 4.4.3.a shows the start of a model checking execution for an application which contains tree-like synchronisation. As the model checker explores the possible state transitions, the number of unique states to explore next (“state size”) increases, and the number of unique states already explored also grows. Figure 4.4.3.b shows the progression of the set of states to explore next versus the number of unique states explored with time, showing the inherent cyclic communication nature of this particular application, but also demonstrating that it is not truly cyclic – it never revisits a state it has previously visited.

The ability to model-check applications directly is very powerful, and only really possible due to the strong underlying semantics of the model. It allows us to look for very unlikely edge-cases which are difficult to find in simulation, but can be found through exhaustive search. A memorable bug uncovered using this method was due to a comparison of a fixed-point number which used `<` rather than `<=` in one expression, and so deadlocked at run-time. This bug occurred for large-scale problems in hardware, but thousands of simulation runs could never find the root cause. Exhaustive model-checking eventually forced an assertion to fire, allowing the bug to be fixed.

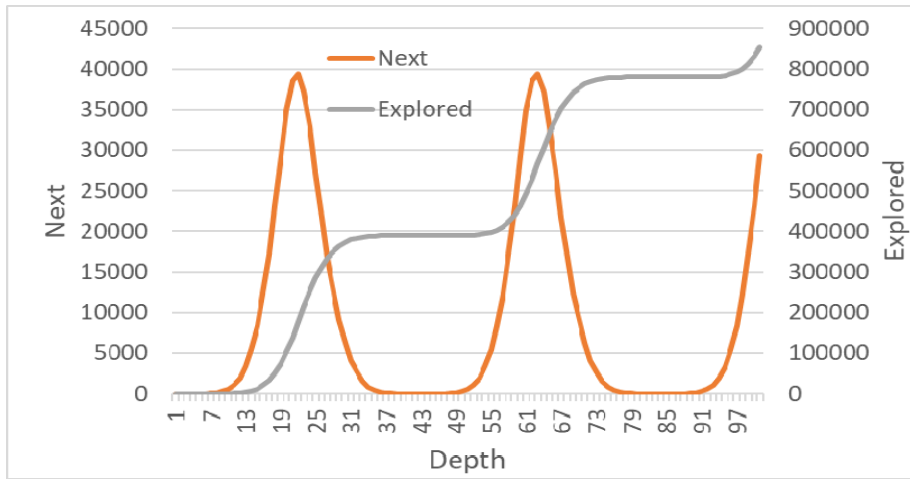
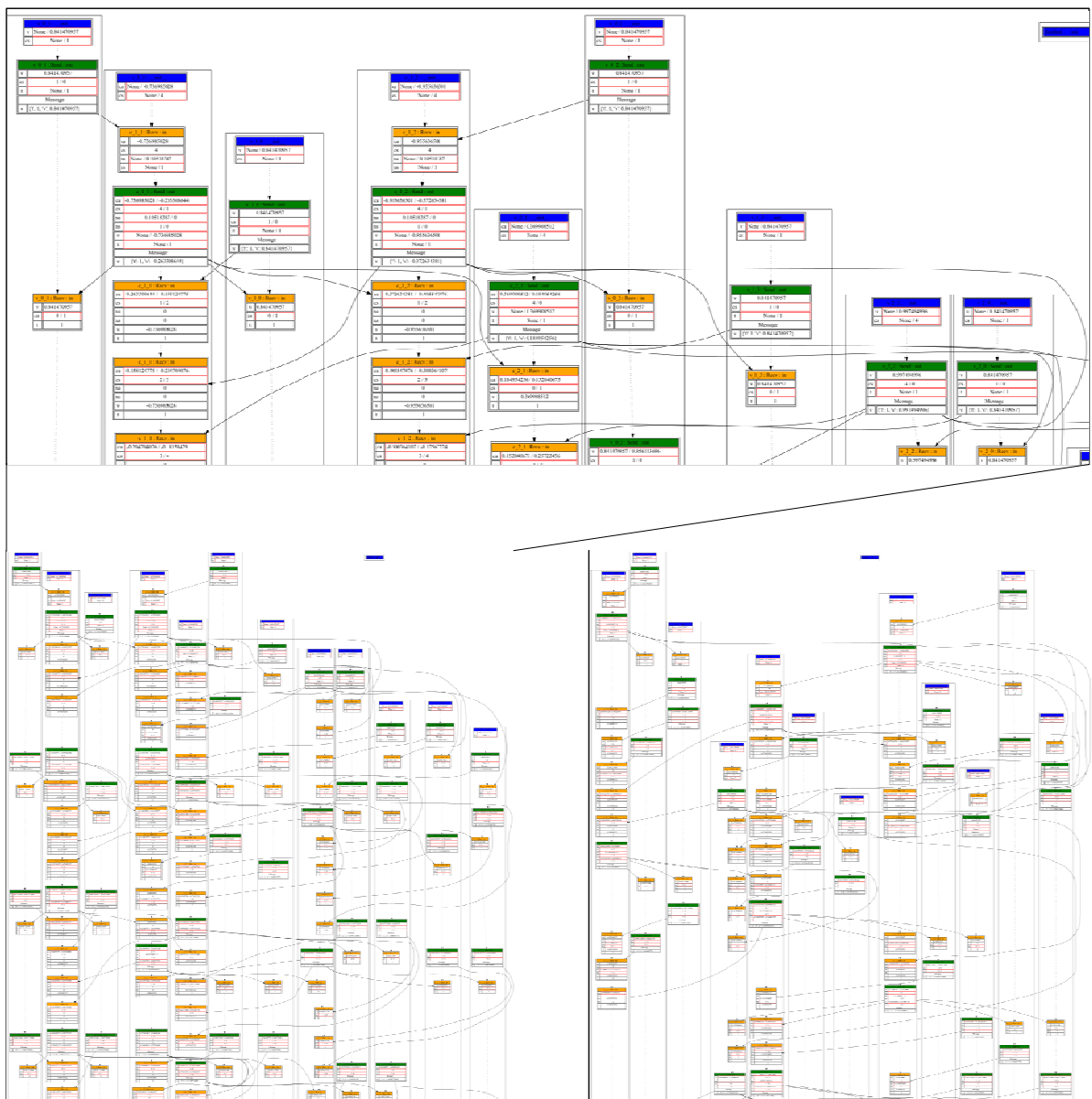


Figure 4.4.3.b: Number of states versus depth.



FIFO Simulation Order

LIFO Simulation Order

Figure 4.4.3.c: Event traces for a heat equation simulation.



4.5 Visualisation - data & traffic fluxes

Knowing *how* an application executes can be just as important as knowing that it executes correctly. POETS includes several sources of performance data that can be interrogated: Tinsel includes several hardware-provided performance counters, and both the Softswitches and Motherships log various different events. The Orchestrator can leverage all of these data sources to give insight into how a given application uses the hardware and behaves.

Tinsel provides several performance counters that can be used to determine core utilisation and cache performance. These are interrogated by the softswitch as part of its control loops. In addition to the hardware performance counters, the Softswitch tracks the number of times each type of handler is called, how many actual packets were sent or received and how many times the network was unable to accept a packet. A summary of the instrumentation is shown in table 4.5.

Table 4.5: Softswitch instrumentation data. All fields except for the cycle count are the raw count since the last instrumentation packet was sent.

Cycle Count	The CPU cycle count at the time the packet was generated.
Receive Count	The number of actual packets received.
Receive Handler Count	The number of individual receive handlers triggered. One actual packet may trigger multiple receive handlers, e.g. a broadcast.
Transmit Count	The number of packets sent to other POETS devices.
Mothership Count	The number of packets sent to the Mothership.
Transmit Handler Count	The number individual transmit handlers triggered. One transmit handler may emit several packets.
Idle Count	The number of times the idle branch is entered.
Idle Handler Count	The number of individual idle handlers triggered.
Network Block Count	The number of times the network has been unable to accept a packet for transmission. Indicated network congestion.
Cache Miss	The number of times there has been a cache miss.
Cache Hit	The number of times there has been a cache hit.
Cache Writeback	The number of times there has been a cache writeback.
CPU Idle Count	The number of cycles where the CPU did no work.

By default all softswitches send their instrumentation every second, but both the time interval and which softswitches are enabled is configurable at runtime. This allows for the granularity and performance overhead of system monitoring to be controlled.

4.5.1 Offline visualisation

The Orchestrator writes out all received Softswitch instrumentation to a series of files for later processing. These files can be visualised with a comprehensive visualisation script to give an overview of the behaviour of an experiment after the fact.

Figure 4.5.1.a shows a plot of the packet send rate for every thread that was part of a 128x128 heated plate simulation spread across 1025 threads. This is a smaller problem than used to generate the exemplar from section 2.4. This plot gives an overview of the system-level packet traffic and can be useful for determining whether an application is compute, memory or network bound when taken in the context of the other instrumentation plots. The visualisation script also produces per-core plots (figure 4.5.1.b) and an aggregate plot of all threads (figure 4.5.1.c).

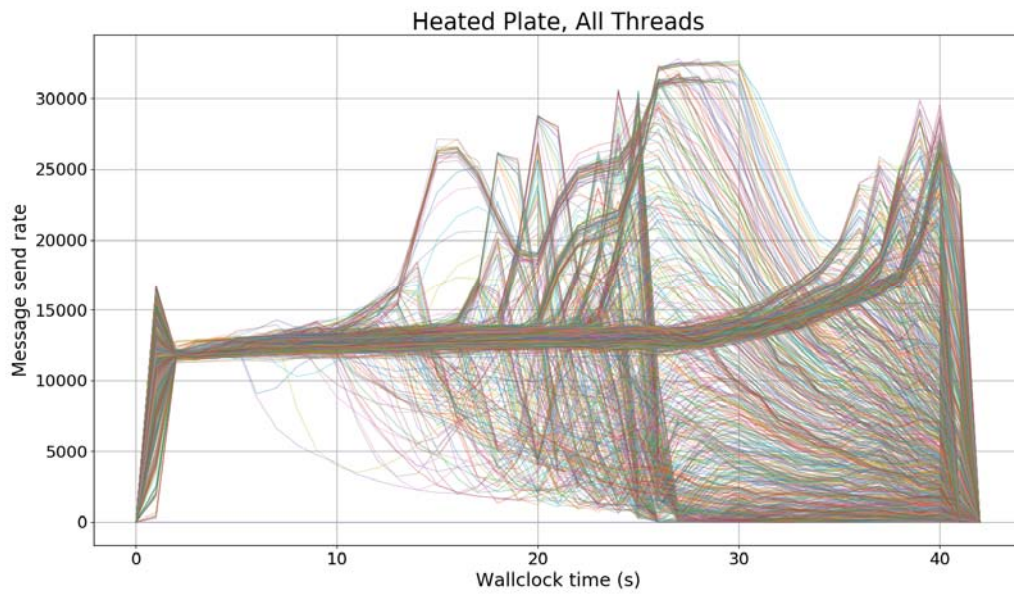


Figure 4.5.1.a: Packet send rate for all threads used by the application. Each coloured line represents a different thread.

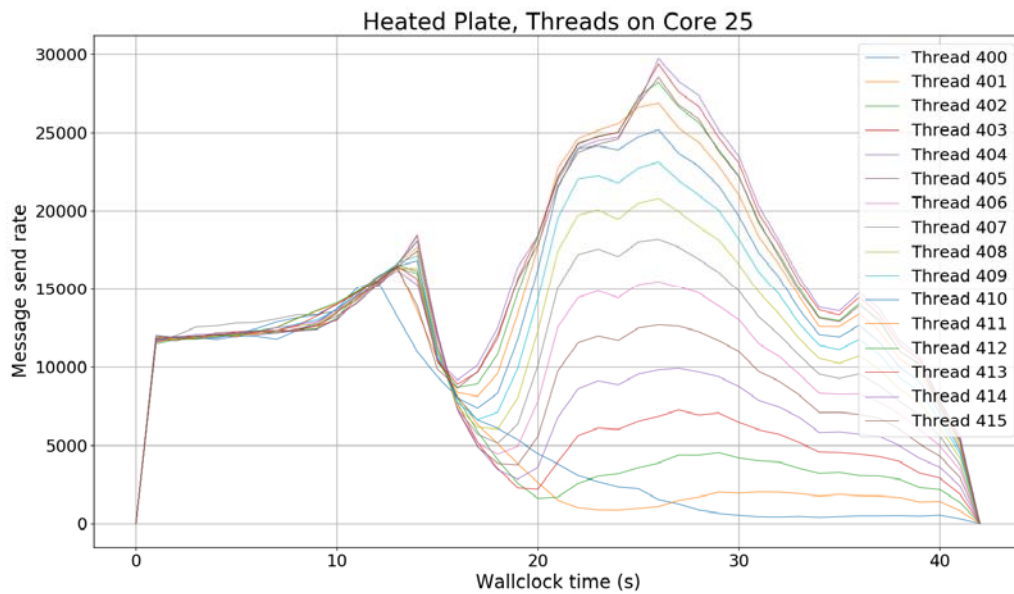


Figure 4.5.1.b: Packet send rate for all threads hosted on a single processor core. Each coloured line represents a different thread.

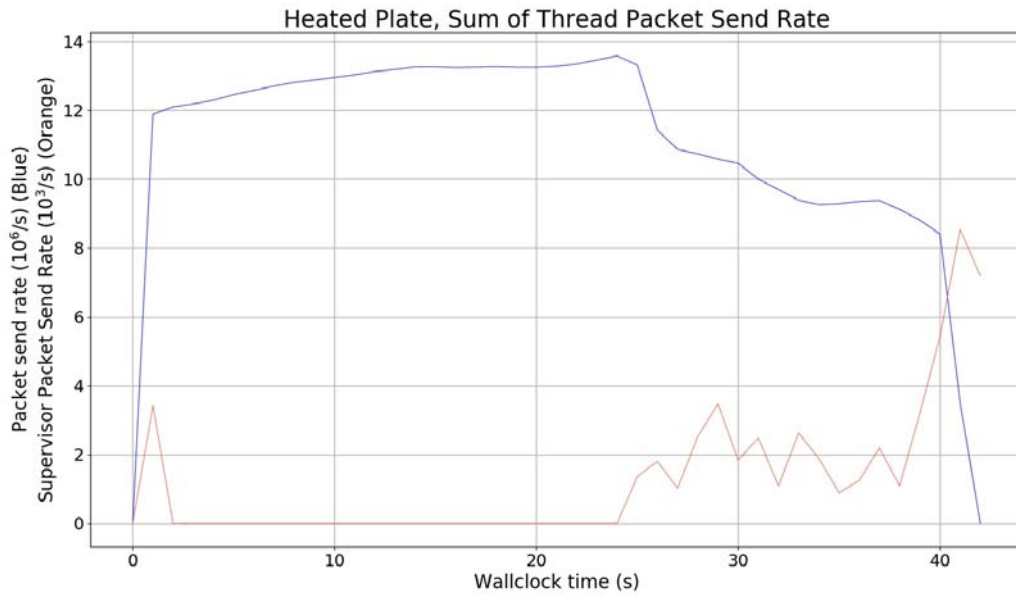


Figure 4.5.1.c: Cumulative packet send rate of all threads used by the application. The blue line represents core-core packets while the orange/red line indicates packets (in this case completion packets) sent to the Mothership.

These plots show that different threads are completing their computation at different times and in different ways. Some threads run for the complete time and finish suddenly at around 42 seconds while others complete suddenly around 26 seconds. Additionally there are threads that see increasing network traffic that then decreases sharply before either gracefully decaying to completion or having a significant resurgence in network activity, as shown in figure 4.5.1.b. The variety of different completion behaviours is reflective of the fully asynchronous nature of the problem and the waves of compute that occur.

At a first glance, these plots could indicate a network bottleneck as the total message traffic across the system appears to hit a plateau. The plot of incidences of the network blocking transmission (figure 4.5.1.d) shows that no packets were refused by the network at any point during the simulation, so the network itself is not struggling.

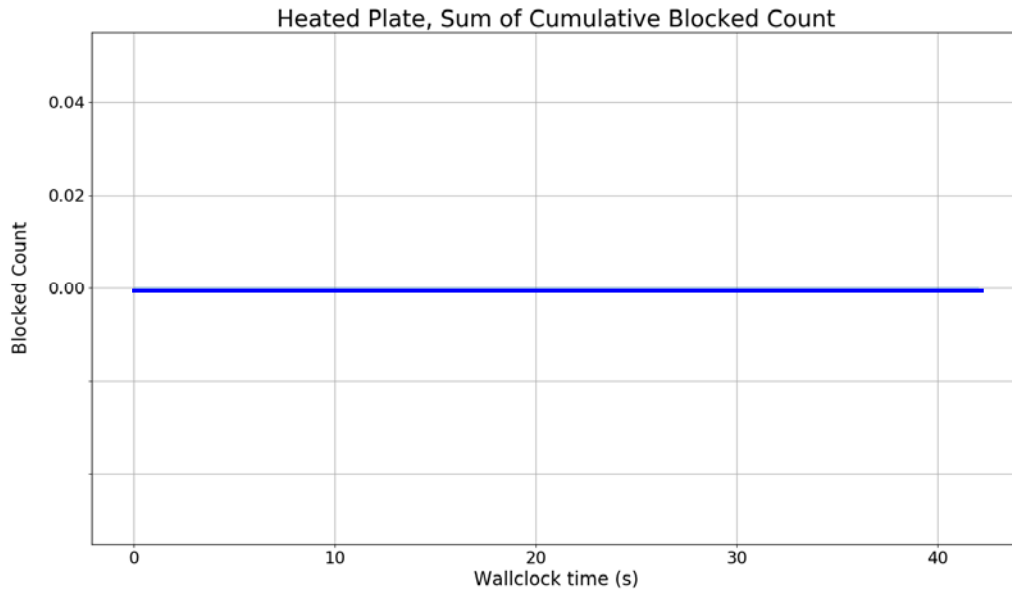


Figure 4.5.1.d: Cumulative network block count.

To get more of an insight we can look at the plot of CPU utilisation (figure 4.5.1.e). This shows that the cores have a significant amount of time (~40%) where they are unable to execute useful instructions while the application is running. Once all threads on a core “complete”, the idle percentage falls to near 0% as the softswitches spin waiting for packets.

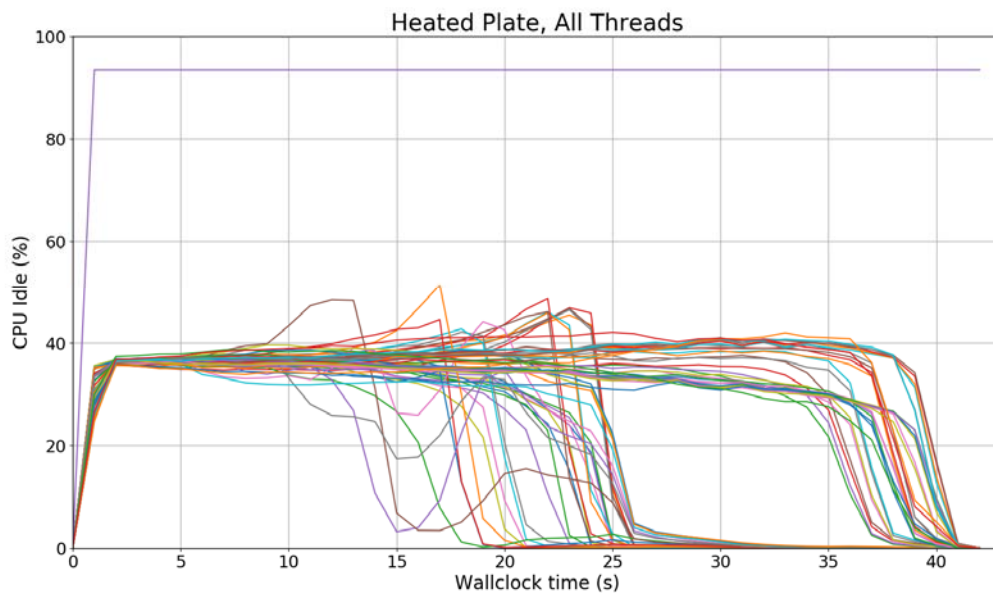


Figure 4.5.1.e: CPU Idle time of all cores used by the application. Each coloured line represents a single CPU core.

This behaviour could have a few different root causes. For example, a high cache miss rate that results in threads stalling and waiting for memory access could result in low core utilisation. The cache miss rate plot (figure 4.5.1.f) shows that this is not the case with this problem showing quite a low cache miss rate compared to other applications.

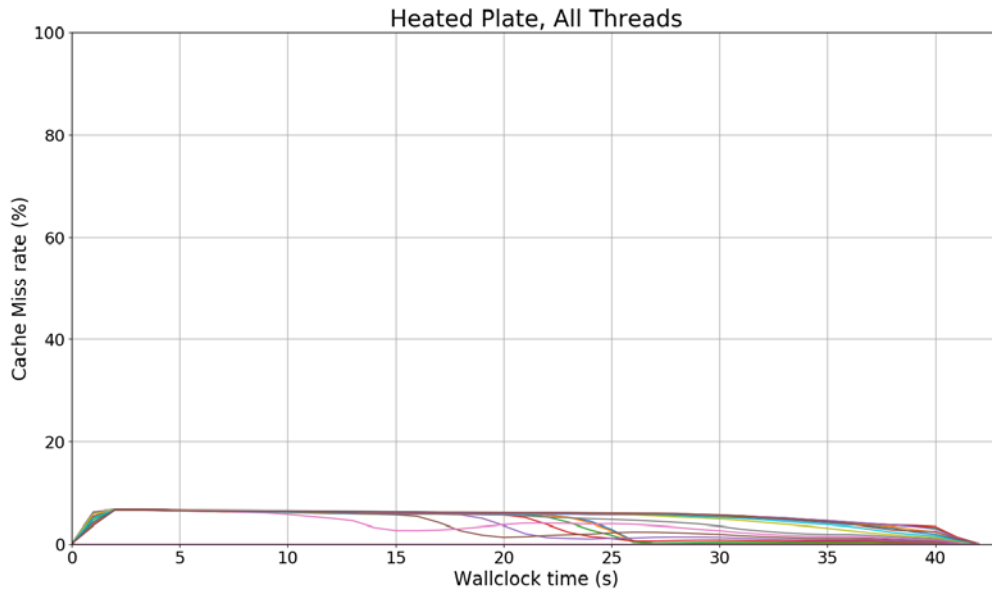


Figure 4.5.1.f: All cores cache miss rate. Each coloured line represents the cache miss rate for all cores hosted by a single mailbox.

Another possible cause could be FPU contention. In the current deployment of Tinsel, a single FPU is shared by four cores (64 threads) and this could limit applications where the content of handlers is mostly FPU operations. High FPU contention could result in threads stalling as the FPU tries to keep up with high demand.

To get more of an idea what is happening, we can exploit the flexibility of the POETS hardware implementation to investigate this possibility by doubling the number of FPUs available to each mailbox. Figure 4.5.1.g shows the CPU idle time for a simulation run of the same problem with double the FPUs. While runtime has decreased, it has only done so by a couple of seconds and we still suffer from a low CPU utilisation. Clearly FPU contention is not the root cause.

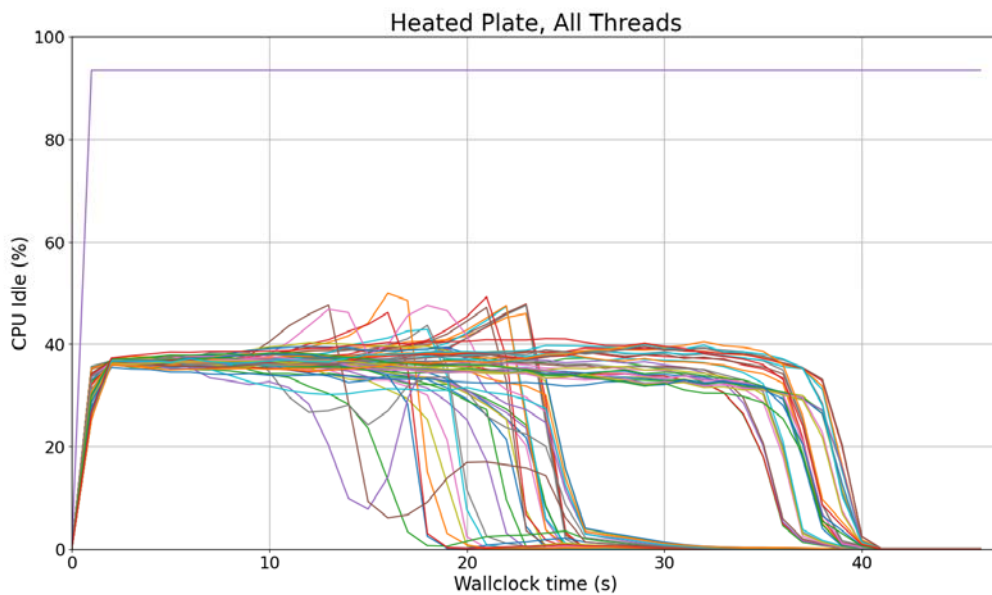


Figure 4.5.1g: CPU Idle time of all cores used by the application with double the number of FPUs allocated to each mailbox. Each coloured line represents a single CPU core.



Having ruled out a high cache miss rate and FPU contention as the cause of our low CPU utilisation, we approach the limits of what the current instrumentation package can definitely tell us. Without additional hardware-derived performance counters to indicate the reason for the high idle time, we cannot draw any firm conclusions as to what is limiting the performance in this case.

This particular application has a very high packet-to-processing ratio (it is a packet storm after all) so the time taken to load each packet into the mailbox memory could be a significant factor.

Beyond raw performance plots, the offline visualisation scripts produce a series of plots that illustrate the distribution of paths taken by the softswitches at any given time. Figure 4.5.1.h shows the handler distribution of a single thread while figure 4.5.1.i shows a core-level overview of the same data for all threads hosted by the core. These plots are useful for seeing how the balance between transmit and receive change over time. In this case, we can see that the core is almost equally balanced in terms of the number of times the send and receive handlers are executed while it is actively computing. This indicates that for the majority of the active compute time, the softswitch is emitting packets in response to a single inbound packet and does not have multiple inbound packets queued.

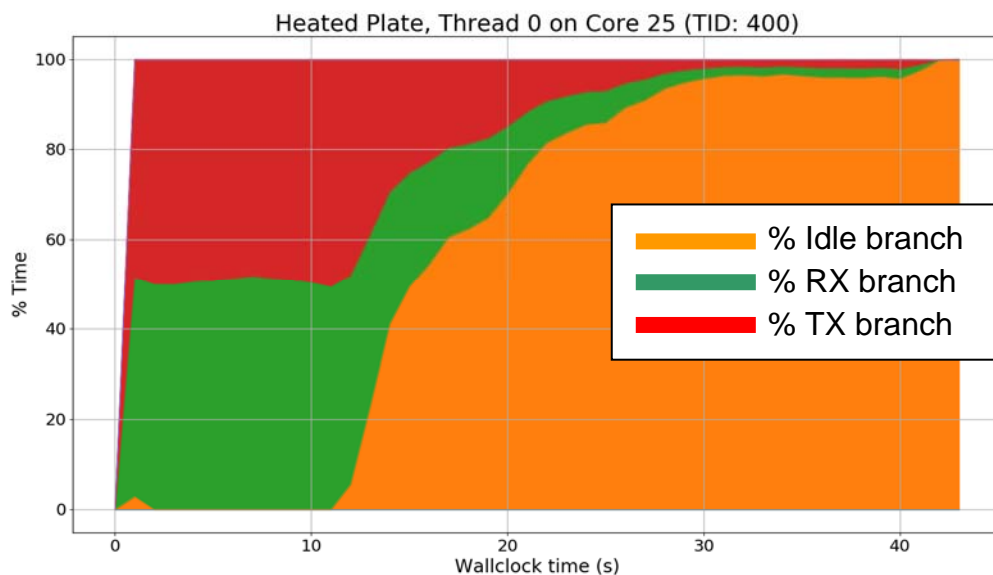


Figure 4.5.1.h: A single-thread view of the distribution of handler events at any given time.



Heated Plate, All threads on Core 25

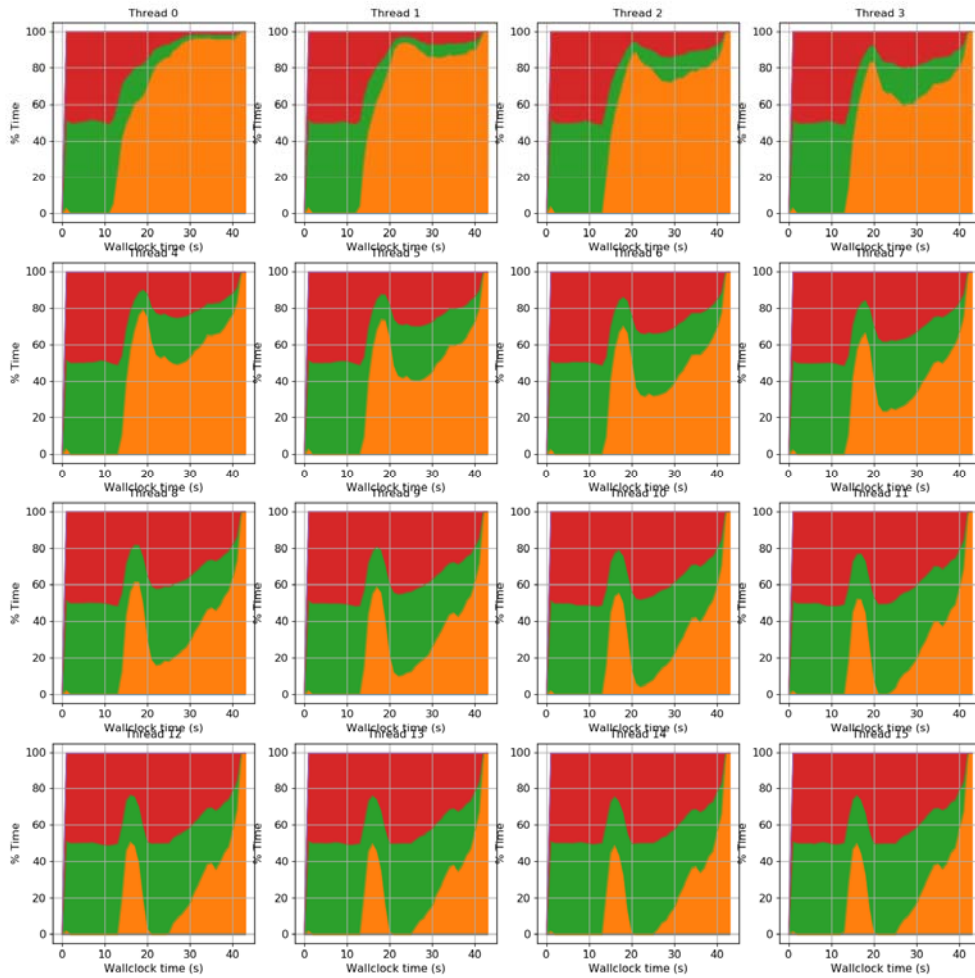


Figure 4.5.1.i: A core-level view of the distribution of handler events at any given time. The colour key is the same as in figure 4.5.1.h.

4.5.2 Online visualisation

In addition to comprehensive offline visualisation capabilities, we have developed some online visualisation capability to provide performance insights during simulation runs. These range from a conceptually-simple real-time scrolling plot of the packet send rate to a fully-featured monitoring package that enables and disables feeds from individual softswitches and maps the underlying hardware performance to application-level devices.

Figure 4.5.2.a shows a snapshot of the real-time packet send rate plot. This was demonstrated at the previous advisory board and is essentially a real-time version of the plots shown in section 2.4 and figure 4.5.1.a. This capability can give an insight into how an application is using the network and can help indicate whether processing is relatively even across the compute fabric or whether specific areas are stalled.

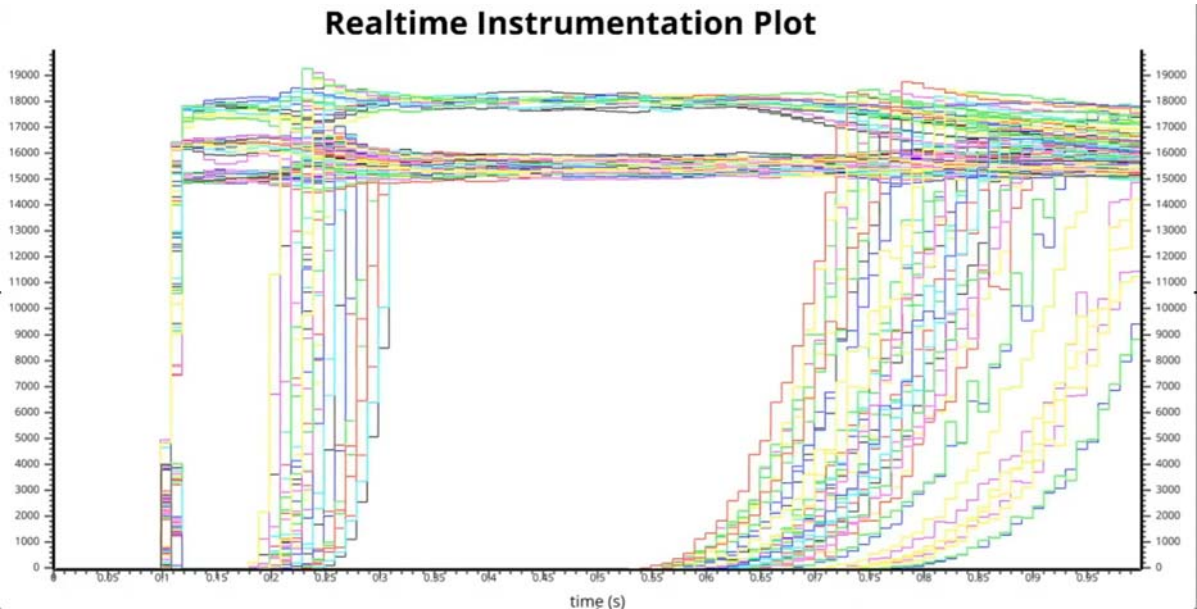


Figure 4.5.2.a: Real-time packet send rate plot.

This basic concept has been extended by a postgraduate project student to become a more full-fledged web-based visualisation package. In addition to a real-time moving line plot of the per-thread packet send rate, a heatmap of the packet rate is presented (figure 4.5.2.b). This shows definitively which parts of the system are sending packets at any given time. Additionally, the visualiser shows delayed plots of the idle time and cache metrics (figure 4.5.2.c).

The visualiser allows inspection of different levels of instrumentation, e.g. system level, per-mailbox, per-core, etc. This enhanced capability allows us to gain an even deeper understanding of how an application that is currently running is behaving.

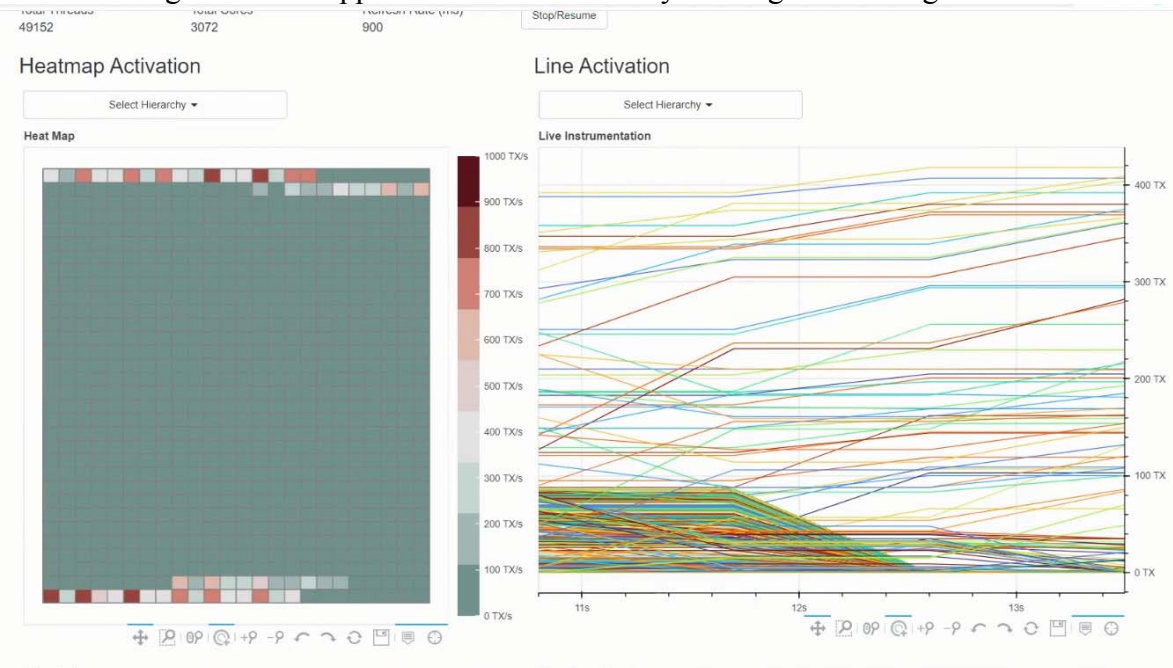


Figure 4.5.2.b: Web-based visualisation package showing the packet send rate heatmap and real-time line plot.

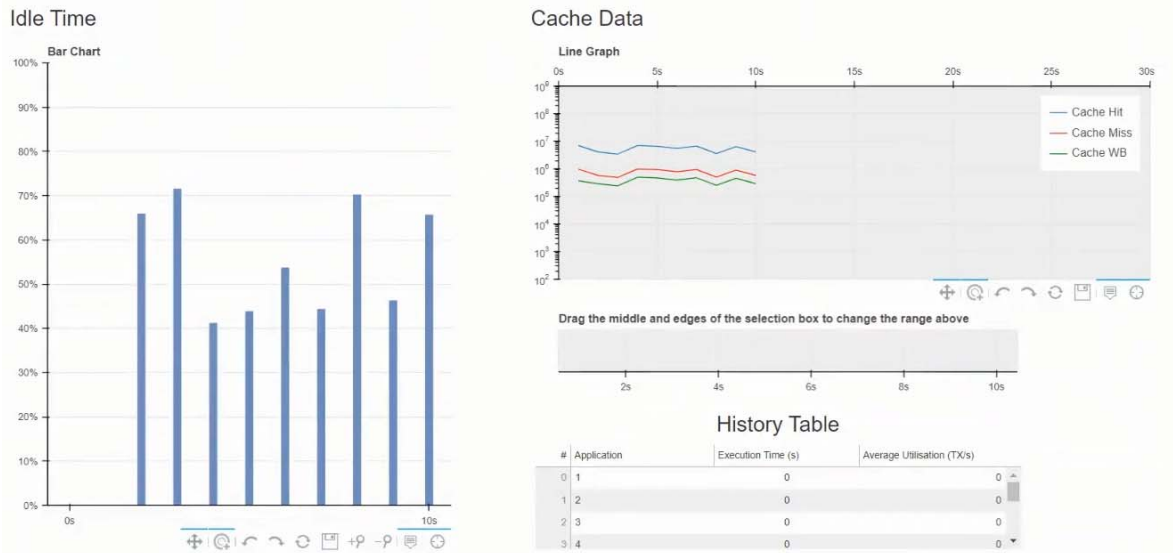


Figure 4.5.2.c: Web-based visualisation package showing the aggregated CPU idle time and cache data. This data is delayed from the heatmap and line plots as it requires additional processing before presentation.



5. Domain specific applications - I

5.1 Introduction

The POETS research program has very much been a polymath's dream. Virtually every component of the development stack has been novel, and we have had to develop, from first principles, tools and techniques that conventional computer engineering activities take for granted. Pretty much the only tools we have used "off the shelf" have been hardware synthesis and compilers.

Whilst the hardware/software co-development stack has been highly unusual and extremely challenging - as possibly indicated by the previous chapters - the *raison d'être* of the entire research program has been to identify and explore real-world application domains, to establish how and if the (almost) constant scaling of event-based computing can benefit these areas. "Flagship" domains - areas we have explored in some detail - will be discussed in the next section. Here, we outline the other areas that we have identified as being potential beneficiaries of POETS technology. We have also discovered, along the way, further domains that we simply have not had the bandwidth to explore. We present the areas that we have studied in non-trivial detail. Some (high voltage partial discharge modelling) have not been developed to the point of interesting results. Some (graph theoretic algorithms) have performed less well than expected (the scaling *is* impressive, but it is not constant). The others, described here, have **all demonstrated the expected near-constant execution time scaling behaviour** predicted as the cornerstone of this research program.

Most of the **absolute** execution times are taken from an FPGA-based hardware platform that is over a decade old, running soft cores at a frequency of 200MHz. The single thread comparator speeds are taken from conventional contemporary machines, typically with a clock frequency $> 2\text{GHz}$; still POETS prevails.

The scaling is the thing: Irrespective of the absolute wallclock performance, the scaling behaviour implies that larger and larger capabilities can be realised, simply by adding more hardware.

POETS was derived from the neuromorphic computation capabilities of SpiNNaker. SpiNNaker was designed from the outset as a spiking neural network simulator, and contained many design decisions that made it admirably suited for that application. POETS is a generalisation of that approach, building on the design philosophy of event-triggered compute that it embodied. We have, naturally, reproduced experiments undertaken on SpiNNaker on the POETS architecture, but they are not reported here because they add little to the neuromorphic research trajectory that SpiNNaker itself cannot provide¹³¹⁴¹⁵¹⁶¹⁷.

¹³ A.D. Brown, S.B. Furber, J.S. Reeve, J.D. Garside, K.J. Dugan, L.A. Plana and S. Temple, "SpiNNaker - Programming Model", IEEE Transactions on Computers, 64, no 6, June 2015, pp1769-1782, doi 10.1109/TC.2014.2329686

¹⁴ A.D. Brown "Event-driven computing" keynote talk, ACSD/PN 2015, Brussels

¹⁵ A.D. Brown, J.E. Chad, R. Kamarudin, K.J. Dugan and S.B. Furber "SpiNNaker: Neuromorphic simulation - quantitative behaviour", IEEE T Multi-Scale Computing, 4, no 3, July 2018, pp450-462, doi 10.1109/TMSCS.2017.2748122.

¹⁶ Mahyar Shahsavari, David Thomas, Andrew Brown, Wayne Luk "Neuromorphic Design Using Reward-based STDP Learning on Event-Based Reconfigurable Cluster Architecture" ICONS 2021: International Conference on Neuromorphic Systems July 2021 pp 1-8 doi 10.1145/3477145.3477151



5.2 Graph Traversal

Graph traversal is a subclass of graph analytic algorithms used for analysing graph properties such as, for example, shortest paths. We focused on graph traversal applications that can be described with a combination of the following requirements:

- type of analysis, e.g., **single-source shortest paths to all nodes (SSSP)** or **all-pair shortest paths (APSP)**;
- graph topology: irregular graphs or regular grids;
- edge weights: weighted versus unweighted/unit edges.

Under this application topic, we considered two practical use cases: drug discovery and seismic raytracing.

5.2.1 Drug discovery: Biological systems can be modelled as networks of protein interaction representing normal cellular functions with up to 20,000 proteins and over half a million connections. Disease mechanisms can be considered as emerging from collections of pathological interactions over such interaction networks. Identification and perturbation of those systems aimed at combating the disease is an underlying principle of drug discovery, developed in e-Therapeutics. Numerous measures have been used in studies of network percolation and robustness with two commonly used measures being network diameter and the average of APSP. As such, this use case represents *APSP calculation on irregular unweighted graphs*.

5.2.2 Seismic raytracing: Marine geologists use seismic tomography to construct 3D models of the seismic velocity at points in the crust and underlying mantle in order to study the geophysical properties and understand magmatic, hydrothermal, and tectonic processes involved. A volume of Earth's crust is represented as a spatially regular 3D grid overlaid with an acoustic velocity model represented by edge weights. The problem reduces to an *SSSP calculation over a regular weighted grid*. Due to the volumetric nature of the data, the grid sizes can vary from millions to tens of billions of points depending on the target accuracy of the model.

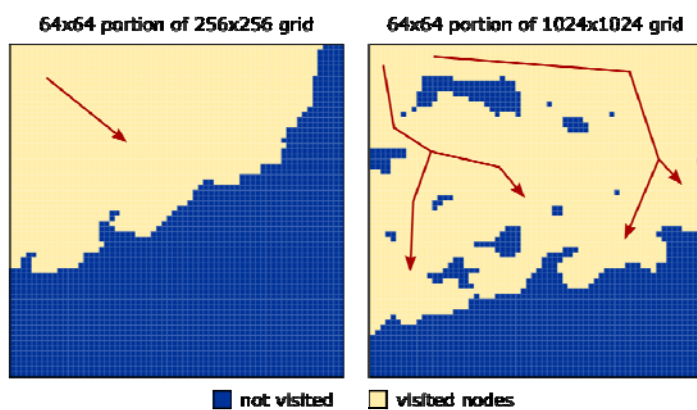


Figure 5.2.2.a: A snapshot of the asynchronous SSSP progress showing the change in update propagation between different graph sizes due to traffic density. Arrows show the general direction of travel.

A crucial feature added to the POETS platform is a hardware-implemented refutable barrier primitive that can be used for termination detection or global synchronization. This feature allowed a flexible multi-mode implementation of the graph traversal algorithm that can switch between asynchronous and globally synchronized communication.

Graph traversal is a great opportunity to study POETS communication fabric as the

¹⁷Alexander Rast, Mahyar Shahsavari, Graeme M. Bragg, Mark Vousden, David Thomas, and Andrew Brown. "A Hardware/Application Overlay Model for Large-Scale Neuromorphic Simulation". 2020 International Joint Conference on Neural Networks (IJCNN). 19-24 July 2020. DOI: 10.1109/IJCNN48605.2020.9206708



application's performance is mainly dependent on communication. Instead of analysing message passing at HW level, we observed emergent statistical effects at the application level (the level of logical devices). For example, figure 5.2.2.a shows a phase transition in the message behaviour due to traffic density. Other observed and described effects are the wave behaviour of updates, head traffic, back traffic, revisits, and reconvergence.

Across all tested graph types (from regular grids to random scale-free graphs), six topologies prefer the asynchronous mode with the average speedup of 5.2 and ten prefer the synchronous mode with the average speedup of 2.4. However, there was no obvious indication of why, therefore we resorted to a model-fitting techniques (namely, PCA followed by gradient descent) to create performance models. These models can be used as a classification heuristic to predict communication mode preference from the properties of the application graph at design time. Figures 5.2.2.b and 5.2.2.c show the experimental performance comparison between model-assisted implementation of the graph traversal on POETS (shown as **POETS with model**) and the original POETS implementation (shown as **POETS no model**), as well as a number of state-of-the-art performance results from the literature.

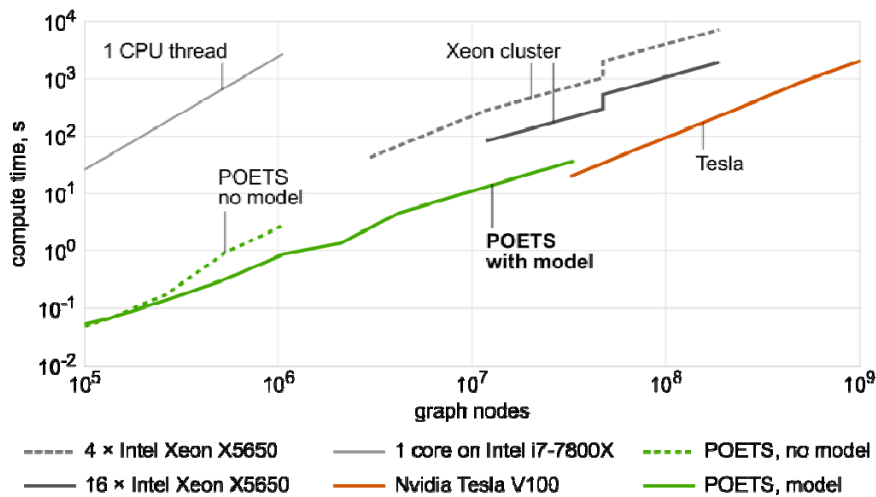


Figure 5.2.2.b: Seismic raytracing experimental results (SSSP on a regular 3D grid with weighted edges) compared to the state of the art.

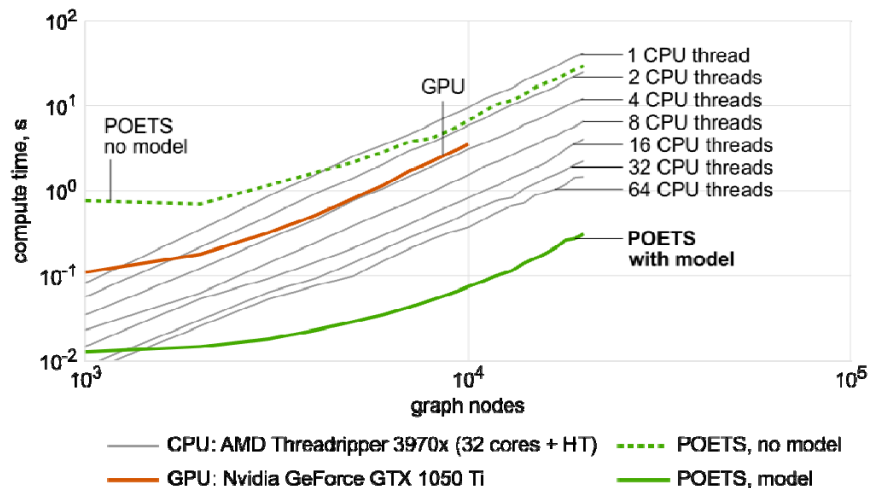
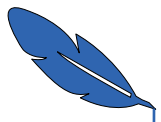


Figure 5.2.2.c: Drug discovery experimental results (APSP on an irregular unweighted graph).





5.3 Genomic imputation

One of the most important concepts in modern human genetics is the ability to link observable traits back to their causative genes. Whilst the original unravelling of the human genome was achieved via genetic (Sanger¹⁸) sequencing, even modern (next generation) sequencing remains prohibitively expensive for studies that require the genetic data of many individuals.

One such study that has gained in popularity and accuracy over the past decade is called a Genome Wide Association Study (GWAS). These studies sample marker locations spread across the entire genome and are conducted on populations at large (1k-100k participants). By separating the sampled data based on an observable trait, differences in the allele (DNA base – A/C/T/G) frequencies of the partitioned groups can be used to identify loci with statistically significant correlation to the trait under investigation. Studies of this nature have been enabled by a different, silicon-based technology called Genotyping-By-Chip (GBC) that provides DNA sampling across the whole genome.

The accuracy of GWAS improves as the number of participants increases, due to the increase in the probability that the data includes rare genetic variants. Early studies sampled around 1K participants, yet sample sizes of 10M haplotypes are predicted within a decade. Moreover, genotyping-by-chip (GBC) technology has also improved exponentially over the past decade, with early studies sampling 1.4M markers and yet modern studies sampling 240M.

GWAS are expensive endeavours and at the current rate of exponential improvement, their results would only be relevant for 18 months before being superseded by newer studies. To extend the relevance of the data gathered during GWAS, a methodology to statistically infer predicted markers based on results from the latest GWAS was derived. This methodology is known as genomic imputation.

5.3.1 Methodology

Here we investigate a distributed, event-driven approach to genomic imputation. The underlying methodology is based on a customised version of the forward/backward algorithm which is used to solve a 2-dimensional Hidden Markov Model (HMM). The HMM forms the most computationally expensive part of the model and is directly dependent on the number of participants and the number of markers sampled in the GWAS study. As these are the metrics growing exponentially, genomic imputation is beginning to test the computational feasibility of x86 execution, with wall-clock runtimes now measured in days/weeks.

In POETS, each state in the HMM was given its own hardware thread and the POETS messaging solely used the accelerated local multicast functionality. Problem sizes up to the hardware thread count of the current cluster were run and the wall-clock runtimes were compared to an original single threaded x86 implementation. The results demonstrated a consistent increase in comparative speed up. Each hardware thread was then permitted to govern multiple HMM states and problem sizes up until memory exhaustion in the current cluster prevented further increase in problem size. An optimal region of around 10 states per hardware thread was identified. Beyond this region, the distributed implementation still provided an impressive speed up compared to the single threaded x86 version. Both of these scaling graphs are shown in figure 5.3.1.a.

¹⁸ Richard Sinke "Targeted Next-Generation Sequencing can Replace Sanger Sequencing in Clinical Diagnostics" Academia, doi: 10.1002/humu.22332

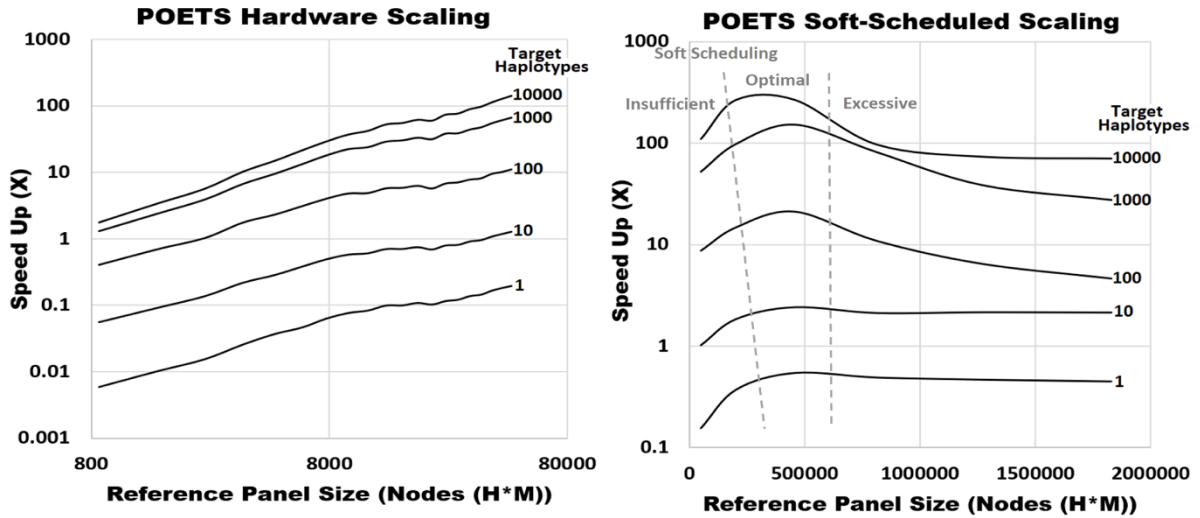


Figure 5.3.1.a: Hard scaling and soft-scheduled POETS distributed genomic imputation scaling graphs.

Genomic imputation has been investigated in the field for around 2 decades. Over that period of time, several optimisations have been investigated in the x86 domain. The optimisation with the greatest impact has been linear interpolation. This permits states within the HMM that do not have an annotated base from the original data to be dropped from the HMM and linear interpolation to be performed to estimate their value. This can reduce the complexity of the HMM significantly with only a negligible impact on the accuracy of the imputation. Linear interpolation was added into both the POETS distributed algorithm and the x86 baseline. Each hardware thread governed a single HMM state plus several linear interpolation states. Results demonstrated a continuing comparative increase in speed up over the single-threaded x86 baseline until memory exhaustion of the current cluster. This may be seen in the scaling graph below.

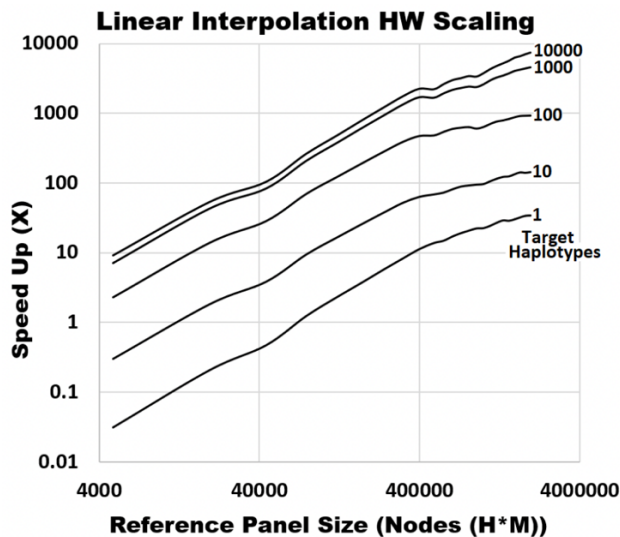


Figure 5.3.1.b: Genomic imputation scaling on POETS.



5.4 Micromagnetic Modelling

Hard drives magnetise regions of the space either "up" or "down" into magnetised domains. Spintronics research over the last two decades has realised several alternative potential data carriers, including skyrmions and magnons, which are both considerably smaller in size (50nm to atom spacings), and require less energy to drive through a material (10^{-6}Am^{-2} to 10^{-12}Am^{-2}). If these properties can be exploited, the size and power requirements of data storage and memory devices can be considerably improved.

Present computing technology imposes a practical restriction on the size and fidelity of these models, restricting the scope for designing spintronic devices. However, the POETS approach is particularly effective here, as the physics lends itself well to massively-parallel event-driven computation.

The micromagnetic model, a mathematical model, can be used to model the behaviour of these data carriers. We first introduce this model, then explain how this model has been adapted to a POETS environment for massive performance improvements.

5.4.1 The problem

Given a three-dimensional space \mathbf{x} , the magnetic material is broken up into sub-domains using a regular finite-difference discretisation. Figure 5.4.a shows a skyrmion broken up in this way - each cone represents the magnetisation (a three-dimensional vector of fixed length, $\mathbf{m}(\mathbf{x})$) in each of these sub-domains. Micromagnetic behaviour is often dominated by short-ranged interactions, including isotropic exchange (which favours aligned magnetisation):

$$w_{\text{exch}}(\hat{\mathbf{m}}) = A(\|\nabla m_x\|^2 + \|\nabla m_y\|^2 + \|\nabla m_z\|^2),$$

and Dzyaloshinskii-Moriya (DM) exchange (which favours perpendicular magnetisation):

$$w_{\text{dmi}}(\hat{\mathbf{m}}) = D\hat{\mathbf{m}} \cdot (\nabla \times \hat{\mathbf{m}}).$$

Skyrmions also usually require an external magnetic field \mathbf{H} for stability:

$$w_{\text{zeeman}}(\hat{\mathbf{m}}) = -\mu_0 M_s \hat{\mathbf{m}} \cdot \mathbf{H}.$$

Here, A , D , \mathbf{H} , and μ_0 all determine the relative strengths of these interactions. These energy densities all contribute to the dynamics of the system, as the magnetisation changes in time to minimise total energy.

In POETS, each device represents the magnetic behaviour in one sub-domain from the finite-difference discretisation. The derivatives in the isotropic and DM exchange terms are computed rapidly by exploiting POETS' fast communications network. Each device moves forward as fast as its neighbours will allow, using a globally-asynchronous, locally-synchronous (GALS) communication pattern. In brief:

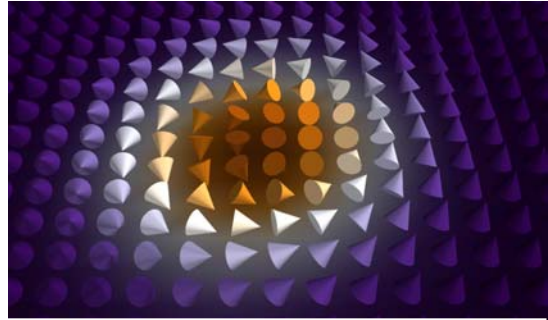


Figure 5.4.a: A simulated micromagnetic skyrmion. Each cone represents the magnetisation at different positions in the skyrmion.



- Messages contain the magnetic moment vector of the sender (three floating point numbers), and an iteration number.
- When a device receives a message, it stores the quantities in a double-buffer: one for the current iteration, and one for the next iteration, for the GALS synchronisation pattern.
- After receiving each message, if it knows all of its neighbours magnetic moment vectors for this iteration, it updates its state by integrating forward in time (by one iteration), and bursts the new state to its neighbours.
- This process repeats until a certain simulated time (order of nanoseconds) has passed, at which point the device sends its final magnetic moment vector to the Supervisor, which then writes the result to a file on the disk.

The problem considered here is a skyrmion relaxation problem:

- What are the bounds on the strength of the external magnetic field that can support a skyrmion?
- How does the magnetic field strength affect the radius of the skyrmion?

Such an analysis is used to determine the viability of a given material for use in spintronic hardware - an expensive process if attempted physically.

5.4.2 Results

Figure 5.4.2.a shows how a POETS-based micromagnetic simulator compares with a simulation package (Fidimag) on traditional hardware, as a function of problem size. POETS outperforms the traditional approach by orders of magnitude, supporting parameter sweeps and more advanced investigations. The figure also shows how the performance of the POETS-based approach has improved since the previous advisory board. This is largely down to improvements in the placement approach, the development of the underlying hardware, and algorithmic improvements. These changes have also resulted in a **33% reduction** in instruction space utilisation, making it easier to support other interactions like magnetocrystalline anisotropy and demagnetisation, and making it possible to support instrumentation and other development tooling.

For completeness, figure 5.4.2.b shows how skyrmion formation is affected by external magnetic field strength in this material. The results from the POETS simulator agree with those run on traditional hardware. The results from the 2019 simulation work have been submitted for publication in IET Computers and Digital Techniques, and is currently under review.

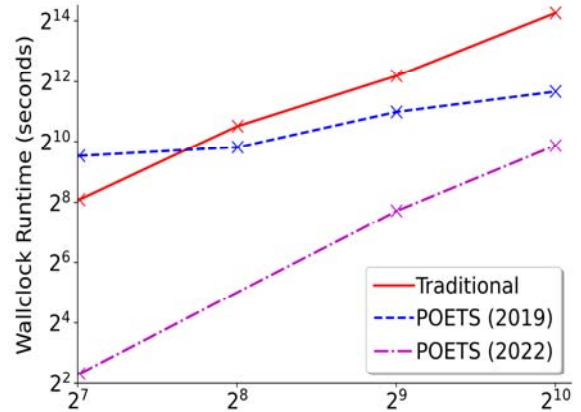


Figure 5.4.2.a: Execution times for POETS past and present, and a simulation package on traditional hardware. Each point is an average of three timed simulations, and represents execution for eight different external magnetic field strengths.

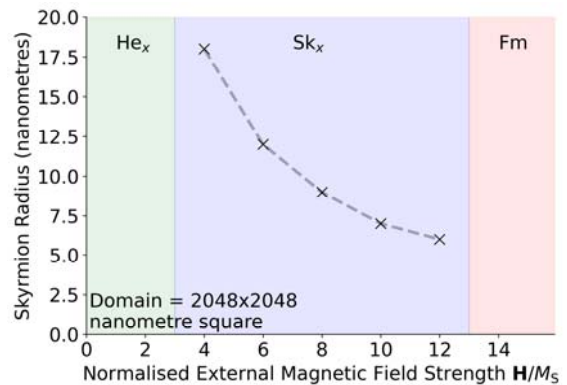


Figure 5.4.2.b: Skyrmion radius as a function of external magnetic field strength. Too strong, and the material becomes uniformly magnetised. Too weak, and a disordered helical configuration emerges as the solution.



5.5 Reactive systems

The canonical example used at the outset of the research is the heat equation, which is arguably the most well-behaved differential equation known. Specifically it contains no reactive components, so whilst it can accurately describe the behaviour of systems in thermal equilibrium, it cannot describe any systems containing any kind of thermal inertia (non-zero heat *capacity*).

Physical domains that can be described in terms of node potentials and edge flows are ubiquitous: electrical (voltage, current); magnetic (magnetic potential, flux); thermal (temperature, heat flow); mechanical (force, movement). Mixed domain (as opposed to mixed-mode) simulation systems capitalise on this, because the underlying equations are identical. The system descriptions, mapped onto parameter spaces that are familiar in reality, change the appearance of the equation sets, but this is superficial.

For this application domain we present the use of POETS to solve first order differential (reactive) systems. The analysis is presented in terms of the electric domain, for two reasons: (1) it is well within the comfort zone of the investigators, and (2) commercial simulators are common, so comparison of results is easy.

The analysis is presented in Appendix A in some detail, because it illustrates the point made earlier that it is necessary to strip an analysis back to the fundamental mathematics before recasting to fit the POETS computation model, and to demonstrate the validity of the POETS technique in numerical methods.

In this, the body of the report, we restrict the description to a statement of a test circuit, and a comparison of the runtimes between a POETS based simulation, and that of SPICE - a common analogue circuit simulator.

The goal of analogue system simulation is to generate the complete time history of all node voltages in some specified time interval. Conventional approaches use a march-in-time Newton-Raphson scheme, that discretises the circuit in time and solves for the time history as a sequence of state vectors, where each element n th vector corresponds to a specific node.

The POETS approach uses a pure, unsynchronised packet storm. One POETS device is assigned to each node *for each time point*. This quickly becomes extremely expensive in device count, but the mantra of event-based compute has always been that devices are of negligible cost - "free".

The analysis proceeds as a simple relaxation solution of the circuit, with the relaxation iterations permitted to extend forward in time, as well as over the circuit at a time point.

5.5.1 Demonstration circuits

The first circuit (figure 5.5.1.a) is algorithmically unchallenging; it is presented here to demonstrate the numerical correctness of the technique. Figure 5.5.1.b shows the time histories of each node in the circuit generated by both SPICE and POETS - they are numerically indistinguishable, which gives us some confidence that the POETS technique is valid. The circuit is displayed as an electronic schematic; it corresponds exactly to a one-dimensional heat-plate with non-zero thermal capacity.

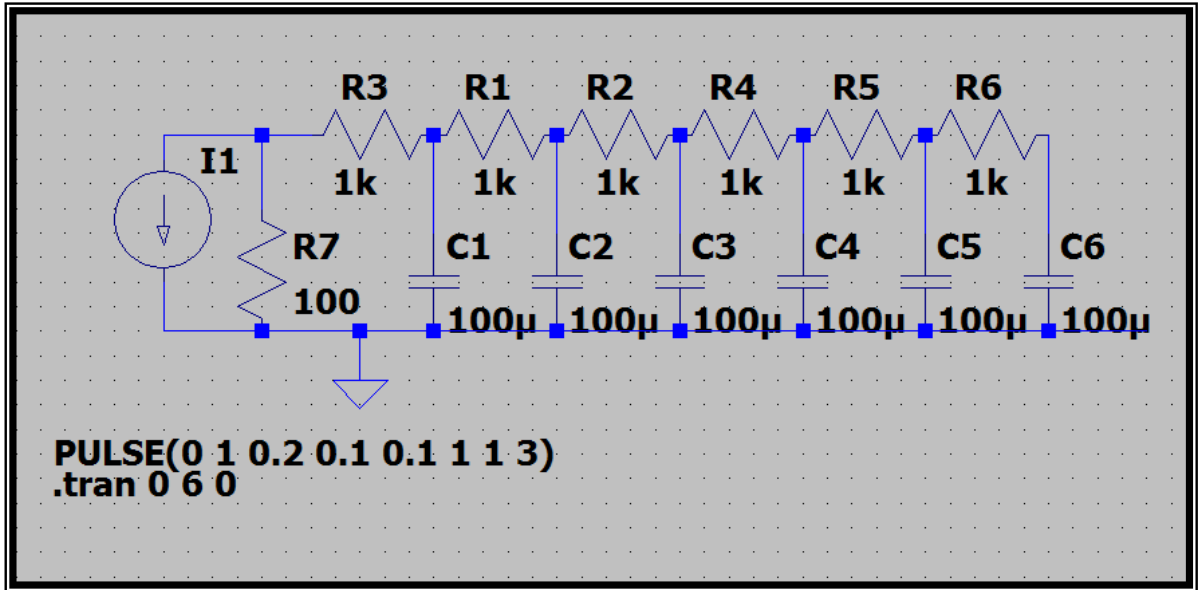


Figure 5.5.1.a: Simple test circuit.

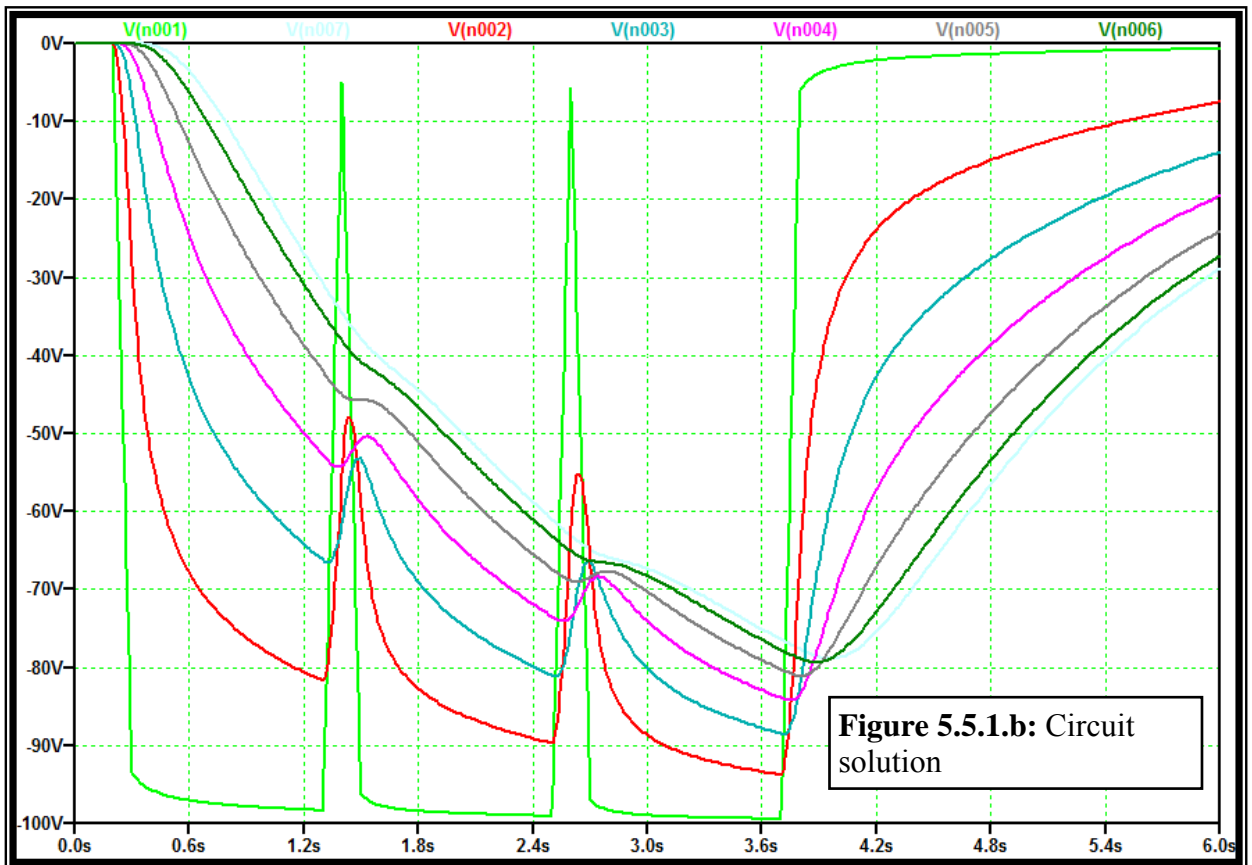


Figure 5.5.1.b: Circuit solution.



An idealised version of the second circuit is shown in figure 5.5.1.c. This extends the circuit of figure 5.5.1.a into two spatial dimensions (making a 3D structure when the time dimension is included). The size of the grid (electrical node count) is used as the independent variable in the graph of figure 5.5.1.d.

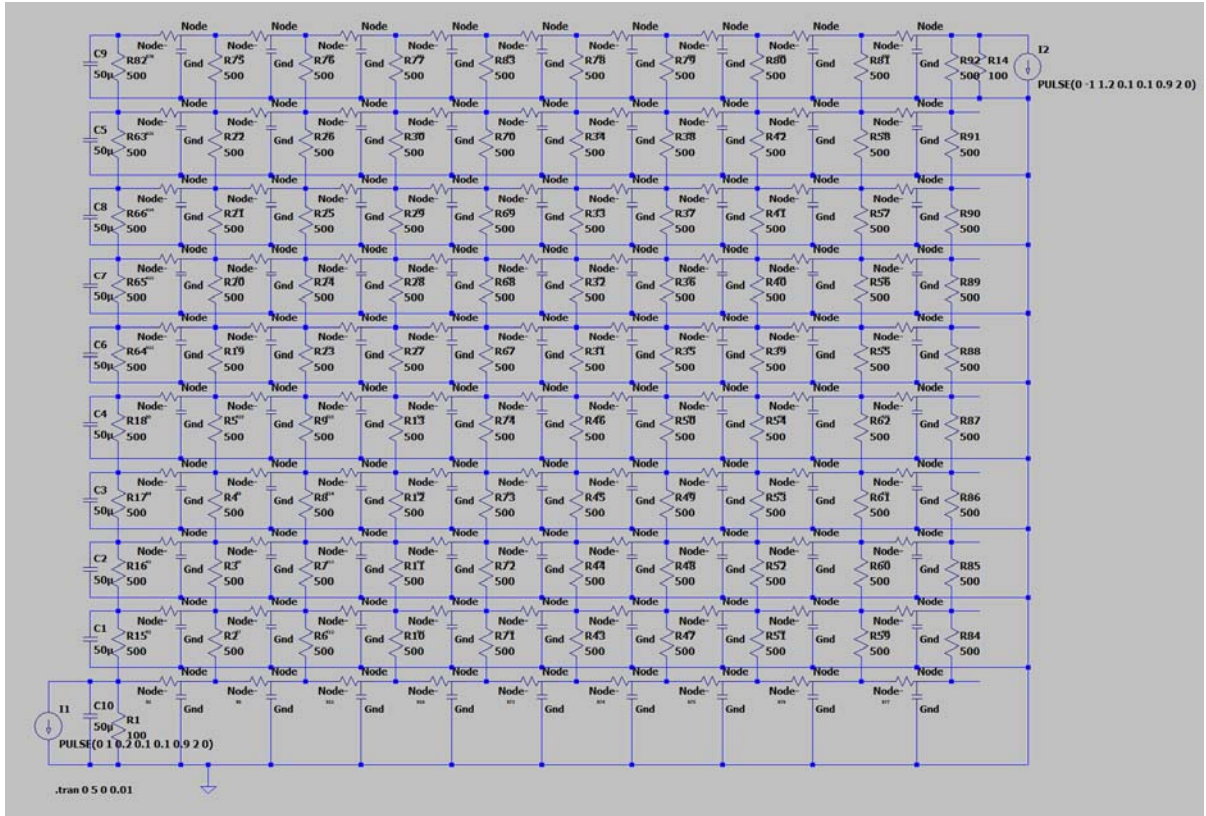


Figure 5.5.1.c: Distributed reactive "surface".

Each simulation covers five seconds of simulated time with a 0.01 second timestep. Each grid point has a total of 501 distinct time points and each time point is represented by a single POETS device. Grids from 5x5 (12,525 devices) to 200x200 (20,040,000 devices) were simulated on the full 8-box cluster.

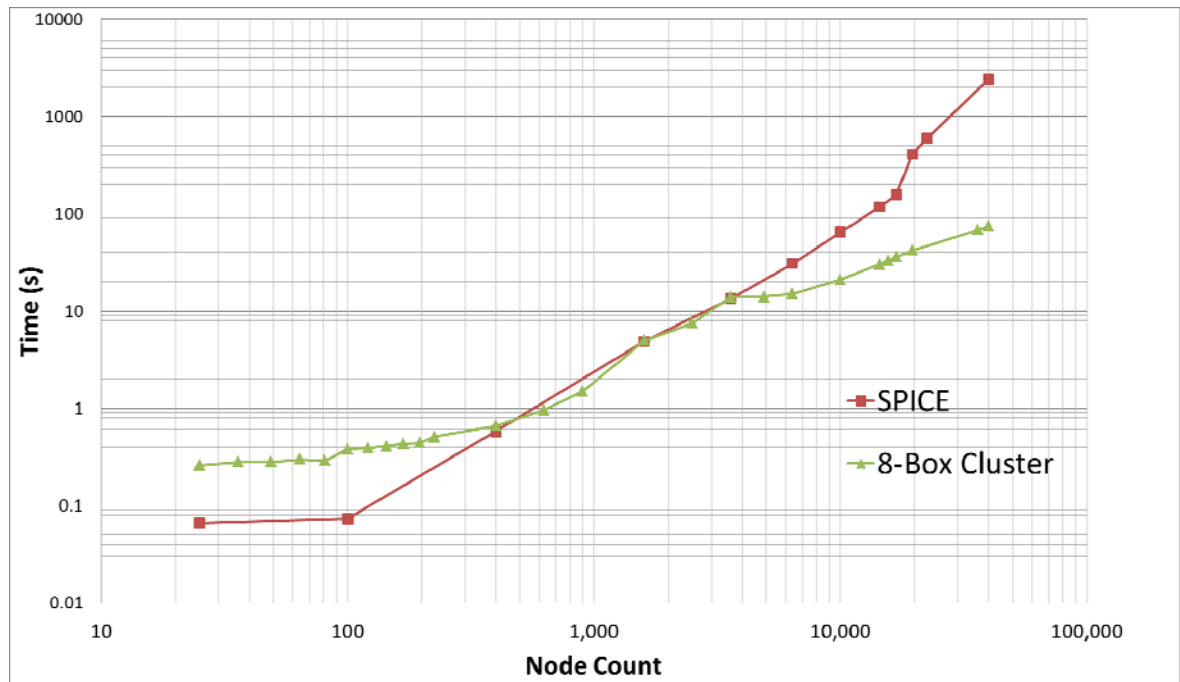


Figure 5.5.1.d: Wallclock runtime .vs. circuit node count.



POETS exhibits desirable scaling (figure 5.5.1.d) with *runtime only increasing by two orders of magnitude when the node count is increased by three orders of magnitude*. In comparison, the runtime of the SPICE exemplar increases by five orders of magnitude for the same increase in node count. For a 200x200 problem, POETS is 32x faster than SPICE.

Near-linear scaling is demonstrated when one or two devices are allocated to each hardware thread (i.e. up to a 14x14 grid with 196 nodes for a total of 98,196 devices). With larger grids, more devices are allocated to each thread and the performance scales less desirably due to increasing serialisation. Plotting the number of devices-per-thread over the runtime against the node count (figure 5.5.1.e) shows a fairly flat profile. This indicates that the near-linear scaling seen for smaller problems (up to 14x14) can be achieved for larger problems by adding additional POETS boxes.

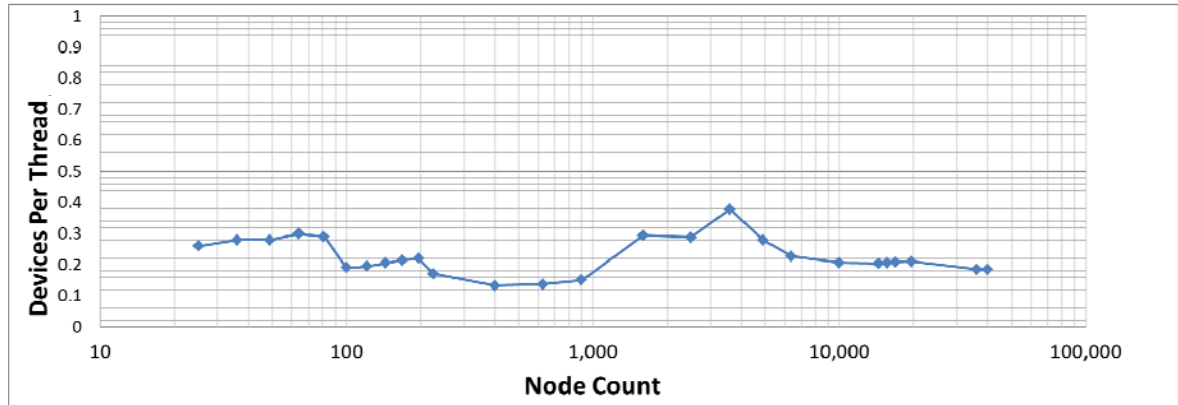


Figure 5.5.1.e: Core loading for the POETS runs of figure 5.5.1.d.

5.5.2 Application

This section illuminates how POETS may be used to simulate reactive continuous systems, exploiting parallelism in both time and space. The circuits used here are first order, but the extensions necessary to accommodate second order {L,C,R} circuits are not mathematically challenging.

Aside from the obvious application to reactive thermal circuits (physical systems of *arbitrarily complex geometry* with a non-zero thermal capacity), the driver for this analysis came from a BEIS sponsored "Unmet needs by industry" meeting. The speaker was from a company that makes mobile high-powered radar systems (mobile in the sense that they are mounted on a truck). These systems require high voltages (10..100kV) to operate. Not in itself challenging, but the *regulation* on these supplies is ferocious: sub millivolt, to the point that they care about electric-wind mediated electron radiation from sharp corners of the physical structure, making analyses of the type outlined here extremely relevant.



5.6 Partial discharge

High voltage (>MV) power distribution is becoming ever-more societally important, and the design and manufacture of reliable high voltage cables capable of transporting GW of power through inaccessible environments is far from a mature discipline. One problem - amongst many - is the phenomenon of **partial discharge** within HV cables. This is frequently a precursor to a catastrophic breakdown, and is the subject of much research.

Partial discharge may be thought of as a (relatively) slow-growing discharge, from the inner to the outer core of a cable. Post-mortem cable dissection reveals these to resemble tree-like structures, not dissimilar to a spark, frozen in time and space. The precise underlying physics of this phenomenon is currently unknown, and material scientists need an understanding of this before they can design cables that are resistant to it. One of the barriers faced is that the state-of-the-art simulation techniques require the use of 3D plasma dynamics models, which have a prodigious computational cost.

The work is described here because it is compelling, relevant and a beautiful fit for POETS. Time constraints did not allow it to be taken to the point of generating results, but we do not intend work on POETS to end at the termination of this project.

5.6.1 The POETS model

The POETS cable model is outlined in figure 5.6.1.a. It is a two-dimensional canonical decomposition of a cable section (prima facie, we ignore axial fields, but these are an obvious step if preliminary work is successful) into discrete resistive components. In practise, the insulating sheath is composed of a low-conductivity (usually plastic) layer. The conductance as a function of radius is carefully controlled, starting with an inner layer of

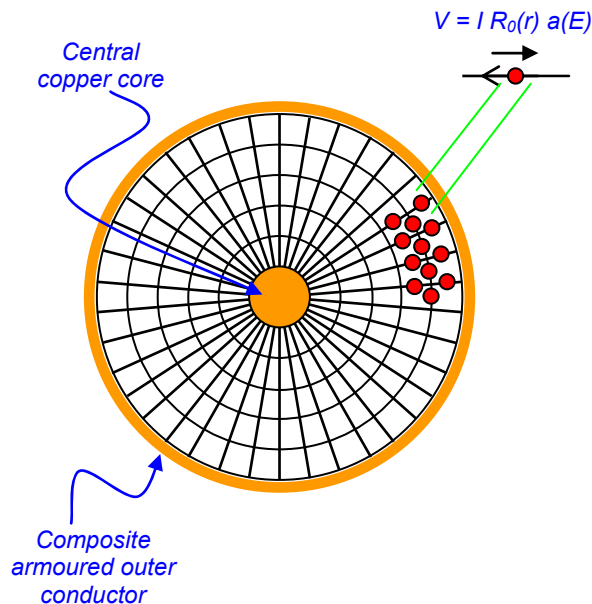


Figure 5.6.1.a: POETS cable model.

relatively high conductance (to smooth out any field discontinuities arising from manufacturing artefacts), and then quickly becoming as low as is possible to make. Confusingly, this inner zone of the insulator is referred to as *semiconducting*.



The inner and outer conductors are assumed to be isopotential, which means that initially the tangential grid elements have no field over them.

The model consists, then, of a grid of resistance elements whose base resistance R_0 is a well-behaved function of distance from the core. The functional complexity of the system arises from the non-linear hysteretic multiplier $a(E)$, which can switch irreversibly between two values a_H and a_L , when the potential over the element exceeds some threshold. (Which can cause the tangential elements to experience a non-zero potential difference). The nature of the threshold is defined by some random phenomenological field that pervades the system.

The entire system is characterized by a set of parameters describing, amongst other things, principally the non-linear resistances and the probabilistic nature of the pervasive breakdown field, and the spatial semiconduction attributes of the insulating sheath. Under what combination of circumstances does a breakdown discharge tree develop? Extremely preliminary experiments show that tree-like partial breakdowns are quite easy to simulate, but we must beware of the eternal trap of simulation: just because the mathematics looks like reality, it doesn't mean it necessarily represents the physics. An essential test is the ability of the model to predict outcomes unknown a priori.

The problem rapidly becomes that of parameter space exploration: embarrassingly parallel. This by no means diminishes its value: embarrassingly parallel problems are important: in an unknown highly non-linear parameter space initial exploration is essential to locate regions that merit further study. (This is a big part of the dissipative particle dynamics work presented in a later section.)

The modelling work will be taken forward under the aegis of a National Grid research program.



5.7 Large-scale Petri Net simulations

With the advance of digital computing, there has been a growing demand for modelling very large and complex systems, a few examples being circuit design, developmental biology, and modelling machine-learning automata. These systems can all have millions of interacting elements, putting them out of range of conventional simulation techniques. **Petri nets (PNs)** is a formalism that can be used to underpin the design and analysis of these examples. This demonstrates the usefulness of PNs as an *interpreted graph model* for exploring current and future extremely large concurrent systems.

At this scale, the models are no longer hand-crafted but automatically synthesized from libraries of components. While the constituent components are small enough to be formally verifiable, the statistical quantitative properties as well as the emergent behaviour of the system as a whole call for different kinds of exploration techniques, such as simulation. But even then, millions of places and transitions in a PN present a significant time cost for each simulation run on a single CPU thread.

Even though PNs are well suited for describing concurrency, simulating them does not immediately lend itself well to parallelism, as evidenced by the lack of parallel PN simulators used in the field. In the POETS project, we created a practical implementation of a PN simulator on the POETS platform and made the following scientific contributions in the process:

- Developed a new and general algorithm for distributed PN simulation.
- Formally described a partitioning approach that enables purely forwards communication to resolve choices.
- Implemented and tested this approach on the POETS platform. Experimental results show that an 8-box POETS cluster is 45 to 220 times faster than a single-core simulator.

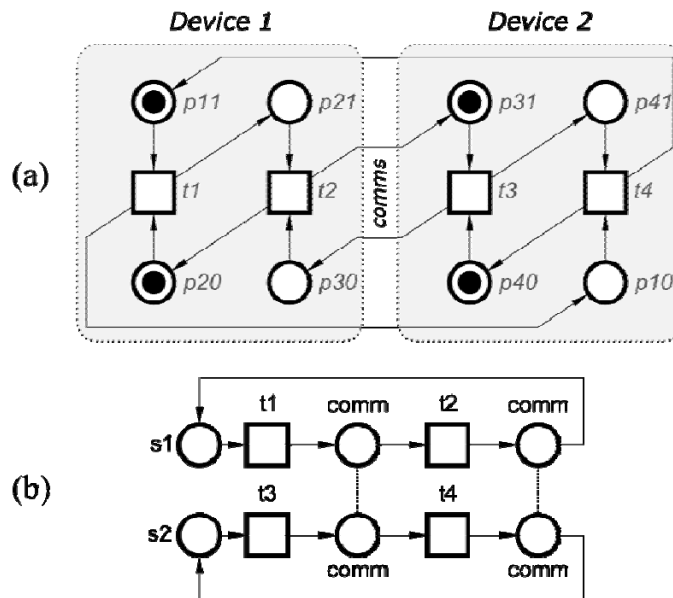


Figure 5.7.a: (a) PN example distributed over two execution devices and (b) corresponding flow networks synchronized at comm states.

The core idea of the developed algorithm is that a firing of a transition, which is an atomic action in the classical definition of a PN, is split into two phases: firing phase and update phase. The firing phase is equivalent to executing a preset subnet of the fired transition



(or transitions). The update phase adds back the tokens removed during the firing phase to create a new valid marking. When distributed over multiple devices, the algorithm executes firing phase locally on each device and performs communications during the update phase. We use the hardware-implemented refutable barrier feature of the POETS to ensure efficient global synchronization at this phase. The example is shown in figure 5.7.a.

For the performance evaluation experiments, we simulated up to 50 million places PN for 10,000 simulation steps in different types of PN execution semantic. We slice the models to have approximately 1,000 places per logical device. Figure 5.7.b shows the performance and scaling of the simulation running on just one POETS box. For larger net sizes, we put more than one logical device on each hardware thread. The effect is visible on the graphs as "bumps". Figure 5.7.c provides the experimental evidence for scaling to multiple POETS boxes. Single-core CPU performance results (obtained from 3.50GHz Intel Core i7-7800X) are given as a reference. Locally-interleaving execution may seem faster, however, the total number of transitions fired is smaller than for max-step semantics. The latter represents much further evolution of the PN within the same number of simulation steps – millions or billions of transition firings.¹⁹²⁰²¹²²

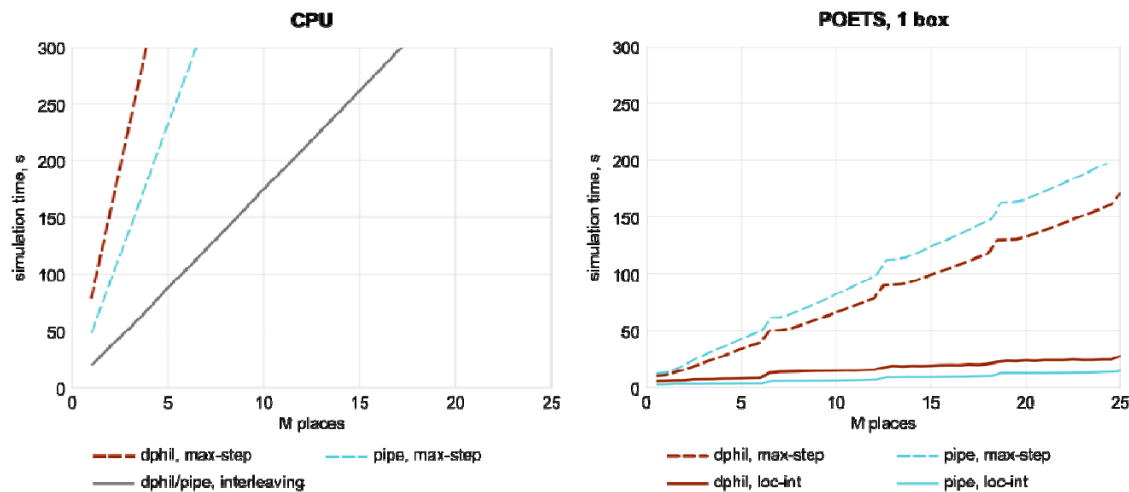


Figure 5.7.b: Performance results for Muller pipeline and dining philosophers benchmarks in different execution semantics on a single POETS box (up to 6144 parallel hardware threads) compared to a single thread CPU simulation.

¹⁹ Ashur Rafiev, Jordan Morris, Fei Xia, Alex Yakovlev, Matthew Naylor, Simon Moore, David Thomas, Graeme Bragg, Mark Vousden, Andrew Brown, "Practical distributed implementations of very large scale Petri net simulations": in ToPNoC XVI, LNCS 13220, pp112-139, 2022 https://doi.org/10.1007/978-3-662-65303-6_6.

²⁰ M. Koutny, M. Pietkiewicz-Koutny, and A. Yakovlev, Asynchrony and persistence in reaction systems, Theoretical Computer Science, Volume 881, 2021, pp 97-110, ISSN 0304-3975, <https://doi.org/10.1016/j.tcs.2020.11.040>.

²¹ M. A. N. Al-hayanni, F. Xia, A. Rafiev, A. Romanovsky, R. Shafik and A. Yakovlev, "Amdahl's law in the context of heterogeneous many-core systems - a survey," in IET Computers & Digital Techniques, vol. 14, no. 4, pp. 133-148, 7 2020, doi: 10.1049/iet-cdt.2018.5220.

²² V. Khomenko, M. Koutny, and A. Yakovlev (2022). Avoiding Exponential Explosion in Petri Net Models of Control Flows. In: Bernardinello, L., Petrucci, L. (eds) Application and Theory of Petri Nets and Concurrency. PETRI NETS 2022. Lecture Notes in Computer Science, vol 13288. Springer, Cham. https://doi.org/10.1007/978-3-031-06653-5_14

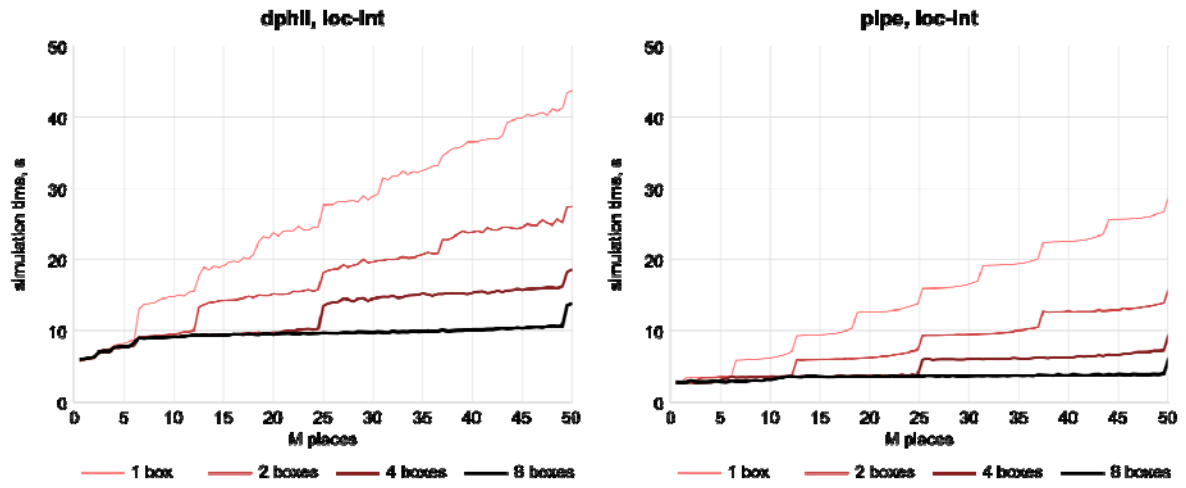


Figure 5.7.c: Locally-interleaving semantics performance results for multiple POETS boxes: Muller pipeline benchmark (right), dining philosophers benchmark (left).



5.8 Machine learning using Tsetlin Machines

The Tsetlin Machine (TM) is a nascent machine learning algorithm that uses binarized input vectors and derives sub-patterns from a given entropy function. Binary literals from the input vector are accumulated in conjunctive clauses which in turn vote for a given outcome. These votes are then accumulated per class in order to resolve classification problems. This mechanism has been extended to other machine learning tasks such as regression and text categorization.

The problem of classification is one of the most typical use cases for machine learning. Given an input Boolean vector X , the goal of the machine is to correctly identify which of the L classes is the best fit for this input. A Tsetlin Machine is an automata-based classifier system that uses conjunctive clauses in propositional logic with this objective. Single-class TM matches the input X to a given class and produces an integral score v , called the *class sum*, which can be positive or negative. The score represents how confident the single-class TM is about the match between the input and the class. A multi-class TM is a team of L single-class TMs joined by majority voting that decides the best-fit class. Figure 5.8.a shows the inference part of the architecture of such a machine.

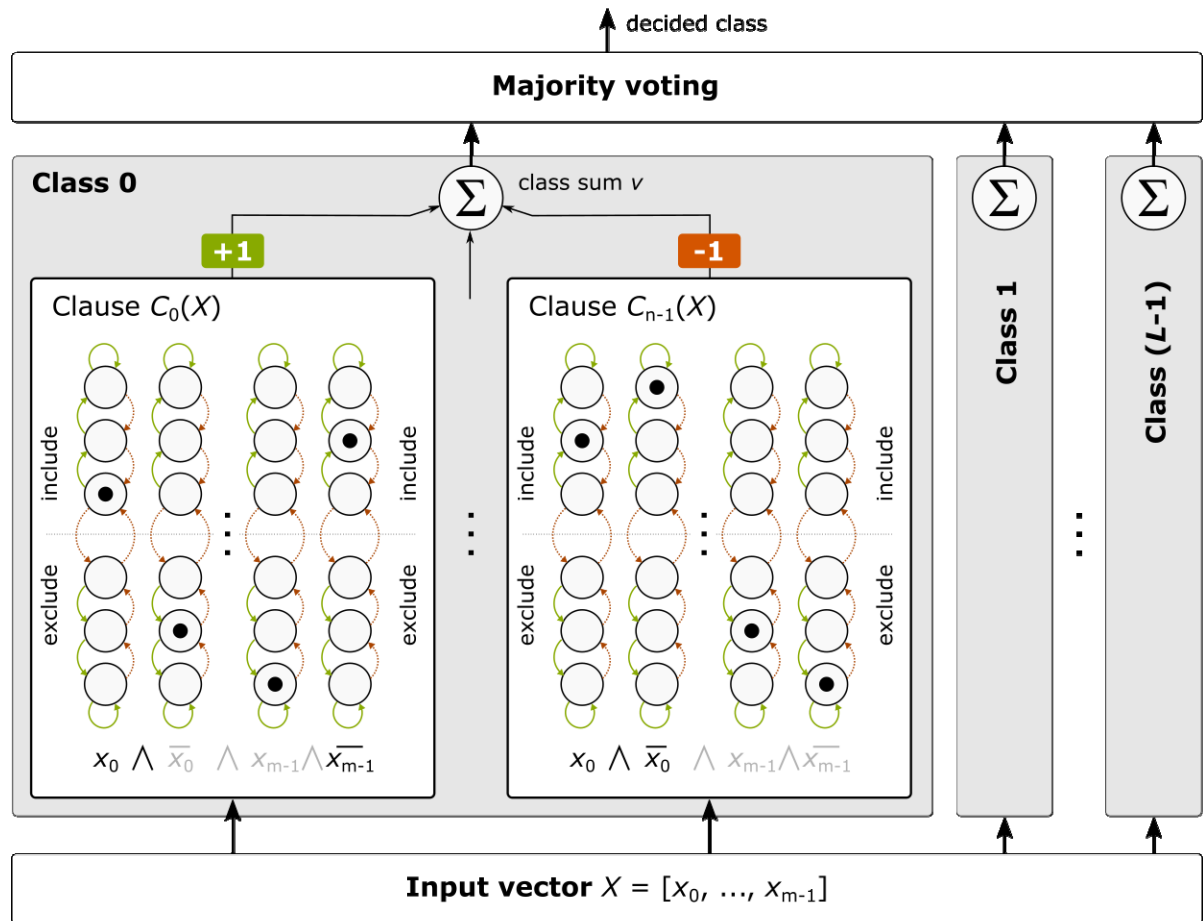


Figure 5.8.a: Multi-class Tsetlin Machine inference architecture.



In the original Tsetlin Machine algorithm, the accumulated class sums are used within the feedback mechanism to stimulate sub-pattern optimisation and allow the algorithm to eventually converge. The drawback to this is that it requires inter-clause communication in an otherwise outright parallelizable algorithm. In the POETS project²³²⁴, we proposed an alternative algorithm that monitors and limits the literals that form the sub-pattern within a clause. This "literal count" is then used to stimulate sub-pattern optimisation and converge the algorithm, replacing the class sum and allowing the clauses to run independently.

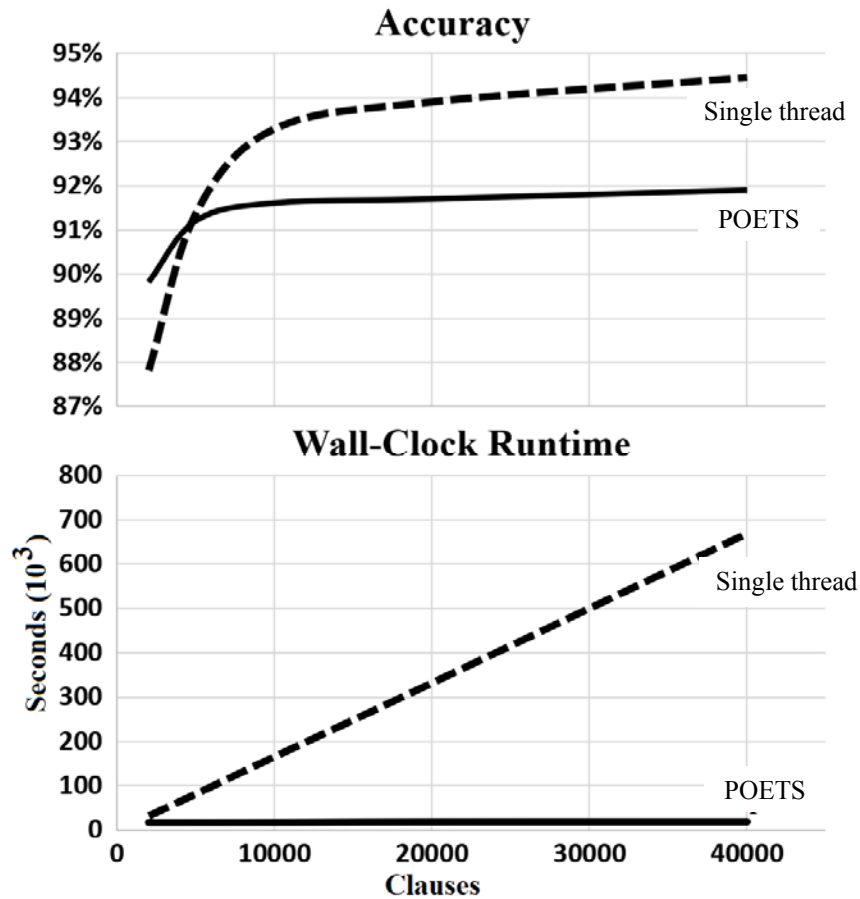


Figure 5.8.b: Accuracy and wall-clock time over clause scaling.

Figure 5.8.b compares the proposed algorithm to the original TM solution in terms of the accuracy and wall-clock runtime over varying clause counts. The original algorithm was run on an Intel i9-7940X CPU @ 3.10GHz. The proposed algorithm was run on the POETS cluster. The results demonstrate a minor loss of 2.5% in accuracy for 40K clauses but a reduction in wall-clock runtime of 36X. The difference in accuracy is likely due to the superior ability of the original algorithm to identify outliers in the dataset.

²³ Ashur Rafiev, Jordan Morris, Fei Xia, Rishad Shafik, Alex Yakovlev, Ole-Christoffer Granmo, and Andrew Brown, "Visualization of Machine Learning Dynamics in Tsetlin Machines" in International Symposium on the Tsetlin Machine (ISTM), IEEE, 2022

²⁴ Jordan Morris, Ashur Rafiev, Fei Xia, Rishad Shafik, Alex Yakovlev and Andrew Brown, "An Alternate Feedback Mechanism for Tsetlin Machines on Parallel Architectures" in International Symposium on the Tsetlin Machine (ISTM), IEEE, 2022



5.9 High-energy physics

POETS research has contributed to two novel optimisations of machine learning used in high-energy physics targeting reconfigurable hardware implementation, in collaboration with Researchers at CERN and Caltech.

The first optimisation concerns a new reconfigurable architecture for reducing the latency of recurrent neural networks (RNNs) that are used for detecting gravitational waves²⁵. Gravitational interferometers such as the LIGO detectors capture cosmic events such as black hole mergers which happen at unknown times and of varying durations, producing vast quantities of time-series data. We have developed a new architecture capable of accelerating RNN inference for analyzing time-series data from LIGO detectors. This architecture is based on optimizing the initiation intervals (II) in a multi-layer LSTM (Long Short-Term Memory) network, by identifying appropriate reuse factors for each layer. A customizable template for this architecture has been designed, which enables the generation of low-latency FPGA designs with efficient resource utilization using high-level synthesis tools. The approach has been evaluated based on two LSTM models, targeting a ZYNQ 7045 FPGA and a U250 FPGA. Experimental results show that with balanced II, the number of DSPs can be reduced up to 42% while achieving the same IIs. When compared to other FPGA-based LSTM designs, our design can achieve about 4.9 to 12.4 times lower latency.

The second optimisation concerns a new architecture for reducing the latency of JEDI-net, a Graph Neural Network (GNN) based algorithm for jet tagging in particle physics, which achieves state-of-the-art accuracy²⁶. Accelerating JEDI-net is challenging since it requires low latency to deploy the network for event selection at the CERN Large Hadron Collider. Our work here concerns an outer-product based matrix multiplication approach customized for GNN-based JEDI-net, which increases data spatial locality and reduces design latency. It is further enhanced by code transformation with strength reduction which exploits sparsity patterns and binary adjacency matrices to increase hardware efficiency while reducing latency. In addition, a customizable template for this architecture has been designed and open-sourced, which enables the generation of low-latency FPGA designs with efficient resource utilization using high-level synthesis tools. Evaluation results show that our FPGA implementation is up to 9.5 times faster and consumes up to 6.5 times less power than a GPU implementation. Moreover, the throughput of our FPGA design is sufficiently high to enable deployment of JEDI-net in a sub-microsecond, real-time collider trigger system, enabling it to benefit from improved accuracy.

²⁵ Z. Que et al., Accelerating Recurrent Neural Networks for Gravitational Wave Experiments, IEEE International Conference on Application-specific Systems, Architectures and Processors, 2021.

²⁶ Z. Que et al., Optimizing Graph Neural Networks for Jet Tagging in Particle Physics on FPGAs, International Conference on Field Programmable Logic and Applications, 2022.



5.10 Machine learning on a national scale

The previous domain specific applications demonstrate that:

- Due to typically sub-linear scaling with problem size, POETS lends itself well to **large** problems, and can solve them orders-of-magnitude more quickly than traditional approaches.
- POETS lends itself well to **complex** problems: problems that can be broken up into many small parts, where those small parts interact frequently to produce an emergent solution.

These two characteristics, large and complex, have been used to guide the "exploitation" component of POETS' research trajectory. The Coronavirus pandemic, which hit the UK with full force in 2020, inspired us to turn POETS to national-scale disease modelling, as it is a very large problem governed by dynamics that are inherently complex. While we were not able to conceive a high-TRL solution for EPSRC's Rolling COVID-19 scheme, we identified epidemiological modelling as an area of national interest. From there, we identified the potential of POETS to work on machine learning problems, which has implications for modelling financial risk, the creation of real-time digital twins, general combinatorial optimisation, and more.

In addition to being large and complex, national-scale models typically introduce themes of **irregularity** - their problems are typically unstructured (e.g. networks of people), and their interactions spasmodic and intermittent. Such characteristics had, thus far, not been deeply explored with POETS, though SpiNNaker has demonstrated that event-driven computing works well solving irregular problems in the neuromorphic computing space.

Two national-scale model proof-of-concept studies are presented here to demonstrate this idea: A simplified epidemiological approach, based on the traditional SIR compartmental mechanism, to model the spread of disease across a population, and a graph-based machine-learned inference machine, using Graph Convolutional Networks. Both of these case studies model **complex** behaviours, require understanding of **irregular** domains and communication patterns, and have real value at **large** scales.

5.10.1 SIR: A simplified epidemiological model

The Susceptible-Infected-Removed (SIR) epidemiological model has seen reasonable success in predicting the spread of diseases like measles, where recovery inhibits reinfection. It is comprised of three compartments:

- Susceptible: Members of the population that are susceptible to the disease, if brought in contact with an infected individual.
- Infected: Currently infected by the disease, and act as carriers.
- Removed: Individuals that have either recovered from the disease, or have perished from its effects. This model assumes that the number of deaths is negligible.

Macroscopic compartmental models use differential equations to show how members of the population transition from sustainable to infected, and from infected to recovered, over time. However, here we propose a spatial model, where the population exists on a graph, and each member of the population has state in $\{S, I, R\}$, as shown in Figure 5.10.1.a. In this spatial model, a susceptible individual's probability of infection is a function of the number of infected individuals around it, and the time to recovery is fixed.

As the problem domain is already a graph, the most obvious POETS approach maps each individual onto a device, and each edge of the problem graph corresponds to one edge connecting two devices together. Devices share their state (S, I, or R) with their neighbours when they step forward in time, and the simulation ends when all individuals are recovered.



At that point, all devices exfiltrate their "transition times" to the Supervisor, who reconstructs the emergent story from these transition data.

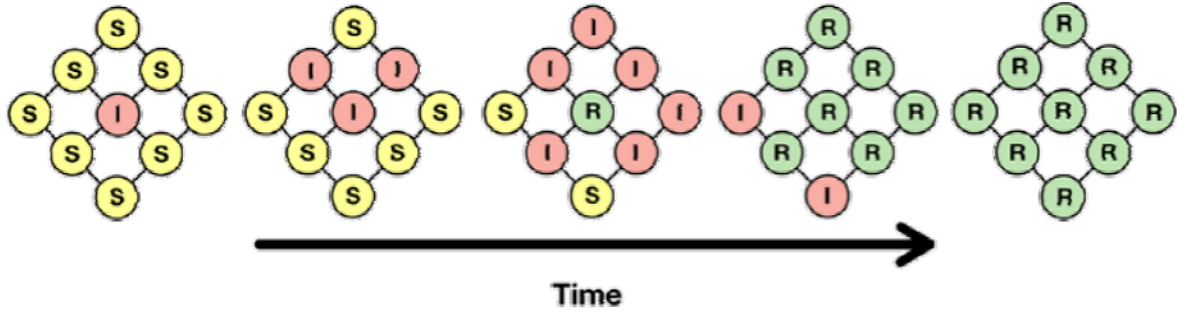


Figure 5.10.1.a: An example nine-person population in a spatial SIR model, with one initially-infected individual. The infection spreads as time passes, and infected individuals are removed after a recovery period.

As the problem domain is already a graph, the most obvious POETS approach maps each individual onto a device, and each edge of the problem graph corresponds to one edge connecting two devices together. Devices share their state (S, I, or R) with their neighbours when they step forward in time, and the simulation ends when all individuals are recovered. At that point, all devices exfiltrate their "transition times" to the Supervisor, who reconstructs the emergent story from these transition data.

Figure 5.10.1.b compares how a traditional comparator (single-thread - Februus) compares with POETS, with two different, relatively naive, device-placement schemes. Note the vertical axis in this figure is not the total execution time, but the execution time divided by the total number of iterations, as larger problems inherently take longer to converge. Using this metric, POETS scales sub-linearly with problem size, until the compute fabric becomes saturated and the application serialises. A larger POETS engine would allow larger problems to be modelled without this compromise of performance.

This proof-of-concept will be extended, to explore how topological properties of the graph (e.g. degree distribution) affect this performance comparison, and the effect of placement on such. This study, together with its extension, will be submitted as a publication in IEEE Computer.

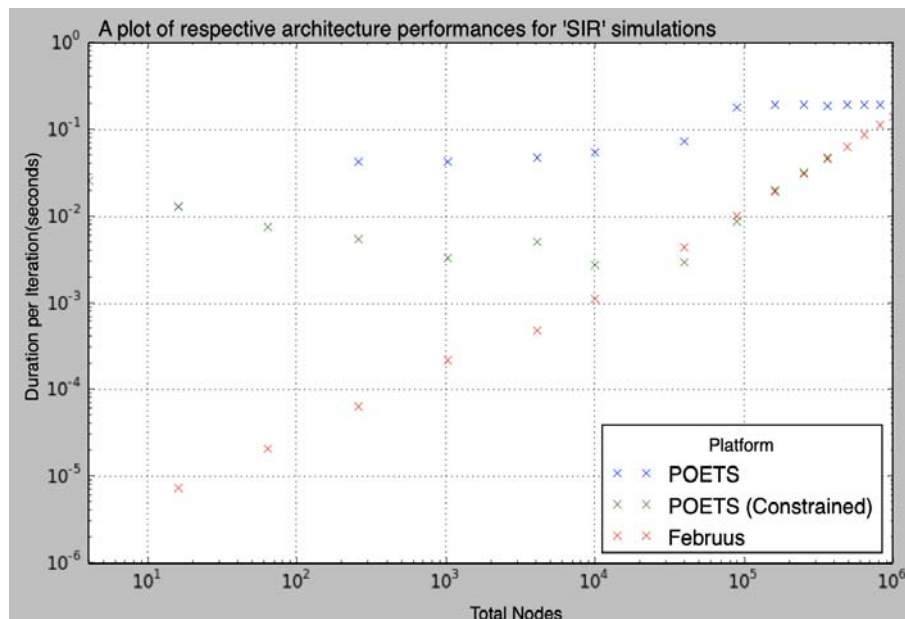


Figure 5.10.1.b: POETS versus a traditional approach (Februus), as described in-text.



5.10.2 Graph convolutional networks: machine learning approach for irregular problems

Modern artificial intelligence is dominated by machine learning approaches: methods that "learn" from data in order to predict behaviours from observed patterns. Like Tsetlin machines, neural networks are a form of machine learning artificial intelligence. Graph Convolutional Networks (GCNs) are neural networks that operate on data that is best represented as a graph.

GCNs operate using a message-passing mechanism: neural network layers that map a data graph onto an "updated" data graph. These layers are so named because each node "sends a message" containing its data to each of its neighbours, who then update their state using the data from their neighbours. Figure 5.10.2.a shows a two-layer GCN.

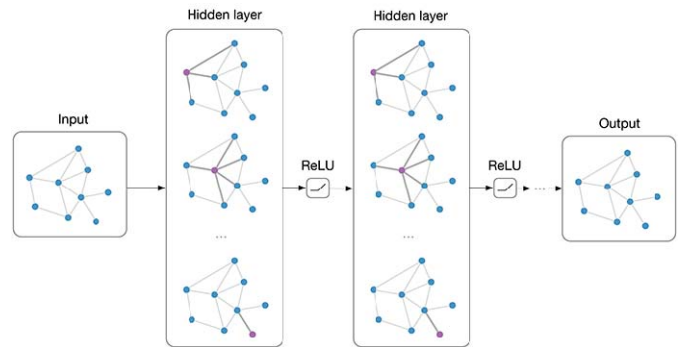


Figure 5.10.2.a: Schematic describing the operation of a two-layer Graph Convolutional Network. In each layer, each node broadcasts its state to its neighbours, and each node updates their state using this information. *Image credit: Thomas Kipf.*

Given:

- A matrix of feature values \mathbf{X} , with rows equal to the number of nodes in the graph, and columns equal to the number of features associated with each node, and
- Another matrix \mathbf{A} , which is the adjacency matrix added to the identity matrix, the n th layer in a GCN is defined by

$$\mathbf{X}^{(n+1)} = f(\mathbf{X}^{(n)}, \mathbf{A}),$$

where f is a nonlinear function that transforms the features of the graph from one iteration to the next. One commonly-used example of such a function is

$$f(\mathbf{X}^{(n)}, \mathbf{A}) = \sigma(\mathbf{A}\mathbf{X}^{(n)}\mathbf{W}^{(n)}),$$

where σ is a nonlinear source (often a linear rectifier) applied in an elementwise manner, and where $\mathbf{W}^{(n)}$ is a matrix of weights defined by the training process of the GCN. It is standard practice to also normalise \mathbf{A} to be degree-invariant like so:

$$f(\mathbf{X}^{(n)}, \mathbf{A}) = \sigma(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{X}^{(n)}\mathbf{W}^{(n)})$$

where \mathbf{D} is a diagonal matrix with values equal to the connectivity of each column.

The POETS approach requires a node-wise formulation. From the node-level perspective, the feature vector $\mathbf{x}^{(n+1)}$ for node i is given by summing the products, for each neighbouring node j , of that node's features with the corresponding entry in \mathbf{W}_n , and with the entry in the diagonally-normalised matrix $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, all subject to elementwise σ . Formally:

$$\mathbf{x}_i^{(n+1)} = \sigma\left(\sum_j (\text{deg}(i) \text{deg}(j))^{-1/2} \mathbf{W}^{(n)T} \mathbf{x}_j^{(n)}\right)$$

where j represents the indices of neighbouring nodes, and where "deg" is the degree of a node, including the self-loop (as per the definition of \mathbf{A}). Many of the elements of this



equation are constant with respect to the problem, allowing most of the layer behaviour to be precomputed, resulting in:

$$\mathbf{x}_i^{(n+1)} = \sigma\left(\sum_j \mathbf{b}^{(n)T} \mathbf{x}_j^{(n)}\right) \text{deg}(i)^{-1/2}, \text{ where}$$

$$\mathbf{B}^{(n)} = \text{deg}(j)^{-1/2} \mathbf{W}^{(n)T},$$

and where \mathbf{b} is a column of \mathbf{B} corresponding to the node i .

As with the SIR example, this node-wise formulation maps naturally onto POETS: each node in the graph is represented as a device, and each edge in the problem maps to a connection between two devices. The state of each device is updated as the GCN passes through each layer. Note that there is no training here - we load a pre-trained network onto POETS to perform inference.

Figure 5.10.2.b compares how this POETS-based approach performs inference on a network, with an existing machine learning tool (PyTorch) running on traditional compute hardware. POETS handles larger networks well, despite the problem acting on a relatively small timescale.

While this study serves as a proof-of-concept, it is worth noting that, in both POETS and Torch, the time to load in these graphs dominates the inference time. A follow-up study will explore how an event-triggered compute approach can be used in the training process, and to feed in data while the application is running on POETS. This upgraded approach will enable a combined learning and inference application - essential for the creation of national-scale digital twins.

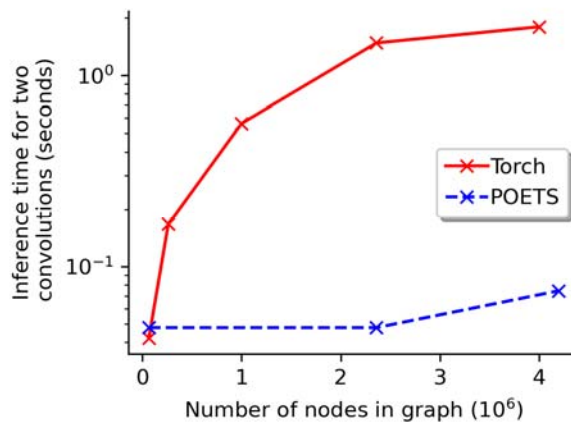


Figure 5.10.2.b: Inference time for a GCN on POETS, compared with Torch. Each point is an average of three timed simulations.



6. Domain specific applications - II

At the outset of the POETS programme grant, we identified a number of application domains (which included drug exploration/discovery and dissipative particle dynamics). The intention was to focus initially on this set - and specifically the two areas above - whilst exploring other application domains that appeared interesting on the way. The previous section outlined the portfolio of application domains we *have* examined - by no means an exhaustive list; there will be many more awaiting discovery. Here we report on the drug discovery collaboration with e-Therapeutics and the dissipative particle dynamics collaboration with EPFL in Switzerland.

6.1 Drug discovery

This research theme was undertaken in collaboration with a drug discovery organisation, e-Therapeutics. The computational drug discovery method developed by e-Therapeutics is intended to dramatically reduce the cost of developing new drugs by suggesting and filtering out drug candidates prior to any lab experimentation.

The core idea of the method is as follows²⁷: Given a network of all protein interactions related to a specific disease, the goal is to "disrupt" the network using the fewest node removals. The metric for measuring network disruption is the Average Shortest Path (ASP) in the network. Calculating this metric is one of the core computational problems of this application area, as the calculation needs to be performed many times during the search for the optimal solution. Using POETS hardware to accelerate this computation can deliver orders of magnitude increase in speed. Figure 6.1.a. depicts the performance we presented at the previous Advisory Board - compare with the results of figure 5.2.2.c.

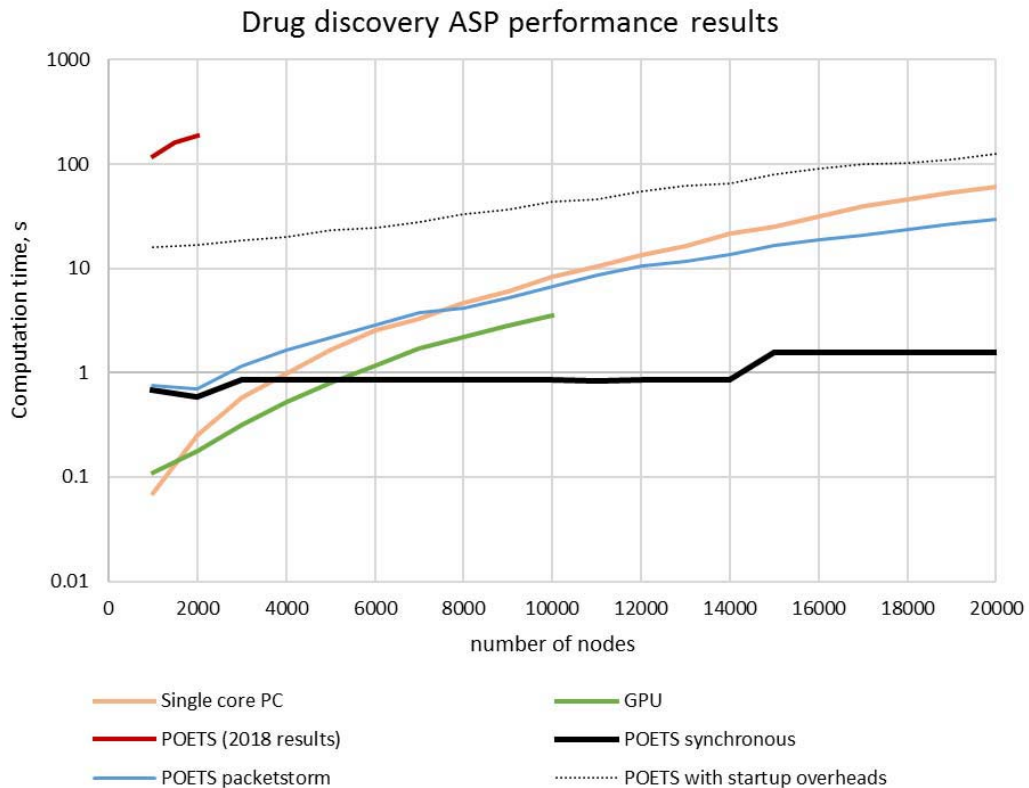


Figure 6.1.a: Searching a protein interaction graph.

²⁷ A. Mokhov, A. de Gennaro, G. Tarawneh, J. Wray, G. Lukyanov, S. Mileiko, J. Scott, A. Yakovlev, A.D. Brown. "Language and hardware acceleration backend for graph processing" Proceedings Forum Design Languages (FDL) pp 71-88, 2017, IEEE.



Promising though this avenue of research initially appeared to be, e-Therapeutics took a corporate strategy decision mid-way through the COVID pandemic to change their research focus, into areas that no longer coincided with what POETS had to offer, and so we (reluctantly) decided it was in the best interests of both parties to discontinue the collaboration. Mutual interest in each others activities persists, however; witness the presence at the Board Meeting of the e-Therapeutics CTO, Jonny Wray.



6.2 Dissipative Particle Dynamics

We have continued our research into the use of POETS for Dissipative Particle Dynamics (DPD) simulation, in collaboration with Dr. Julian Shillcock from EPFL. A DPD simulation models chemical processes through the interaction of "beads" representing multiple atoms or molecules, which avoids the need to model long-range forces. DPD is particularly attractive for POETS due to the short-range interaction forces and unpredictable local distribution of beads, which allows us to use fine-grain communication-centric simulation algorithms.

At the previous advisory board we presented initial results from simulation of DPD in the POETS platform, and were able to correctly simulate the DPD forces using a velocity-Verlet algorithm. This used a four-phase algorithm with initial support for simplified Hookean bonds, allowing us to simulate simple "floppy" polymer chains, but without the ability to tackle real simulation problems. Our goals at the previous advisory board were to:

- Increase functionality to the level needed for scientifically useful simulations.
- Allow complex problem descriptions to be imported into POETS-DPD.
- Improve and optimise performance, through algorithmic and implementation optimisations.

We have achieved those goals and more:

- **Angle bonds:** New algorithms to support angle bonds have been developed, implemented, and tested in Tinsel. Angle bonds are critical for real-world problems with "stiff" polymers.
- **Integration:** The POETS-DPD tool has been adapted to import problem descriptions from the existing Osprey DPD tool, allowing complex problems to be imported and executed in POETS hardware.
- **Optimisation:** New algorithms have been developed which reduce the number of messages and the cost per message, improving the baseline performance in the Tinsel hardware.
- **Generalisation:** insights gained from designing the POETS algorithms have been ported back into new multi-threaded DPD simulators, resulting in new pure software simulators that are faster than existing tools such as LAMMPS.
- **Exploration:** working directly with an end-user highlighted areas where computer scientists could see solutions that are not obvious to an end-user, allowing us to develop a powerful parameter search space tool that is in active use.

6.2.1 Baseline simulation algorithm

To understand these changes we must briefly summarise the original event-driven algorithm. Algorithm 1 shows the state machine for a single device managing a unit cell volume of space. The "resident" set contains all beads current contained within that volume of space.

The algorithm starts in the migration phase, by applying velocity-verlet Newtonian forces to each bead based on their current position, speed, and force. If the bead's new position is still inside the cell it remains resident, while if it has left the cell it is added to the migration list ("beads_to_migrate").



Once the beads have all moved, the cell starts broadcasting any beads that are migrating to its neighbours. At the same time it is also receiving any migrating beads from neighbours. If an incoming migrating bead is located within the cell, it is added to the resident set. Each migrating bead will be located in exactly one destination cell, so the other 26 cells will reject it – this is wasteful, but the number of migrating beads per time-step is relatively small.

The migration phase ends once all migrating beads have been both sent and received.

The second phase is the sharing phase, where the position and velocity of each resident bead is shared with neighbouring devices. As beads are received from neighbours, they are interacted with every local resident, adding to the accumulated force on each resident bead.

```
resident = list()
while True:
    #####
    ## Migration phase
    beads_to_migrate = list()
    resident_n = set()
    for b in resident:
        b = velocity_verlet(b, dt)
        if b.x in this.cell: resident_n += b
        else: beads_to_migrate.add(b)
    resident=resident_n

    when len(beads_to_migrate) > 0 and can_send:
        send("migrate", beads_to_migrate.pop())

    when b=recv("migrate"):
        if b.x in this.cell: resident += b

    wait_for_idle()

    #####
    ## Sharing phase
    beads_to_share = resident.clone()

    when len(to_share) > 0 and can_send:
        send("share", beads_to_share.pop())

    when o=recv("share"):
        for r in resident: r.f += calc_forces(r, o)

    wait_for_idle()
```

Once all beads have been shared and received, we have calculated every pair-wise force, and can return back to the migration phase.

This simulation algorithm is well matched to the fine-grain concurrency of POETS, and is guaranteed to implement the exact same algorithm as a sequential algorithm (ignoring floating-point associativity artefacts). This is possible even though random numbers are used in `calc_forces`, as we created a Tinsel-specific hash function that combines two bead ids and the current time to create a unique random number.

6.2.2. Angle bond support

Many chemical systems of interest contain a mix of single-bead monomers (such as water), and multi-bead polymers (such as proteins and oils). Previously we were able to support linear polymer chains, where each bead is bonded to at most two other beads, and the bonds are simple Hookean springs. Supporting complex simulations required us to support beads with stiffness, which requires 3-body angle bonds, so that if a polymer is bent it tries to restore its angle. We also need to support polymers with branches, in order to create head and side-groups on polymers. In a software system these are both relatively simple to support, as we can calculate all pair-wise DPD and Hookean forces, then loop over each polymer's structure and calculate the internal angle bond forces.

The fine-grain spatial decomposition used in the POETS-DPD approach makes this much more complex, as each cell receives the positions from beads in different messages. This presents two problems:



- The cell containing the centre bead in the angle bond will receive both ends of the bond, but they will arrive in different messages in an unknown order.
- The cells containing beads at either end of an angle bond will receive the position of the centre bead, but are unlikely to receive the position of the other end of the bond.

This means that the cell containing the centre bead receives enough information to calculate the angle forces, though in an unpredictable order, while the cells containing the angle ends will usually not see enough information to calculate the angle forces for the ends. We considered a number of solutions to this problem, including:

- Broadcasting the positions of angle end-beads to a neighbourhood of distance two. However, this requires broadcasting such beads to $5^3=125$ cells, of which 124 will simply waste time ignoring the information.
- Giving each angle bond an “anchor device” in the graph, which will always receive a unicast message containing the position of the beads in angle bond. Once the anchor device receives all three positions, it sends the forces back to the devices. This is quite attractive, but is not currently expressible within the constraints of the POETS application language.

The approach we used is to have the cell containing the angle bond wait until it sees both angle ends as part of the normal position broadcasting. Once both ends are received it calculates the angle forces on all three beads, and then sends the force to the two end-points as extra messages. This approach still results in message broadcasts, but they are within the $3^3=27$ neighbourhood used for sharing positions and bead migration. As an optimisation we also observe that these force updates are quite small, requiring a 4-tuple of (bead_id,fx,fy,fz), so we are able to batch up multiple forces into a single message.

Algorithm 2 shows the changes to the original algorithm, showing just the migrating phase, which now becomes a migrating and forcing stage. The original parts are shown in italics, while angle-force specific parts are shown in bold. A new set we need to maintain in this phase is the “forces_to_share” set, which contains angle end-point forces for beads centred in this cell which need to be sent to neighbours.

The main change occurs when receiving a shared bead *o*, after calculating the interaction with a resident bead *r*. We now check whether bead *o* is a partner in an angle bond centred on bead *r*. If so, we consult a local “partners” dictionary to check for a previously seen partner. If no partner is found then *o* is the first partner, so we store it in the partners dictionary. Eventually the other partner will be received, at which point we have all the information needed to calculate the angle forces: the previously seen end-point *p*, the resident centre bead *r*, and the just-arrived end-point *o*. This calculation results in three forces for the three beads in the bond; the force applied to the centre can be directly applied, while the other two beads may be located in other cells, so they are added to a list “forces_to_share” for forces to broadcast to the neighbourhood.

```
#####
## Sharing and forcing phase phase
wait_for_idle()
beads_to_share = resident
forces_to_share=set()

when len(to_share) > 0 and can_send:
    send("share", beads_to_share.pop())

when o=recv("share"):
    for r in resident:
        r.f += calc_forces(r, o)
        if is_angle_partner(r, o):
            p=partners[r.id]
            if not p:
                partners[r.id]=o
            else:
                (fhead,fmid,ftail)=angle_force(p,r,o)
                forces_to_share += (p.id, fhead)
                r.f += fmid
                forces_to_share += (o.id, ftail)
                partners[r.id]=None

when len(forces_to_share)>0 and can_send:
    send("angle_force", forces_to_share.pop())

when af=recv("angle_force"):
    for r in resident:
        if r.id == f.id: r.f += af.f
```



The new send and receive handlers are there to take elements from “forces_to_share” and distribute them to neighbours, and then to check incoming forces to see if they should be added to any resident beads. Once all forces have been shared (and all positions have been shared), we can return to the migration phase as normal.

As with the original algorithm, this technique is guaranteed to provide the same results as a sequential implementation (ignoring floating-point associativity). It requires memory proportional to the maximum resident set in a cell, just like the original POETS-DPD algorithm, and the asymptotic message count and time complexity remains the same. This technique has been implemented and tested in Tinsel, and verified in practise against sequential software versions.

6.2.3 Integration

A common weakness of research into accelerating applications is to focus on the computation kernel, and then only apply the kernel to model problems. We want to make POETS useful for real-world DPD problems, which requires ingesting real-world problems using the descriptions and data formats already used in those domains. Our goal was to directly load and simulate DPD problems generated within the Osprey DPD packages, as this has a large number of tools for constructing and analysing simulation problems, and is also the package created and maintained by our research collaborator.

The POETS-DPD software package needs to link with multiple common and uncommon libraries, such as the hostlink library for accessing Tinsel hardware, graph clustering libraries like Metis, and parallel processing libraries like Intel TBB. Attempting to embed this within the already complex Osprey DPD package would add more complexity, so the approach taken to integration is to define an interchange format for exporting DPD problems from Osprey into a file, then loading that file into POETS-DPD. We can then execute the simulation within POETS-DPD for some target number of timesteps, while generating Osprey DPD compatible output files at regular intervals.

Using an intermediate format in this way has provided a relatively simple route to running an existing Osprey problem using POETS-DPD as a backend. It has also been extremely useful for debugging, as it allows us to take the same DPD problem and run it side-by-side in POETS-DPD and Osprey. While they eventually diverge due to floating-point rounding, it allows us to check that the two simulations are performing the same calculations, providing strong evidence that POETS-DPD running in Tinsel accurately implements the DPD model.

6.2.4 Optimisation

The POETS DPD implementation presented at the previous AB was correct, but was not yet optimised. We have since implemented two types of optimisation:

- Implementation optimisations that make better use of cache, network, and the CPU.
- Algorithmic optimisations that reduce the overall amount of work needed.

One very effective algorithmic optimisation was to exploit Newton’s 3rd law – rather than calculating each pair-wise force twice, we can calculate it for just one bead in a pair, then forward the force to the other bead. This is a well-known technique for improving performance in molecular simulations as it halves the number of force calculations - which makes up most of the compute cost - and is often used in software simulators. We initially did not implement it in POETS because we wanted to explore extremely parallel implementations, and also because it is relatively complex to implement in an event-based algorithm.



One change needed to support Newton's 3rd law is to break the symmetry of the neighbourhood. In the two algorithms shown earlier, each message sent is broadcast to the entire 3³ neighbourhood, including the sender of the message. To support the asymmetry of Newton's 3rd law, we split the DPD force calculations into five parts:

- **Forwards sharing:** each cell sends its resident beads forwards to half of its neighbours.
- **Forwards receipt and force calculation:** as each bead is received, the pairwise force is calculated and applied for each resident bead, then the opposite force is added to a pending list of backwards forces.
- **Backwards distribution:** each cell sends forces backwards.
- **Backwards receipt and forces accumulation:** backwards forces are collected, and applied to the target beads.
- **Resident interactions:** we now handle intra-cell calculations as a separate step, as they don't fit with the forwards and backwards pattern.

A weakness of this approach in the current application language is inefficiency during the backwards pass, as the forces will be sent backwards to $(3^3-1)/2=13$ cells, but it can only ever be relevant to one cell. This is a similar problem to the angle force problem, where it is impossible to precisely target the message against the one cell (i.e. device) that needs the message. In practise the message distribution is very fast, and the overhead is low compared to the computational savings from halving the number of force calculations.

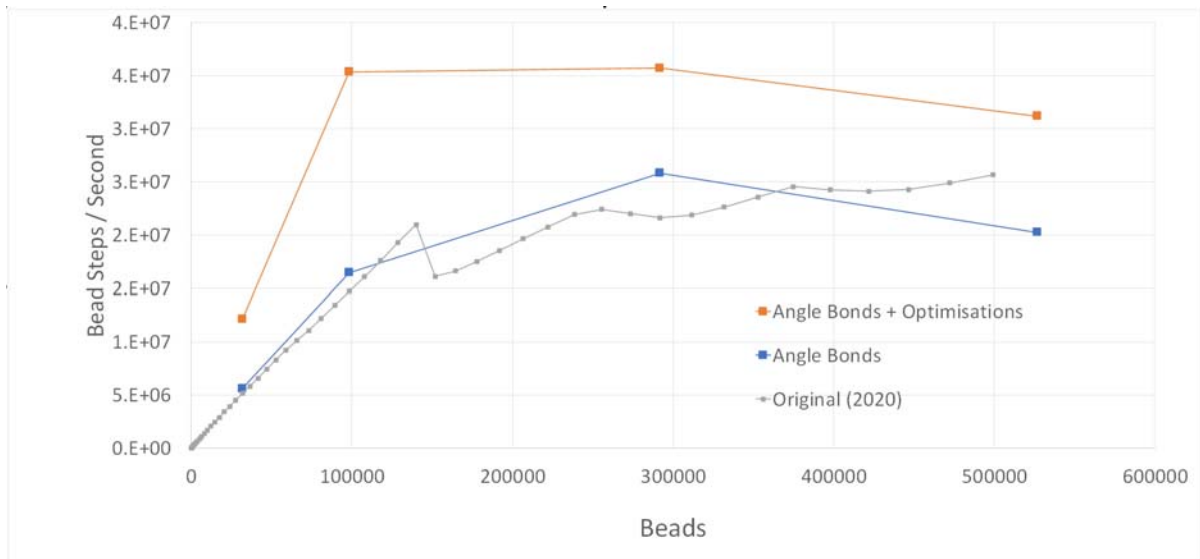
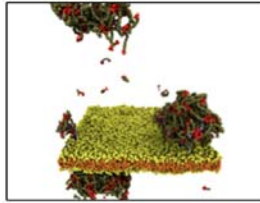


Figure 6.2.4.a: Comparison of original simple DPD performance vs angle bond and optimised performance.

Figure 6.2.4.a compares the performance of the original POETS DPD algorithm against the new versions, all running on 48 Tinsel cores. Performance is measured in Bead Steps per Second (BPS), which provides a metric that is independent of problem size and the number of time-steps. Despite the additional complexity of the angle bond support, the implementation optimisations mean that it remains the same speed as the original, much simpler and less useful, algorithm. The addition of optimisations for Newton's 3rd law provide even higher performance, while still supporting the additional features.



Protein driven domain formation accelerated by novel hardware



- 6x spatial resolution
- 5x simulated time
- 40x performance

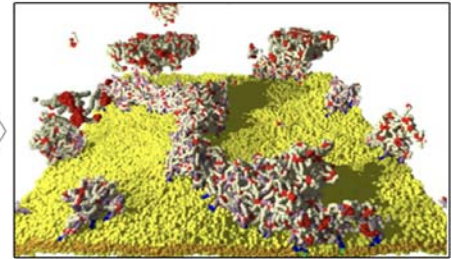


Figure 6.2.4.b: DPD on POETS.

These features allowed us to perform simulations of a much larger spatial and temporal time-scale compared to Osprey, and was used to explore interactions of protein driven domain formation on membranes²⁸.

6.2.5 Generalisation

One surprising result of the algorithmic work to support POETS DPD is that it has led to much faster multi-core software DPD simulators. The POETS approach explicitly moves beads around in memory, both during migration and during position sharing. This has the effect of maximising spatial locality within the system – beads that are accessed together tend to be located close to each other in memory, as they are resident in the same spatial cell managed by one device. This locality also works very well in traditional cache hierarchies on multi-core CPUs, which was noticeable when running POETS simulations in software. In particular, the high-performance parallel POEMS simulator was actually competitive with the standard LAMMPS DPD simulator. This was surprising at first, until the benefits of locality became more obvious.

This insight led to a strand of research which explored how to use the ideas of POETS to write better multi-core x86 implementations of DPD. We combined the event-triggered algorithm with a number of existing parallel computing optimisations including:

- Parallel design patterns based on work-stealing
- Vectorisation using AVX;
- Switching between Array of Structs and Struct of Arrays representations;
- Cache packing and locality optimisation;
- Cache-oblivious data-structure packing;
- NUMA aware code scheduling.

The result of all this was one of the fastest multi-core DPD simulators for x86, which exceeds the performance of all the multi-core and GPU implementations provided by LAMMPS. This wasn't the original intent of the research, but the flexibility of the programme grant allowed us to pursue this research avenue, and end up with new software capabilities alongside the hardware simulations.

²⁸ Julian C. Shillcock, David B. Thomas, Jonathan R. Beaumont, Graeme M. Bragg, Mark Vousden, and Andrew D. Brown. "Coupling Bulk Phase Separation of Disordered Proteins to Membrane Domain Formation in Molecular Simulations on a Bespoke Compute Fabric". *Membranes* **12** 1 2022. DOI: 10.3390/membranes12010017

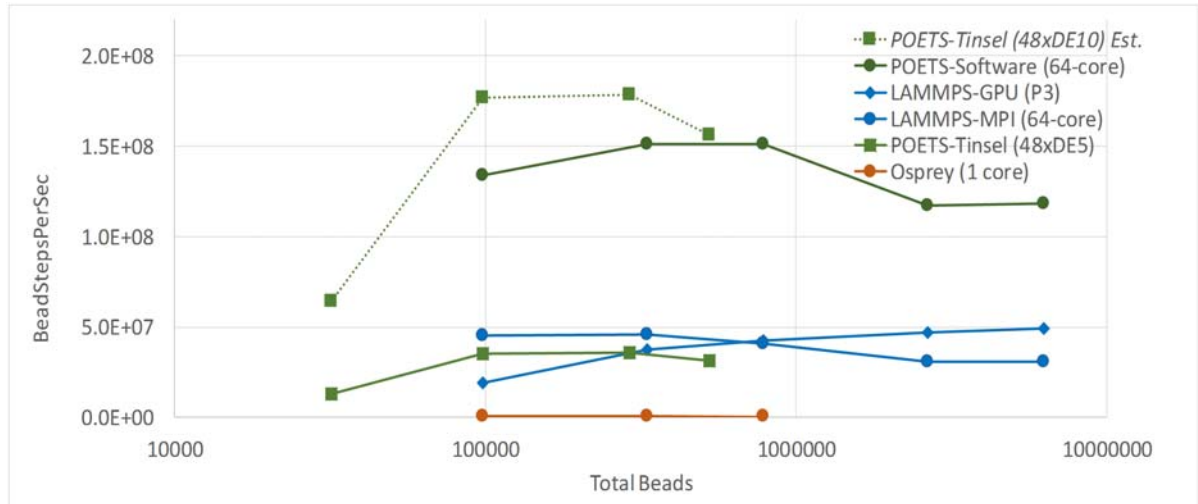


Figure 6.2.5.a: Performance of POETS DPD vs LAMMPS and Osprey simulators.

Figure 6.2.5.a compares the performance of five DPD implementations (solid lines):

- POETS-Software : the event-inspired software implementation running on a 64-core x86 machine.
- LAMMPS-GPU : an off-the-shelf optimised simulator executing on a higher performance GPU.
- LAMMPS-MPI : an off-the-shelf optimised simulator executing on the same 64-core machine used to run POETS-Software.
- POETS-Tinsel : the event-driven algorithm executing in the first generation DE5 hardware.
- Osprey : the reference DPD implementation running in a single core.

We can see that by far the highest measured performance is the new POETS software implementation, which is more than twice the speed of existing solutions.

While we have not yet been able to run it (see comments on hardware supply chain issues elsewhere), we also include extrapolated performance for running POETS-Tinsel in the newer DE10 boards. This is likely a lower bound, as we have not been able to optimise the implementation to take into account the new hardware features.

6.2.6 Exploration

Another unexpected benefit of the POETS research followed on from the discovery that POETS algorithms are very effective in software as well. When using POETS to perform real DPD simulations with our collaborator we initially had a simple process:

1. a few DPD problems would be sent for execution on POETS;
2. the simulations would be executed and the results gathered;
3. results would be returned for analysis;
4. the next set of DPD problems would be prepared based on the previous results, returning to step 1.

This iterative method was slow, and eventually it became clear that we were thinking about the problem in the wrong way – effectively we were sequentially sending DPD problems through the single DPD hardware system, then tuning the DPD problem parameters using a human in the loop. Given the very fast software DPD solvers provided by the POETS-style algorithms, we could both run simulations very fast, and exploit computer-level parallelism provided by institution HPC facilities – in our case, we could use the large Iridis system at Southampton to run up to 20 DPD problems in parallel.



This approach has allowed us to take humans out of the loop, and provides a form of parallelised grid search. Given a DPD parameter space to explore, we can form the cross product of two parameters of interest, and run all of them at once. The human can then examine the 2D grid of results, and use them to better understand how behaviour varies with the parameters, and also to decide where to search next.

Using a combination of the POETS software implementations, the Iridis HPC cluster, and some front-end scripting, we have created a system that allows the following work-flow:

1. DPD user prepares a zip file containing a grid of Osprey DPD problems
2. A single script is executed on the zip file which
 - a. Runs Osprey on each element in the grid to generate the initial DPD world state.
 - b. Creates a job script for running POETS DPD on the world state, including generation of snapshots and images as it runs.
 - c. Submits the job for execution in Iridis
3. All jobs run to completion with no extra user interaction.
4. A finalisation script is executed which assembles the individual outputs into a grid of images, and prepares a large zip of all results to return to the user.

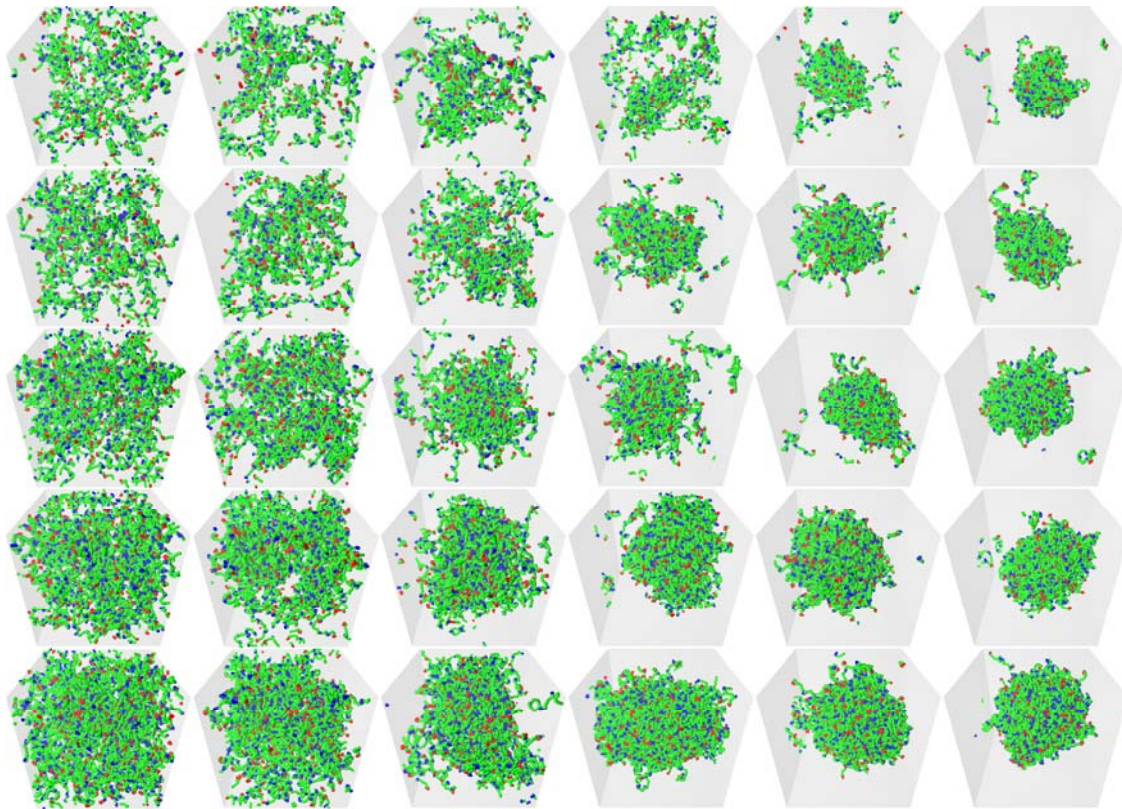


Figure 6.2.6.a: Example of a parameter exploration grid.

Using this approach we regularly execute problem grids containing 8×10^{12} bead-steps, requiring more than 10^{15} total floating-point operations. Figure 6.2.6.a gives an example of this kind of grid, where concentrations of a crowder polymer (not shown) increase from left to right, and the concentration of intrinsically disordered proteins increases from top to bottom. Looking at this grid provides insight into how the parameters affect behaviour, particularly when looking at the grids over time. Using Osprey this would have taken about 6 months, but using the results from POETS we were able to complete them in 4 hours.



7. Future technical plans

A research trajectory is a dynamic thing: rarely do investigators start out with a concrete goal and move linearly towards it. Event-based compute is no exception.

SpiNNaker is based on the assumption that biological nervous systems are susceptible to modelling as discrete systems, and is architecturally focussed tightly on delivering simulations based on this axiom in (human) biological real time. It is realised as an ASIC that absorbed - with hindsight - a disproportionate amount of effort, but nevertheless delivered a remarkable machine. In an academic environment, it is a remarkable feat.

POETS is a generalisation of these ideas: it is not a general-purpose machine, but for application domains that fit the event-driven paradigm, it can deliver outstanding (orders of magnitude) performance advantages. To attempt to avoid the lead time of ASIC development, the decision was taken at the outset to use COTS hardware, specifically FPGA platforms. These entailed significant compute and power budget performance losses, but had the obvious advantage that development could be an iterative process: a significant benefit over an R&D environment where design decisions cannot be reversed. POETS is ASIC-based, and so will never compete with an ASIC in terms of power or numerical performance. Further, for reasons outlined in the earlier sections of this report, we were constrained to use platforms almost a decade old for most of the project. This notwithstanding, POETS delivered on the initial research hypothesis: ***(almost) constant scaling, for a diverse range of application domains***. The absolute speeds may not be prima facie impressive, but context is everything: the machine is realised in terms of soft cores running at 200MHz. The implication is clear: ***we can make the system-wide FLOPS grow almost without limit simply by adding more hardware***.

There will be a roadblock, possibly several - there always is. However, POETS sidesteps many of the conventional issues by virtue of its architectural attributes:

- **Building it:** Educational projects notwithstanding, building ASICs is extremely expensive, and commercially daunting unless an instantaneous market of millions of parts can be all-but guaranteed. (An EUV lithography machine depreciates at about \$1000/hour; a significant cost factor in even installing one is the size of the foundations to take the weight.)
POETS is based on COTS components, where the NRE is spread over entire industries.
- **Designing it:** Submicron devices are not just voltage controlled current sources any more; design is difficult, slow and very expensive.
The individual compute components do not have to be cutting edge; gentler fabrication technologies can be employed.
- **Using it:** Ever higher clock frequencies imply relativity becomes an issue in getting signal edges across chip surfaces.
Computing where the data is, rather than moving the data to the compute, is the obvious way around this.
- **Data choreography:** The tradeoff between compute and communications is biased towards communication, and the situation is getting worse. However, on top of this comes an added compute cost component: choreographing the movement of the data requires communication-devoted compute on top of everything else.



Event based compute sidesteps this elegantly; problems are recast in the form - ultimately - of a relaxation algorithm, and self-time: there is no choreography cost, because we let the dataflow regulate itself.

- **Powering it:** Dark silicon; thermal gradients unbalance matched components, shift thresholds and decrease MTBF. The Silicon Roadmap tells us IC power distribution subsystems must deliver stable supply voltages of less than a volt at hundreds of amps, and keep the chip thermally stable whilst dissipating this power. *Whilst we can ameliorate this by careful rack/box/board design, POETS is not intended to be a laptop technology, and so large systems will usually reside in a server room. Nevertheless, even these have power budget limitations, and ultimately this, we suspect, will prove to be a limitation that cannot be avoided.*

We believe that event-based compute has a significant future, and our intention (after the end of the current project) is to move the technology forwards on three fronts:

- **SONNETS:** A natural, direct descendant of POETS. This will use event based technology to support machine learning and AI, in turn delivering a national-scale pre-emptive financial modelling capability. The proposal is currently with EPSRC.

SONNETS:
What does success look like? Financial tranquillity, and financial tranquillity is very good for business.

- **ENNUI:** The hardware purchased through the POETS grant will continue to be used to explore new application domains - section 5 in this report gives a flavour of the barely identified, let alone explored applications that we suspect will be amenable to event-based processing. We have the hardware; one of the deliverables of POETS is that a significant number of the prime movers are now in permanent academic positions, and as such, ideally placed to further pursue this area.

ENNUI:
What does success look like? More domains of applicability: Pattern recognition in reciprocal space (interpreting diffraction data directly), particle and field simulations (galaxies, geological processes, earthquake modelling), biological neural simulation, high voltage partial discharge modelling.... The list is large, and populated almost exclusively with arcane problems that will have no direct societal resonance, but nevertheless need to be addressed for the greater good of society.

- **ARIA:** The agency has opened its doors for business, and we have put in motion a massively ambitious project to model the passage of an entire virus through a cell wall. The timeliness of this is obvious, as are the potential returns on the investment.

ARIA:
What does success look like? When the next virus strikes, this technology will enable the elucidation of the attack and transport mechanisms in weeks, not months, and at a fraction of the compute resource cost that was brought to bear on COVID.



7.1 SONNETS

The POETS project has had many successes, but also posed questions about how to take the research further, and what other challenges should be tackled. To explore this we developed SONNETS, which is a 5 year EPSRC programme grant based on event-triggered systems. This project has passed through the pre-outline and outline application stages, and was invited for a full proposal. This was submitted on 30th September 2022, and is currently awaiting reviewer comments.

SONNETS is an ambitious research programme that provides new capabilities spanning computation, machine learning, and risk modelling for government and industry. Supporting all of these capabilities is the idea of Event-Triggered Systems (ETS), taken directly from POETS. In POETS we developed many of the underlying techniques and abstractions for ETS for running compute-intensive applications on the bespoke Tinsel hardware, but we believe that ETS can be used to solve problems arising when using large-scale heterogeneous cloud compute systems to analyse and model huge problems.

7.1.1 Motivating example: national-level financial risk analysis

A good example of the type of large-scale analysis problem that we have identified is modelling and analysing risk in the UK’s financial system. The Bank of England regulates around 1500 institutions, including high-street banks, investment banks, building societies, insurers, and others. These institutions trade a huge number of instruments among themselves, and with other companies in the UK and abroad. Each institution maintains an ever-changing portfolio of long and short positions, including on each other, and that complex web of positions is both a strength and a risk: what if too many institutions are exposed to a particular asset class? Worse: what if multiple institutions are exposed to a particular event, but this is obscured by transitive positions?

Currently regulators attempt to assess national risk through annual stress-tests: a scenario is defined, and each institution individually tries to assess the impact on their business. This is a slow process, with humans in the loop, and may not model risk well: the chosen scenarios have to be unrealistically severe but will also not capture other scenarios that are more likely and present - cumulatively - more danger. Individual institutions perform their own risk analysis, with larger institutions using more detailed counter-party risk analysis based on their current portfolio, but these are institution focussed: the goal is to avoid going bankrupt while maximising profit.

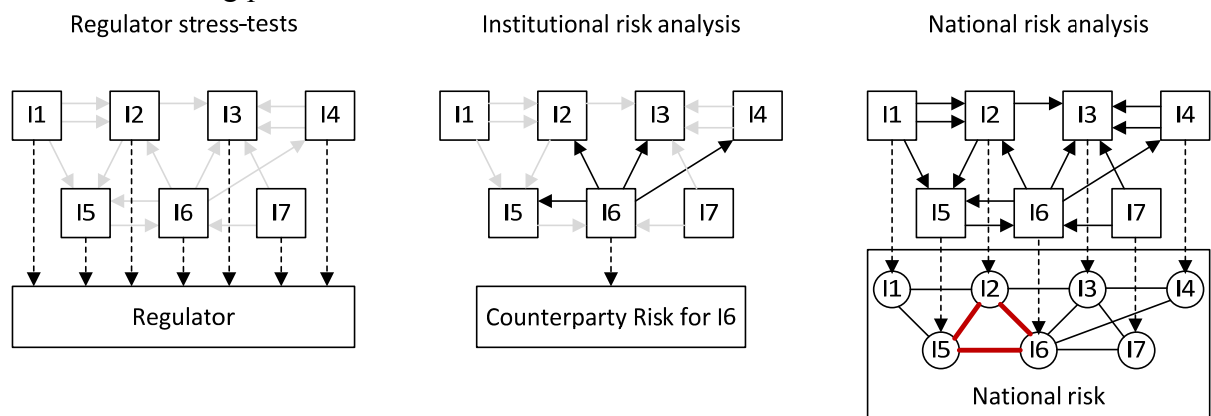


Figure 7.1.1.a: Risk analysis and stress tests

Our proposal is to consider the risk of the UK’s entire financial sector in real-time, where we try to model and analyse the UK as a whole on a day-by-day or even minute-by-minute basis. Certain types of risks can only be identified through the changing web of positions, and generating realistic scenarios requires a good understanding of the current state of the



UK. Trying to actually build a real-time centralised model of every institution's portfolio would require huge dataflows and present a massive computational problem – such a model would always be out of date, and provide a huge point of failure.

We believe that we can use some of the ideas from POETS, along with a lot of new research in other areas, to create distributed modelling capabilities: computation and analysis is pushed closer to the data-sources, with data summarisation and reduction performed locally, and then a global picture is built-up through exchanging local summaries. This would result in a form of "digital twin" for the UK's financial system, which strips away all the detail while retaining the most important signals and relationships.

7.1.2 Building a stack

The national risk analysis is one compelling example of a large-scale complex modelling and analysis task, but there are many others, such as electrical power distribution, traffic modelling, fraud detection, and disease modelling. Building any such a system presents challenges at multiple levels from application down to network and CPU, and we can't just pick one part and try to solve it individually. In SONNETS we plan to build a set of three inter-dependent tiers to solve such problems - with each tier, the research focus sharpens, along with the societal impact and relevance:

Tier 1: Event Triggered Computing (ETC) - a programming paradigm which supports the development of efficient large-scale applications distributed across heterogeneous resources in cloud computing systems - expensive bespoke compute platforms are avoided. This will leverage some of the results of POETS, but will also tackle new challenges not previously considered.

Tier 2: Event-triggered Artificial Intelligence (ETAI): a suite of new machine-learning technologies which can handle large irregular datasets (traditionally extremely challenging for AI), and can operate efficiently in the cloud using ETC. The application portfolio includes many high-value and compute intensive problems such as drug design, pandemic modelling, x-ray diffraction analysis and financial risk modelling. Financial risk modelling is chosen as our Tier 3 research activity.

Tier 3: Event-triggered Modelling (ETM): exploiting ETAI to understand nation-level problems, with UK-wide financial risk modelling as the main target. This capability will be extended to cover other complex modelling challenges, such as energy management and pandemic modelling.

7.1.3 Challenges we want to solve

There are some core challenges which currently block us from building this kind of capabilities, which cut through the different tiers and need to be solved using contributions from multiple tiers.

- **Real-time analysis:** how do we model and analyse large complex systems in real-time, rather than capturing and then analysing snap-shots offline?
- **Distributed computation:** producing and analysing these large models requires huge amount of compute of different types, which can only realistically be provided through a combination of public and private clouds.
- **Data movement:** computation must be moved towards data sources in order to reduce traffic and enable local summarisation and pre-analysis of data.
- **Robustness:** different parts of the system can and will go down or produce incorrect data, so the system must be robust at both a computational and analytical level.



- **Deployment:** to have any impact systems need to be designed for deployment, both in terms of technical aspects such as APIs, but also data privacy and manageability.

7.1.4 *Event-triggered modelling*

The motivating case study for SONNETS is to deliver the nation-scale risk-analysis outlined earlier, with real-time scenario generation and analysis using a fine-grain picture of the UK's institutions. This acts as both an example of the type of large-scale applications we're trying to support, and a distinct deliverable that is useful in and of itself. We have chosen this application because:

1. It is really hard, and we don't (currently) know how to do it.
2. If it can be done, even partially, there is the potential for huge societal benefit.
3. A large grant like SONNETS is the only way of doing it, as it requires regulatory and financial end-users to talk with and work with computer-scientists over multiple years.

We have a decent idea about how to get started: what kinds of analysis could be done, and what sorts of computational techniques could be used to achieve them. However, the only way to make something that works for end-users and could feasibly be deployed for real is to engage with stakeholders throughout. Just like POETS, a grant of this size and length provides a long period (5 years) over which to develop and revisit objectives, allowing substantial dialogue and back-and-forth between end-users and researchers – while we don't expect to have *deployed* the risk modelling capability nation-wide, we expect to have eliminated most of the technical and operational barriers to doing so.

Research and outcomes

The goal of SONNETS is to operate within heterogeneous public private clouds using COTS hardware, which is quite different to the homogenous specialised Tinsel hardware used in POETS. While there is clearly a great deal of research into cloud programming, our ETS perspective opens up others methods for developing and deploying applications at very large scales. Expected contributions include:

- New approaches to integrating and connecting heterogeneous compute resources, including CPUs, GPUs, TPUs, and FPGAs, into a single cohesive application
- Formal methods for proving robustness and correctness of applications in the presence of message re-ordering (already handled in POETS), and message loss.
- Systems for monitoring load balancing across large systems, and re-mapping and re-locating logical processes at runtime, both within/between cloud domains and to/from accelerators.
- Data management techniques to incorporate live data sources of different types within a large execution graph.

7.1.5 *Summary*

POETS has been a success in developing the underlying theory and tools for ETC, as seen in the rest of this report. That said, we always planned to reflect on the results and follow up with research projects that strengthen and deepen this research area. The SONNETS project both exploits the research and tools from SONNETS, while extending the reach into new areas and research challenges. Given the success of the POETS team we have tried to balance keeping researchers who work well together with integrating new researchers. It is notable that two of the post-doctoral researchers from POETS are now included in SONNETS as early-career researchers – this demonstrates the developmental success of POETS, as well as its research achievements.



7.2 ENNUI

The vertical approach taken in POETS led to the Tinsel hardware platform, which has been an extremely successful research platform. Part of its success comes from taking a bespoke approach to hardware development, but these also lead to some platform limitations:

- The use of custom PCBs and controllers makes it difficult to propose it as a general machine that anyone could construct or buy;
- The mesh approach to inter-FPGA connectivity is very efficient, but also presents deployment problems in a standard data-centre due to wiring complexity.
- Despite the highly efficient caching architecture and latency hiding mechanisms in Tinsel, off-chip bandwidth is still a limitation when working with very large graphs.

In order to support follow-on research into POETS technology, we are constructing a new experimental hardware test-bed. Where POETS focussed on custom hardware and network topologies for maximum efficiency, the ENNUI platform is intended to explore the use of pure COTS hardware and networks. Figure 7.2.a shows the main components of ENNUI:

- 24 Xilinx U280 FPGA cards, with 2x100Gb/s network connections and 8GB of on-chip HBM (High-Bandwidth Memory) providing 480GB/s of bandwidth.
- A central 100Gb/s low-latency 32-port router
- Four servers hosting six FPGA cards each, handling management and configuration.

We deliberately choose off-the-shelf components designed for cloud deployment, and choose FPGAs with HBM to explore the potential of on-chip bandwidth.

The ENNUI platform will support a number of follow-on research activities, including:

- The SONNETS proposal will (if funded) use the ENNUI cluster as part of its distributed test-bed.
- New CPU architectures and network topologies that support the Tinsel API while operating over standard ethernet protocols between FPGAs (PhD student candidate identified).
- Alternative implementation strategies for implementing POETS applications in hardware, using off-the-shelf HLS tools to directly synthesise handlers and streaming protocols to connect within and between chips (through recent PhD student Xinbo Zhang).
- Direct synthesis of the POETS-DPD algorithm in FPGAs, where we hope to achieve performance improvements of 10x over our best current implementations (expected to be pursued through a responsive mode grant).

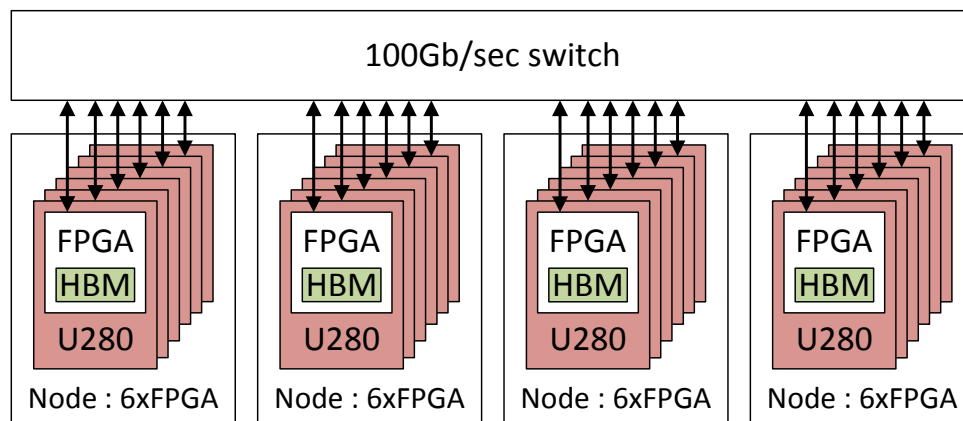


Figure 7.2.a: Architecture of ENNUI.



POETS

It is important to note that ENNUI complements the existing Tinsel platform, and does not replace it – in many cases the mesh approach of Tinsel is the most efficient, and the use of a fixed CPU+network overlay has been extremely robust and reliable. A strength of the POETS approach has been the ability to adapt the hardware implementation without modifying the applications, and ENNUI will only increase the hardware implementation options.



7.3 ARIA

ARIA (the Advanced Research + Invention Agency) has begun tentative operation, with an initial call for expressions of interest. We would like to build a DPD system, capable of modelling the complete endocytosis of a virus particle through a cell wall, at a capital cost that is accessible to all (we intend to implement it in the cloud), delivering wallclock speeds that make it possible (and sensible) to undertake significant parameter space explorations in timescales sympathetic to human reasoning speeds. We estimate a prototype system could be implemented for an outlay of approximately £17M. The initial approach to ARIA is shown below.

Expression of interest to ARIA

Technological advances allow us to realise - cheaply - vast numbers of processors (many millions). Programming these in a manner that allows full and sensible exploitation of the hardware is a significant system design bottleneck. One way of addressing this is by event-driven programming, where millions of small cores communicate asynchronously with each other, and the desired "solution" appears as an emergent property of the ensemble. This is an extremely powerful computational technique, and can produce - for certain classes of problem - virtually constant scaling with problem size.

Dissipative Particle Dynamics (DPD) is a computational technique supporting the physical simulation of billions of chemical particles, producing macroscopic chemistry in a computer: currently an extremely expensive capability, accessible only to those with access to (and experience of) traditional HPC.

COVID illuminated some limitations of these techniques. The structure of the virus was elucidated by complex biochemical sleuthing, assisted by computational modelling, at massive cost. The modelling illuminated parts of the big picture, but the utility of simulation comes in answering the questions that the user didn't think to ask.

The focus here is to build an event-based DPD system that will make it possible to ***watch an entire virus pass through a cell wall*** (in simulation), in a 'sensible' amount of wallclock compute time: hours, not months.

This takes simulation capability to space- and time-scales currently unobtainable, even with nation-scale budgets. This technology will put simulation capability in the hands of every competent lab researcher in every bio lab, and sufficiently cheaply that 'compute resource' disappears as a budget sheet line item. This is vitally important: the cost and accessibility will allow human analysts to ***play***, which allows human creativity to shine.

The next global pandemic may not be as amenable to analysis as COVID. We need computational agility and accessibility, we need vastly more powerful analysis tools, and we need them in place, functional and tested ***before*** the first signs of trouble appear.



8 Closing remarks

POETS has been a multifaceted project, requiring a broad spectrum of research, management and political input and abilities. The team has worked together magnificently, and (we feel, at least) it has been everything that an EPSRC programme grant should be: well resourced, sympathetic oversight from both the Advisory Board and EPSRC themselves, and executed by a team of talented individuals.

It has been successful on almost every level: career progression of staff (at all levels), discipline hopping to an extraordinary degree (both within Electronics and Computer Science and between higher levels of the technical hierarchy - chemistry, materials science, finance.....). It has been directly responsible (at the University of Southampton) for the creation of both a new research group (Computer Architecture) and a new undergraduate degree course (Computer Engineering, the one holding stewardship of the other) that will span the interests of both the Electronic Engineering and Computer Science components of the Department. Above all, it has given rise to a large volume of original and useful research, and has engendered three significant paths forward (the ENNUI, ARIA and SONNETS projects) with the potential for many more from section 5.

Programme grants are special, and extremely productive: the length of time, depth of funding and overall flexibility allowed us to engage four universities and undertake research that would be infeasible if we needed to bid for funding in smaller increments over shorter periods of time. Any of the individual major components of POETS - hardware foundation, software infrastructure, application domain exploration would, in principle, have been feasible under the aegis of Responsive mode funding, but none of the three would have had any real point without a guarantee that the others would be in place alongside it.

The only level on which we could arguably be faulted is on the outreach and communication component, but two years of COVID-induced disruption on that front would not have been a productive battle to fight.

On a personal note, I would like to note that I have never, in 42 years of active research both in industry and academia, worked with such a well-motivated and capable set of individuals - at all levels - before. It has been a pleasure to lead this project.

*Andrew Brown
11 October 2022*



Appendix A: Reactive systems

The analysis is presented here in some detail, because it illustrates the point made earlier that it is necessary to strip an analysis back to the fundamental mathematics before recasting to fit the POETS computation model.

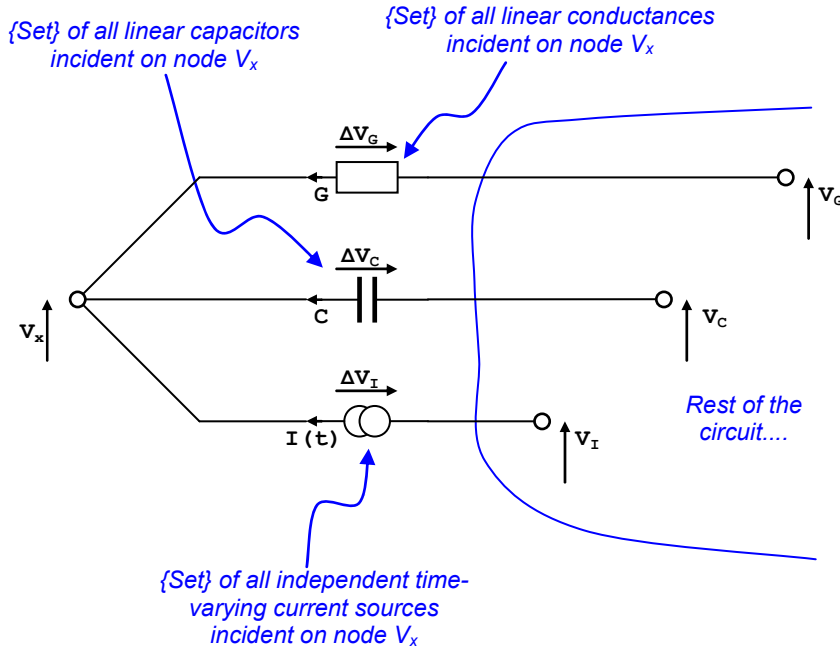


Figure A.a: The canonical reactive linear system.

A.1 Underlying mathematics - limitations

Only conductance (G), capacitance (C) and current sources, I(t) are considered. (Linearly dependent controlled devices are trivial extensions; without loss of generality, resistance (R), inductance (L), and pure voltage sources (V(t)) can be gyrated out of existence.)

A.2 The circuit

See figure A.a. The goal of the simulation is to generate the complete time history of all V_x in $[0,T]$ with no synchronisation - using a pure packet storm.

To do this, we duplicate the circuit for every timepoint. This approach leads to extremely large circuits very quickly, but the mantra of event-based computing has always been that cores are free. Here we push art the boundaries of that philosophy. Figure A.2.a generalises figure A.a into the discrete time domain.

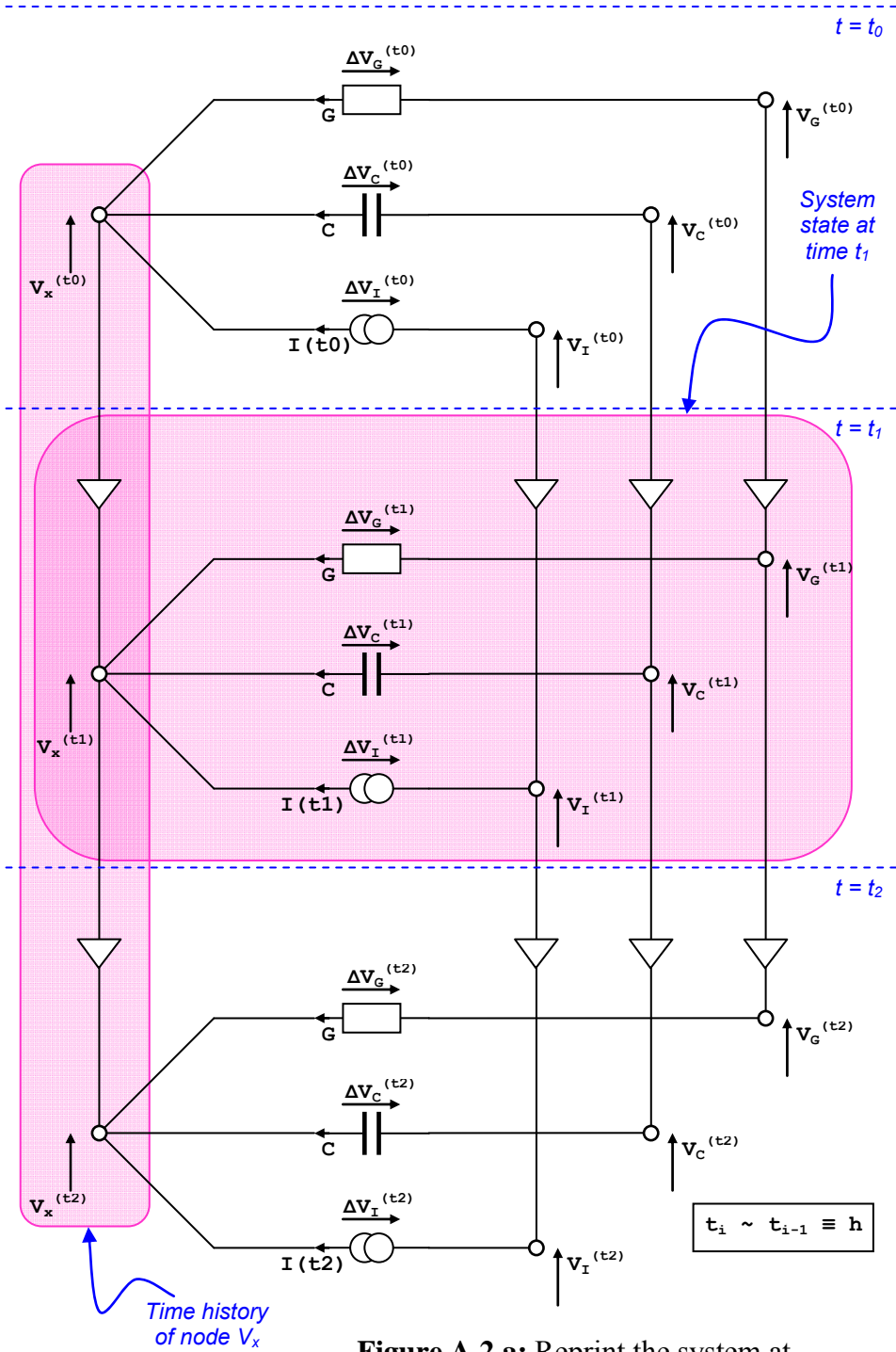


Figure A.2.a: Reprint the system at every timepoint (cores are free).



A.3 Data layout

Re-drawing figure A.2.a in terms of POETS devices (the coarse way) gives figure A.3.a:

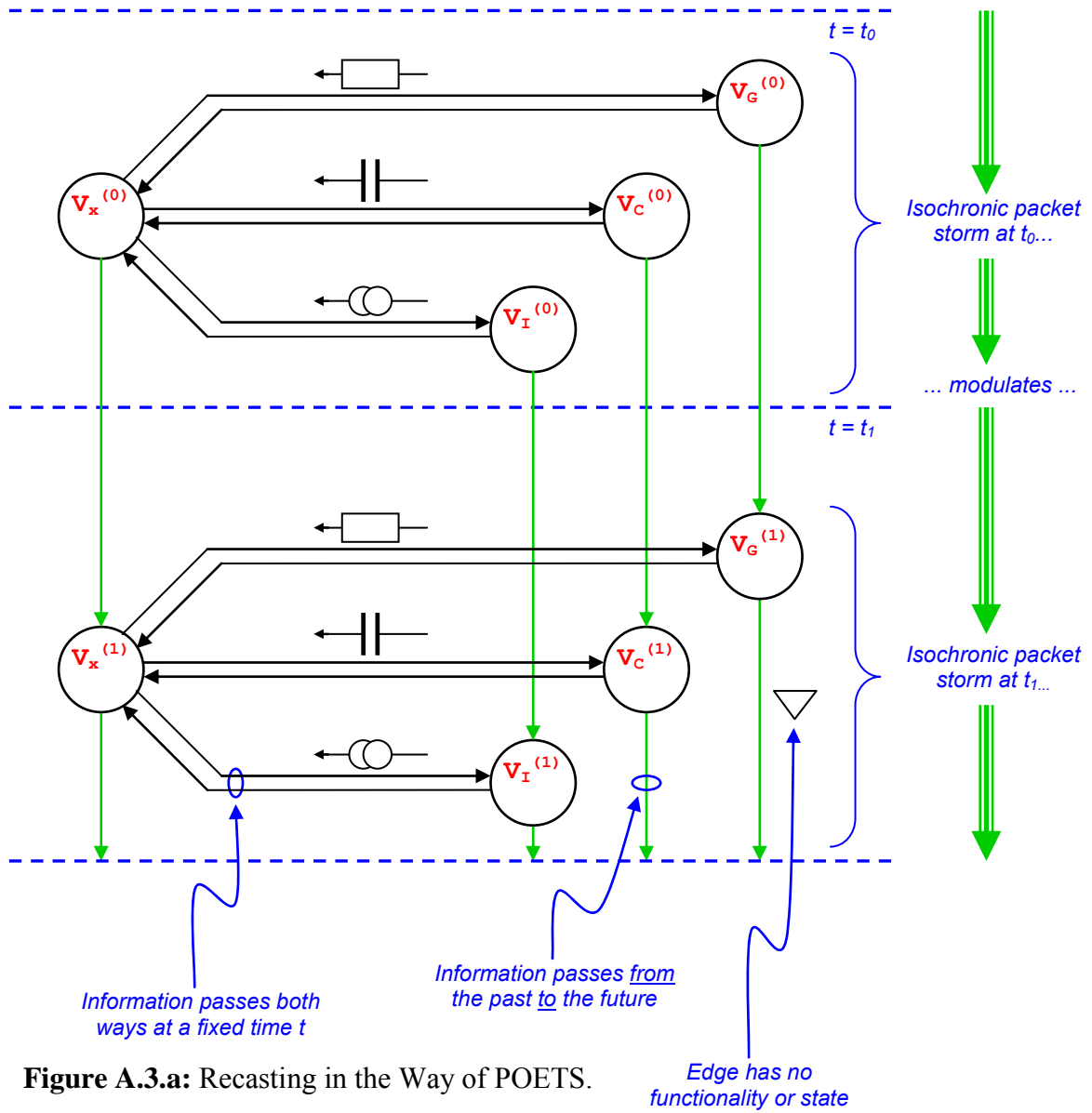


Figure A.3.a: Recasting in the Way of POETS.



Figure A.3.b refines the data layout in a more detailed POETS-sympathetic way.

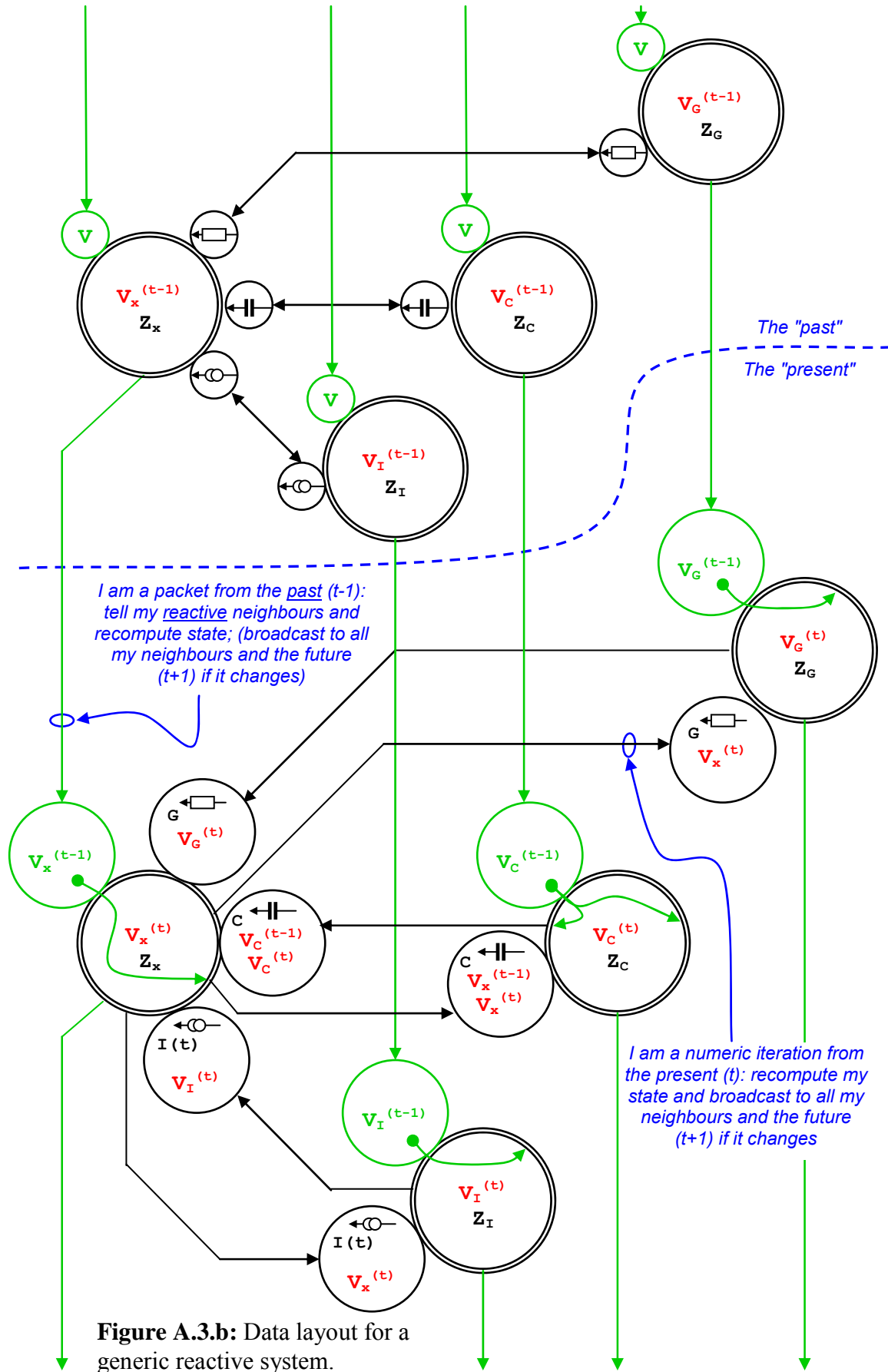


Figure A.3.b: Data layout for a generic reactive system.



A.4 The packet handlers

Having restructured our generic circuit in terms sympathetic to POETS, we need to formulate the device handlers:

Kirchhoffs Current Law on node x (figure A.a) - recall G, C, I(t), V_G, V_C, V_I are sets - gives us

GΔV_G + C ∂ΔV_C / ∂t + I(t) = 0 (1)

Expand the differential (δΔV/δt = (ΔV^(t)-ΔV^(t-1))/h) and re-arrange =>

(V_G^(t) - V_x^(t))G + C/h (V_C^(t) - V_x^(t)) - C/h (V_C^(t-1) - V_x^(t-1)) + I(t) = 0 (2)

Re-arrange again to make V_x^(t) the subject =>

V_x^(t) = (C/h (V_C^(t) - V_C^(t-1) + V_x^(t-1)) + I(t) + G V_G^(t)) / (G + C/h) (3)

Expanding the notation =>

V_x^(t) = (sum_i C_i/h (V_C^(t) - V_C^(t-1) + V_x^(t-1)) + sum_i I_i(t) + sum_i G_i V_G^(t)) / (sum_i G_i + sum_i C_i/h) (4)

From the perspective of evaluating equation (4), V_x^(t-1) is a constant - it is locked in the "past" - any change in any V^(t) can have no effect on anything at t-1. The past affects the future, but not the other way around.



The device **ReCompute** function: walk the pins, and get an absolute voltage contribution from each one (except the "last time point" pin):

```

void Device::ReCompute()
{
  Vxtold = Vxt;
  Vxt = 0;
  WALKPINS(i) Vxt += i.ReCompute(Vxtold);
  Vxt /= Z;
  if (Vxtold == Vxt) return; // To within some tolerance
  BCast(Vxt);
}

```

Pin type specific contribution functions:

```

current C::ReCompute(Vxt)
// Current contribution from reactive neighbours
{
  return (C/h) * (Vct - Vct1 + Vxt);
}

current G::ReCompute(Vxt) // Argument ignored
// Current contribution from conductive neighbours
{
  return G * Vgt;
}

current I::Recompute(Vxt) // Argument ignored
// Current contribution from current source neighbours
{
  return I(t);
}

```

Figure A.3.c ties it all together.

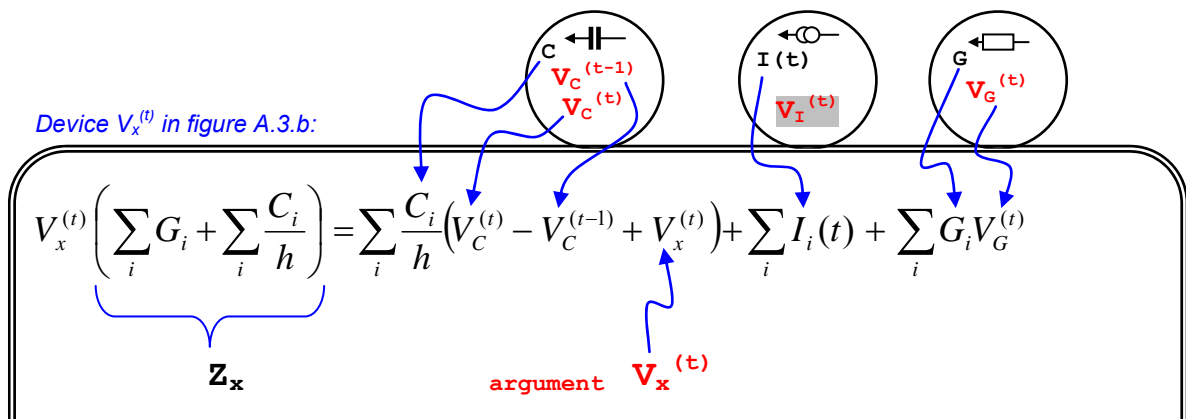


Figure A.3.c: Device **ReCompute** function(s).



A.5 Sanity check

$C/h \rightarrow 0$ in equation 4 gives

$G_i = G$ for all i and $I(t) = 0$ gives

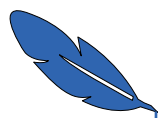
$$V_x^{(t)} = \frac{\sum_i I_i(t) + \sum_i G_i V_G^{(t)}}{\sum_i^{Ng} G_i}$$

$$V_x^{(t)} = \frac{\sum_i V_G^{(t)}}{Ng}$$

So far, everything is consistent.

Where does h come from?

- It's a linear circuit.
- No harmonics can ever be generated.
- Inspection of independent excitation gives highest frequency of interest.
- Whence h .



Appendix B: Publications

Papers cited as footnotes in the main text (except [18]) written by members of the POETS team or our collaborators; EPSRC support is acknowledged:

- [1] A.D. Brown, R. Mills, K.J. Dugan, J.S. Reeve and S.B. Furber "Reliable computation with unreliable computers", IEE Computers and Digital Techniques, 2015, **9**, no 4, pp 230-236, doi: 10.1049/iet-cdt.2014.0110
- [2] A.D. Brown "Event-driven computing" **keynote talk**, ASAP 2016, Imperial College London
- [3] A.D. Brown, D.B. Thomas, J.S. Reeve, G. Tarawneh, A. de Gennaro, A.A. Mokhov, M. Naylor and T.J. Kazmierski, "Distributed event-based computing", International conference on parallel computing, ParCo'17, Bologna, September 2017.
- [4] M. Naylor, S. W. Moore and D. Thomas, "Tinsel: A Manythread Overlay for FPGA Clusters," *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 2019, pp. 375-383, doi: 10.1109/FPL.2019.00066.
- [5] M.F. Naylor, S.W. Moore, D.B. Thomas, J.R. Beaumont, S. Fleming, M.L. Vousden, A.T. Markettos, T. Bytheway, A.D. Brown "General hardware multicasting for fine-grained message-passing architectures" Mar 2021, Proceedings - 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2021. p. 126-133 doi 10.1109/PDP52278.2021.00028
- [6] Matthew Naylor, Simon Moore, Andrey Mokhov, David Thomas, Jonny Beaumont, Shane Fleming, Theo Markettos, Tom Bytheway and Andrew Brown "Termination detection for fine-grained message-passing architectures" Proc ASAP 2020. Hannig, F., Navaridas, J., Koch, D. & Abdelhadi, A. (eds.). IEEE, p. 17-24 8 p. 9153260
- [7] Philippos Papaphilippou, Paul H. J. Kelly, Wayne Luk "Simodense: a RISC-V softcore optimised for exploring custom SIMD instructions" Proc. FPL, 2021, p. 391-397
- [8] Tim Todman, David Thomas, Wayne Luk "Exploring performance enhancement of event-driven processor networks" 2020 International Conference on Field-Programmable Technology (ICFPT), doi 10.1109/ICFPT51103.2020.00056
- [9] Tim Todman, Wayne Luk "Custom enhancements to networked processor templates" Proc. ISVLSI 2021, p. 224-229 2020
- [10] Andrew Brown, Tim Todman, Wayne Luk, David Thomas, Mark Vousden, Graeme Bragg, Jonny Beaumont, Simon Moore, Alex Yakovlev, Ashur Rafiev, "Non-deterministic event brokered computing", in HEART2022: International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies June 2022 pp 84-86 <https://doi.org/10.1145/3535044.3535055>
- [11] Tim Todman, Wayne Luk "Custom Instructions for Networked Processor Templates" IEEE Trans. Circuits Syst. II Express Briefs 2022 69(7): p. 3096-3100



- [12] Mark Vousden, Graeme McLachlan Bragg, and Andrew Brown. "Asynchronous Simulated Annealing on the Placement Problem: A Beneficial Race Condition". *Journal of Parallel and Distributed Computing*. **169** pp 242-251 2022
- [13] A.D. Brown, S.B. Furber, J.S. Reeve, J.D. Garside, K.J. Dugan, L.A. Plana and S. Temple, "SpiNNaker - Programming Model", *IEEE Transactions on Computers*, 64, no 6, June 2015, pp1769-1782, doi 10.1109/TC.2014.2329686
- [14] A.D. Brown "Event-driven computing" **keynote talk**, ACSD/PN 2015, Brussels
- [15] A.D. Brown, J.E. Chad, R. Kamarudin, K.J. Dugan and S.B. Furber "SpiNNaker: Neuromorphic simulation - quantitative behaviour", *IEEE T Multi-Scale Computing*, 4, no 3, July 2018, pp450-462, doi 10.1109/TMSCS.2017.2748122.
- [16] Mahyar Shahsavari, David Thomas, Andrew Brown, Wayne Luk "Neuromorphic Design Using Reward-based STDP Learning on Event-Based Reconfigurable Cluster Architecture" *ICONS 2021: International Conference on Neuromorphic Systems July 2021* pp 1–8 doi 10.1145/3477145.3477151
- [17] Alexander Rast, Mahyar Shahsavari, Graeme M. Bragg, Mark Vousden, David Thomas, and Andrew Brown. "A Hardware/Application Overlay Model for Large-Scale Neuromorphic Simulation". *2020 International Joint Conference on Neural Networks (IJCNN)*. 19-24 July 2020. DOI: 10.1109/IJCNN48605.2020.9206708
- [18] Richard Sinke "Targeted Next-Generation Sequencing can Replace Sanger Sequencing in Clinical Diagnostics" *Academia*, doi: 10.1002/humu.22332
- [19] Ashur Rafiev, Jordan Morris, Fei Xia, Alex Yakovlev, Matthew Naylor, Simon Moore, David Thomas, Graeme Bragg, Mark Vousden, Andrew Brown, "Practical distributed implementations of very large scale Petri net simulations": in *ToPNoC XVI, LNCS 13220*, pp112-139, 2022 https://doi.org/10.1007/978-3-662-65303-6_6.
- [20] M. Koutny, M. Pietkiewicz-Koutny, and A. Yakovlev, "Asynchrony and persistence in reaction systems", *Theoretical Computer Science*, Volume 881, 2021, pp 97-110, ISSN 0304-3975, <https://doi.org/10.1016/j.tcs.2020.11.040>.
- [21] M. A. N. Al-hayanni, F. Xia, A. Rafiev, A. Romanovsky, R. Shafik and A. Yakovlev, "Amdahl's law in the context of heterogeneous many-core systems - a survey," in *IET Computers & Digital Techniques*, vol. 14, no. 4, pp. 133-148, 7 2020, doi: 10.1049/iet-cdt.2018.5220.
- [22] V. Khomenko, M. Koutny, and A. Yakovlev (2022). "Avoiding Exponential Explosion in Petri Net Models of Control Flows". In: Bernardinello, L., Petrucci, L. (eds) *Application and Theory of Petri Nets and Concurrency. PETRI NETS 2022. Lecture Notes in Computer Science*, vol 13288. Springer, Cham. https://doi.org/10.1007/978-3-031-06653-5_14
- [23] Ashur Rafiev, Jordan Morris, Fei Xia, Rishad Shafik, Alex Yakovlev, Ole-Christoffer Granmo, and Andrew Brown, "Visualization of Machine Learning Dynamics in Tsetlin Machines" in *International Symposium on the Tsetlin Machine (ISTM)*, IEEE, 2022



- [24] Jordan Morris, Ashur Rafiev, Fei Xia, Rishad Shafik, Alex Yakovlev and Andrew Brown, "An Alternate Feedback Mechanism for Tsetlin Machines on Parallel Architectures" in International Symposium on the Tsetlin Machine (ISTM), IEEE, 2022
- [25] Zhiqiang Que, Erwei Wang, Umar Marikar, Eric A. Moreno, Jennifer Ngadiuba, Hamza Javed, Bartłomiej Borzyszkowski, Thea Aarrestad, Vladimir Loncar, Sioni Summers, Maurizio Pierini, Peter Y. K. Cheung, Wayne Luk "Accelerating Recurrent Neural Networks for Gravitational Wave Experiments" Proc. ASAP 2021 p. 117-124
- [26] Zhiqiang Que et al. "Optimizing Graph Neural Networks for Jet Tagging in Particle Physics on FPGAs" Proc. FPL 2022
- [27] A. Mokhov, A. de Gennaro, G. Tarawneh, J. Wray, G. Lukyanov, S. Mileiko, J. Scott, A. Yakovlev, A.D. Brown. "Language and hardware acceleration backend for graph processing" Proceedings Forum Design Languages (FDL) pp 71-88, 2017, IEEE.
- [28] Julian C. Shillcock, David B. Thomas, Jonathan R. Beaumont, Graeme M. Bragg, Mark Vousden, and Andrew D. Brown. "Coupling Bulk Phase Separation of Disordered Proteins to Membrane Domain Formation in Molecular Simulations on a Bespoke Compute Fabric". Membranes **12** 1 2022. DOI: 10.3390/membranes12010017

Papers written by members of the POETS team providing supporting material to the report, but not directly cited. EPSRC support is acknowledged:

- [] A.D. Brown, J.S. Reeve, K.J. Dugan and S.B. Furber, "Atomic computing - a different perspective on massively parallel problems", International conference on parallel computing, ParCo'13, Munich, August 2013. (Conference proceedings edited by M. Bader, A. Bode, H.-J. Bungartz, M. Gerndt, G.R. Joubert, F. Peters. Parallel Computing: Accelerating Computational Science and Engineering (CSE). Advances in Parallel Computing 25, IOS Press, 2014.)
- [] Mahyar Shahsavari, Jonny Beaumont, David Thomas and Andrew Brown, "POETS: A parallel cluster architecture for Spiking Neural Network" Jul 2021, International Journal of Machine Learning and Computing. 11, 4, p. 281-285
- [] Jessica Vandebon, José Gabriel F. Coutinho, Wayne Luk, Eriko Nurvitadhi, Tim Todman "Artisan: a Meta-Programming Approach For Codifying Optimisation Strategies" Proc. FCCM 2020, p. 177-185
- [] Andrew Brown, Mark Vousden, Alexander Rast, Graeme McLachlan Bragg, David Thomas, Jonny Beaumont, Matthew Naylor, and Andrey Mokhov. "POETS: Distributed event-based computing-scaling behaviour". The International Conference on Parallel Computing (ParCo) Prague, Czech Republic. 10-13 Sep 2019. https://eprints.soton.ac.uk/432964/1/2019_ParCo_POETS_Distributed_event_based_computing_scaling_behaviour.pdf
- [] A.D. Brown, D.B. Thomas, J.S. Reeve, G. Tarawneh, A De Gennaro, A.A. Mokhov, M.F. Naylor and T.J. Kazmierski, "Parallel Computing is Everywhere" IOS Press BV, Vol. 32, p. 583-592 10 p. (Advances in Parallel Computing)



- [] Ghaith Tarawneh, Andrey Mokhov, Matthew Naylor, Alex Rast, Simon W. Moore, David B. Thomas, Alex Yakovlev, Andrew Brown "Programming Model to Develop Supercomputer Combinatorial Solvers" 46th International Conference on Parallel Programming Workshop, 2017, pp 171-179 doi 10.1109/ICPPW.2017.35
- [] K.J. Dugan, A.D. Brown, J.S. Reeve, R.M. Mills and S.B. Furber "Reliable computation with unreliable computers", DATE-15, Grenoble, 2015
- [] Ashur Rafiev, Alex Yakovlev, Ghaith Tarawneh, Matthew Naylor, Simon Moore, David Thomas, Graeme Bragg, Mark Vousden, and Andrew Brown. "Synchronization in Graph Analysis Algorithms on the POETS Many-core Architecture". IET Computers & Digital Techniques **16** pp 71-88 2022. DOI: 10.1049/cdt2.12041

Papers written by members of the POETS team awaiting referees comments; EPSRC support is acknowledged:

- [] Jordan Morris, Ashur Rafiev, Graeme Bragg, Mark Vousden, David Thomas, Alex Yakovlev, Andrew Brown "An event driven approach to genotype imputation on a RISC-V FPGA cluster" IEEE Transactions on Computational Biology and Bioinformatics 2022.
- [] Tim Todman, Wayne Luk "Combining enhancements to event-driven processor Networks" Submitted to FPT2022
- [] Mark L. Vousden, Jordan Morris, Graeme M. Bragg, Jonathan R. Beaumont, Ashur Rafiev, Wayne Luk, David B. Thomas, and Andrew D. Brown. "Event-Based High Throughput Computing". IET Computers and Digital Techniques.
- [] Jonathan R. Beaumont, Andrew D. Brown, David B. Thomas, Julian C. Shillcock, Matthew F. Naylor, Graeme M. Bragg, Mark L. Vousden, Simon W. Moore, and Shane T. Fleming. "POETS: An event-driven approach to Dissipative Particle Dynamics". ACM Transactions on Parallel Computing.
- [] Julian C. Shillcock, David B. Thomas, John H. Ipsen and Andrew D. Brown, "Structure of a model biomolecular condensate is insensitive to its crowded environment", Biology