Multi-FPGA multi-core overlay architectures for executing graph-based applications

David Thomas, University of Southampton

Andrew Brown, Graeme Bragg, Mark Vousden, Thomas Bain
Simon Moore, Mathew Naylor
Wayne Luk, Tim Todman, Shane Fleming, Jonathon Beaumont, Mahyar Shahsavari
Alex Yakovlev, Ashur Rafiev, Jordan Morris

Based on work supported by URKI/EPSRC through grants EP/X036006/1 and EP/N031768/1

Overview

- POETS : methodology and toolchain for fine-grained event-triggered computing
  - Intended to target fine-grain concurrent meshes of CPUs
- Tinsel : an existing overlay architecture for executing POETS applications
  - Achieved 50K hardware threads across 48 FPGAs
- SONNETS : work-in progress approach for new multi-FPGA overlay approach
  - Design space exploration for 100s of CPUs in commodity hardware and IP

POETS : The big idea

Four partner project running from 2016 – 2023:

"Create a new framework for developing and executing event-triggered applications using asynchronous algorithms in distributed hardware"

- Research challenges:
  - ***Applications***: what should event-first algorithms look like?
  - ***Languages:*** what language do we use?
  - ***Compilation***: how do we describe and compile such applications?
  - ***Hardware***: what does this distributed hardware look like?

POETS : What we achieved over six years

- **Applications**: portfolio of asynchronous event-based applications
  - Flagships are graph processing and “Dissipative Particle Dynamics (DPD)”
  - Allowed us to provide large speed-up for published computational chemistry research

- **Compilers**: multiple compilers and simulators for one language
  - Performs place and route for applications with 1M+ logical threads
  - Includes run-time management for input and output

- **Architecture**: bespoke CPU architecture and network called “Tinsel”
  - Custom RISC-V architecture with deeply embedded routed network
  - Supports 50K hardware threads across 50 FPGAs in a low-latency communication mesh

Challenges

- Asynchronous software design is… hard
  - Message ordering is non-deterministic
  - Can't rely on a shared global clock

- We have to solve a number of big problems
  - ***Performance***: creating enough parallelism
  - ***Correctness***: avoiding deadlock and livelock
  - ***Exfiltration***: getting the answer out

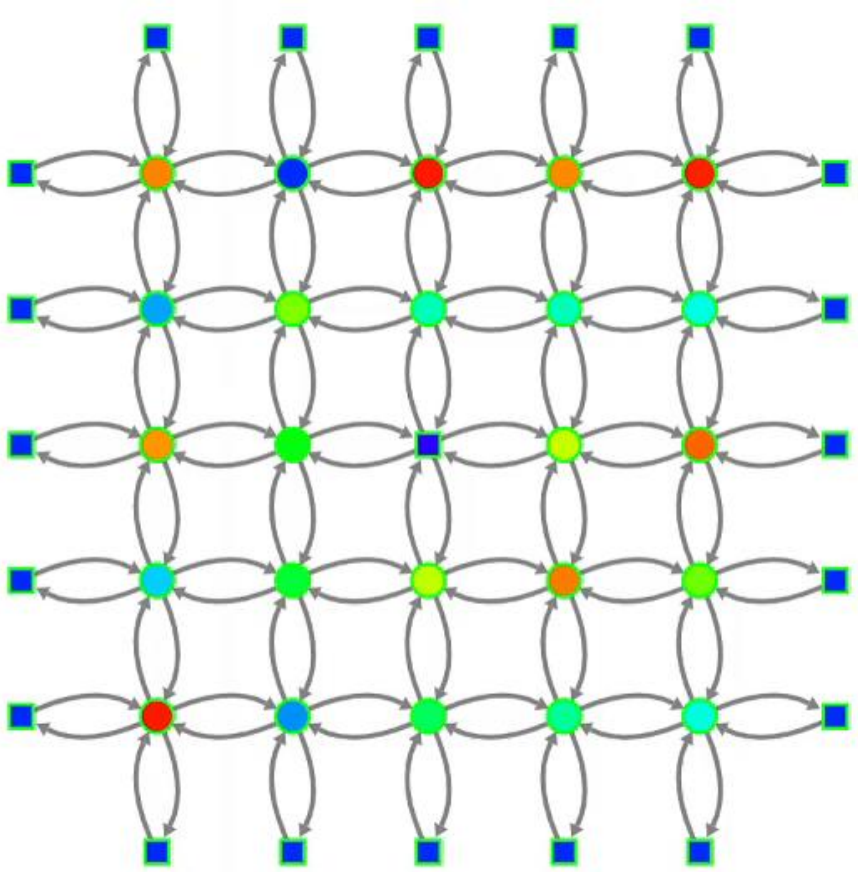
ETC : Event Triggered Computing

- Applications are split into ***devices***
  - "device" = finite state machine
- Device state is a tiny part of the global state
  - No shared memory – only messages
- Devices interact through ***event*** handlers:
  - ***Receive***: a message *m* is sent to device *d*

$$d' = \text{receive_handler}(\, d, m \,)$$

  - ***Send***: device *d* sends a message *m*

$$(d', m) = \text{send_handler}(\, d \,)$$



State changes only occur on send or receive

All state changes are atomic

Hardware schedules both sends and receives

ETC: Inversion of control

University of Southampton

```
class MyActor
{
  int state;

  void run()
  {
        while(1){
            msg = recv();
            state = receive_handler(state, msg);

            while( more_messages(state) ){
                (state,msg) = send_handler(state);
                send( msg );
            }
        }
    }
};
```

Software is “in control”
Devices are finite state
Hardware buffering is un-bounded

```
class MyDevice
{
    int state;

    void on_recv(const Message &msg)
    {
        state = receive_handler(state, msg);
    }

    bool ready_to_send() const
    {
        return more_messages(state);
    }

    void on_send(Message &msg)
    {
        (state,msg) = send_handler(state);
    }
};
```

Hardware+compiler is “in control”
Software must wait for network capacity
Both devices and hardware can be finite state

Simple Graph Example: Single Source Shortest Path

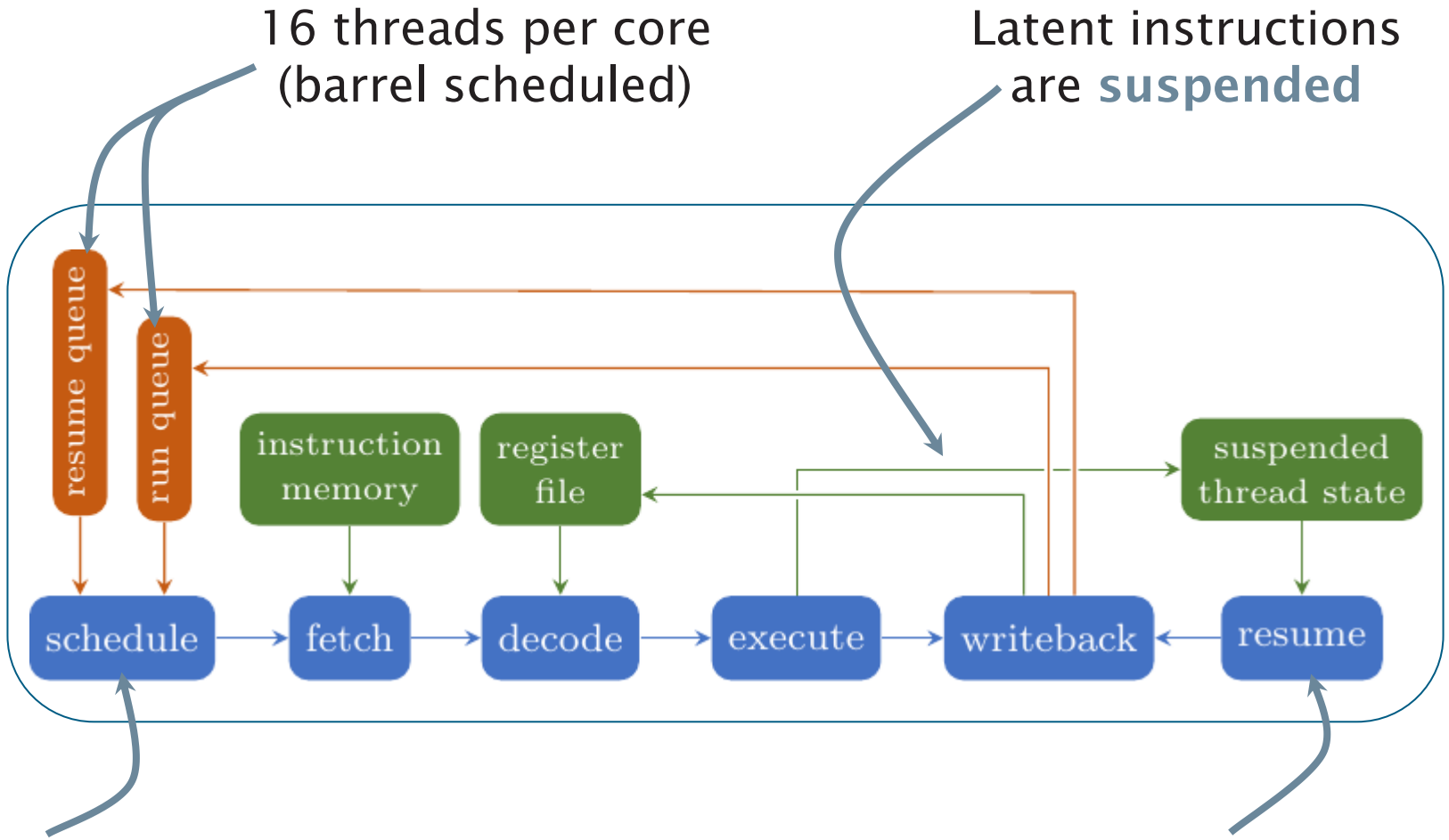
- Messages give distance from source over edge
- State is the current distance to source
- Receive message:
  - Calculate transitive distance from target
  - Update vertex to new distance if it is lower
- Ready to send if new distance not yet shared
- Send message:
  - Send message to share new distance
  - Mark state as not ready to send

```
struct Message {
  int distance;
};

struct State {
  int currDistance;
  bool isDirty;

  void on_recv(const Message &msg) {
    int newDistance = msg.distance + 1;
    if (newDistance < currDistance) {
      currDistance=newDistance;
      isDirty=true;
    }
  }

  bool ready_to_send()
  { return isDirty; }

  void on_send(Message &msg) {
    msg.distance = currDistance;
    isDirty = false;
  }
};
```

What should ETC hardware look like?

- Opportunities : the ETC model is designed to enable scalable hardware
  - Millions of asynchronous devices (small FSMs) provides scheduling freedom
  - Loose messaging and execution timing allows for latency hiding
  - Very constrained ETC software model reduces need for full OS services
  - Small messages : typically 16-64 bytes
- Constraints : the ETC model pushes all messaging and scheduling into hardware
  - Thousands of small FSMs need to be scheduled on each “CPU”
  - Messaging needs to be integrated at a deep level
  - Compute schedule is driven by unpredictable message arrival times
  - Needs to be highly tolerant of memory latency and irregular access patterns
  - *Practically: it had to be something we could prototype and evaluate using lots of FPGAs*

Tinsel core: multithreaded RV32IMF

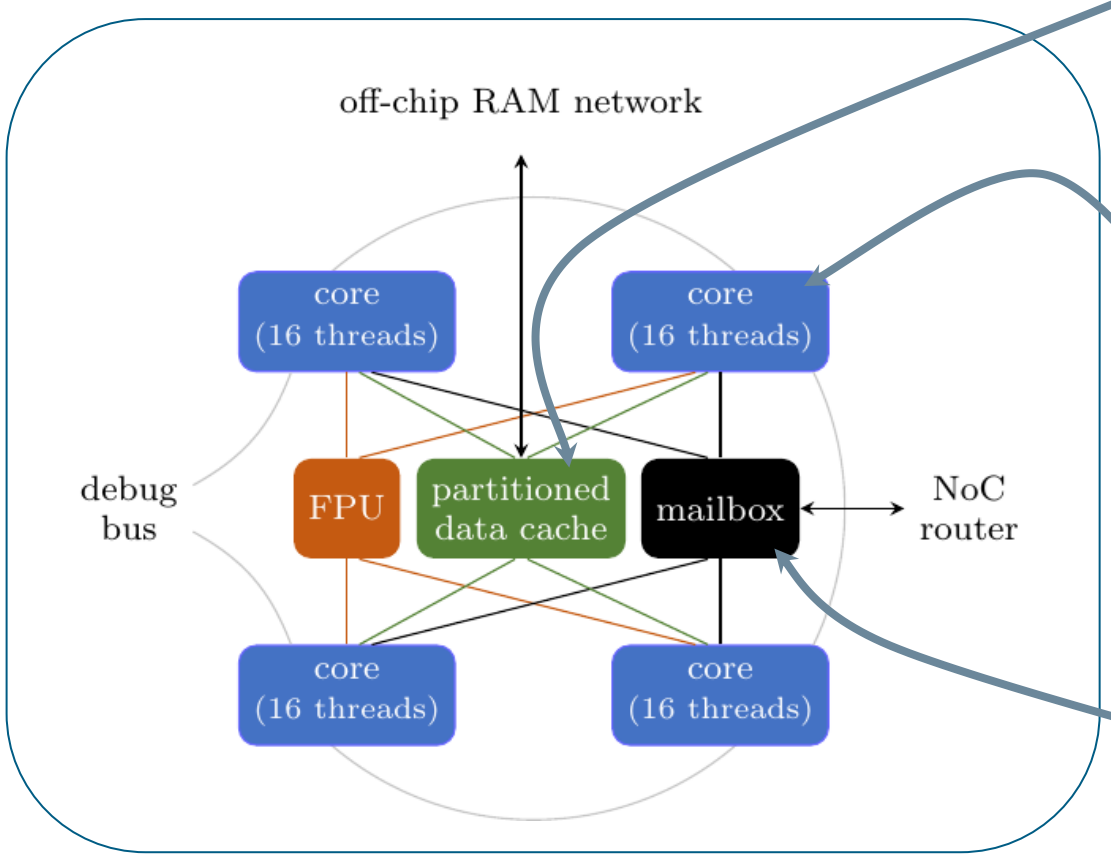


For Stratix V:

Metric	Value
Area (ALMs)	500
Fmax (MHz)	450
MIPS/LUT	0.9

Naylor, M., Moore, S., & Thomas, D. (2019). *Tinsel: a manythread overlay for FPGA clusters*, FPL

Tinsel tile: FPUs, caches, mailboxes



Data cache: **no global shared memory**

Custom instructions for **message-passing**

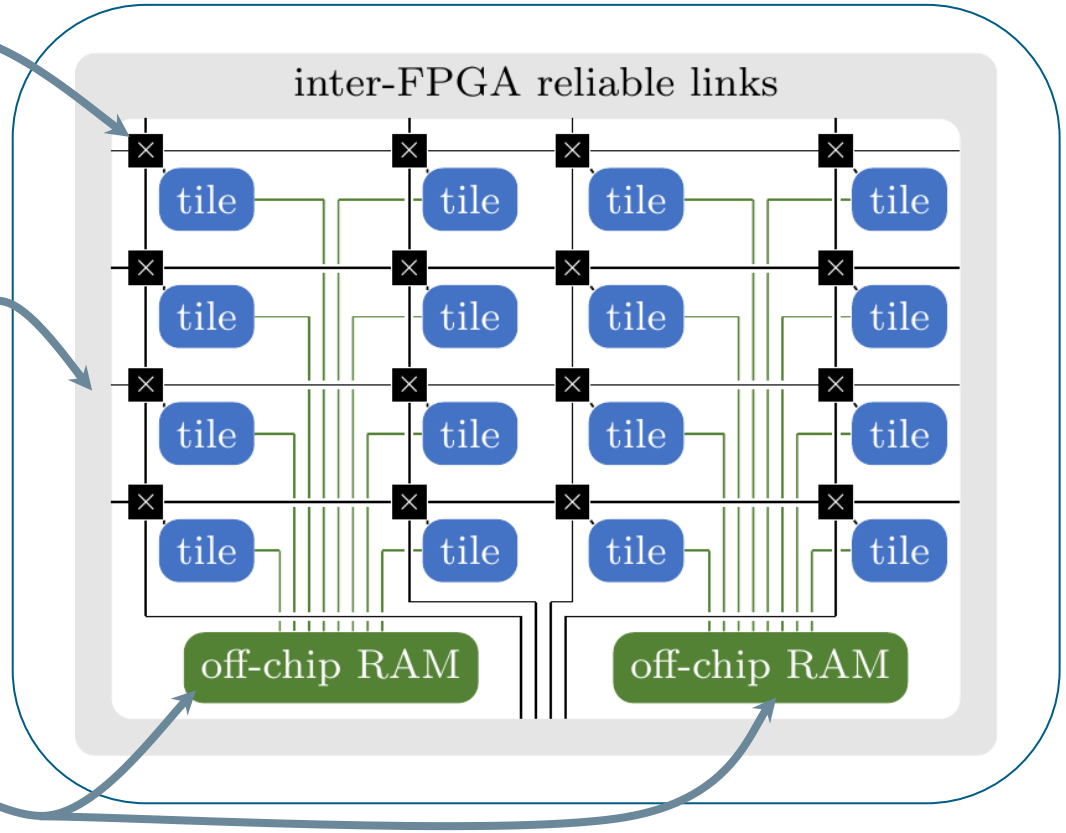
Mixed-width memory-mapped scratchpad

Tinsel network-on-chip

2D dimension-ordered router

Reliable inter-FPGA links: N, S, E and W

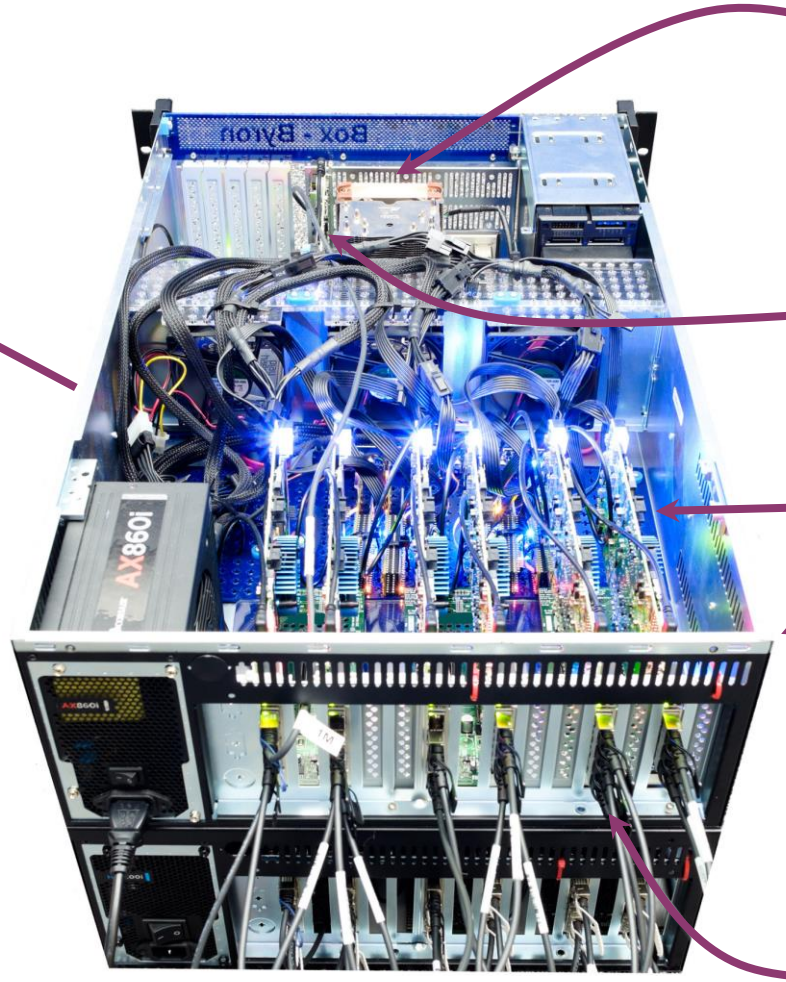
2 × DDR3 DRAM and 4 × QDRII+ SRAM in total



Separate message and memory NoCs reduce **congestion** and avoid message-dependant **deadlock**

Tinsel cluster





Modern x86 CPU

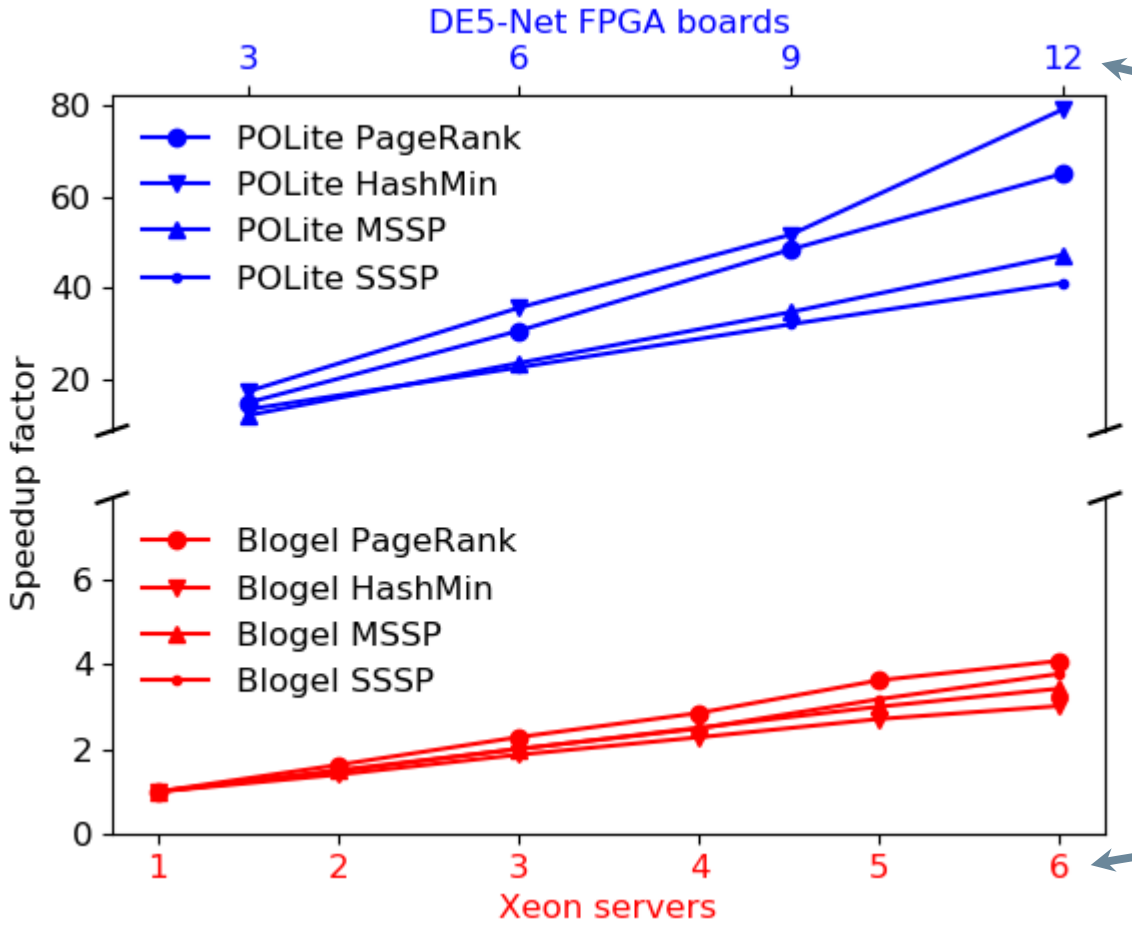
PCIe bridge FPGA

6 × worker
DE5-Net FPGAs

2 × 4U server
boxes

3 × 4 FPGA mesh
over 10G SFP+

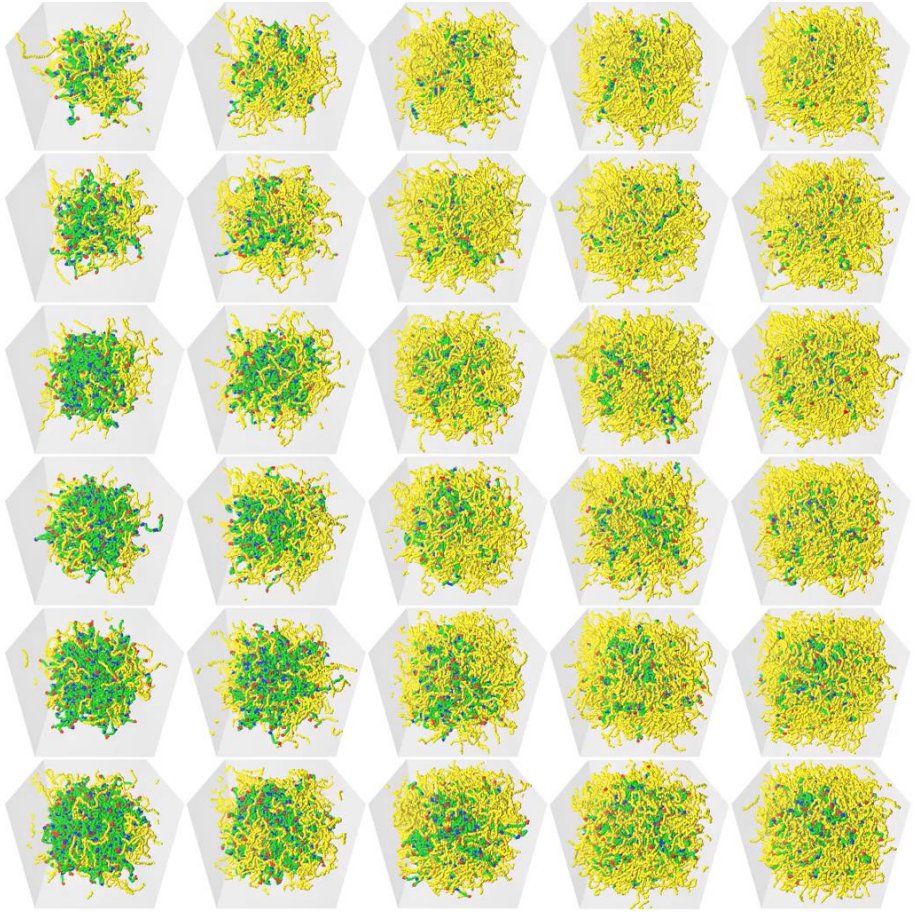
Xeon cluster versus FPGA cluster for graph processing



12 DE5s and 6 Xeons consume same power

POETS : Strengths

- Complete application+toolchain+hardware stack
  - Bespoke all the way down
- Large-scale hardware evaluation platform
  - 50,000 threads in one low-latency network
  - Scaled to 50 FPGAs (that's all we could afford)
- It worked!
- Good performance in many application domains
- It exceeded standard performance in *some* domains
  - Graph processing
  - Certain types of molecular dynamics applications



Andrew Brown et. al. (2022) POETS: An event-driven approach to Dissipative Particle Dynamics. *ACM Transactions on Parallel Computing*.

POETS : Weaknesses

- Old FPGAs (Stratix V)
- Custom inter-FPGA links
- Mesh-based inter-FPGA routing
- Fixed on-chip mesh topology
- Single fixed-size overlay

SONNETS

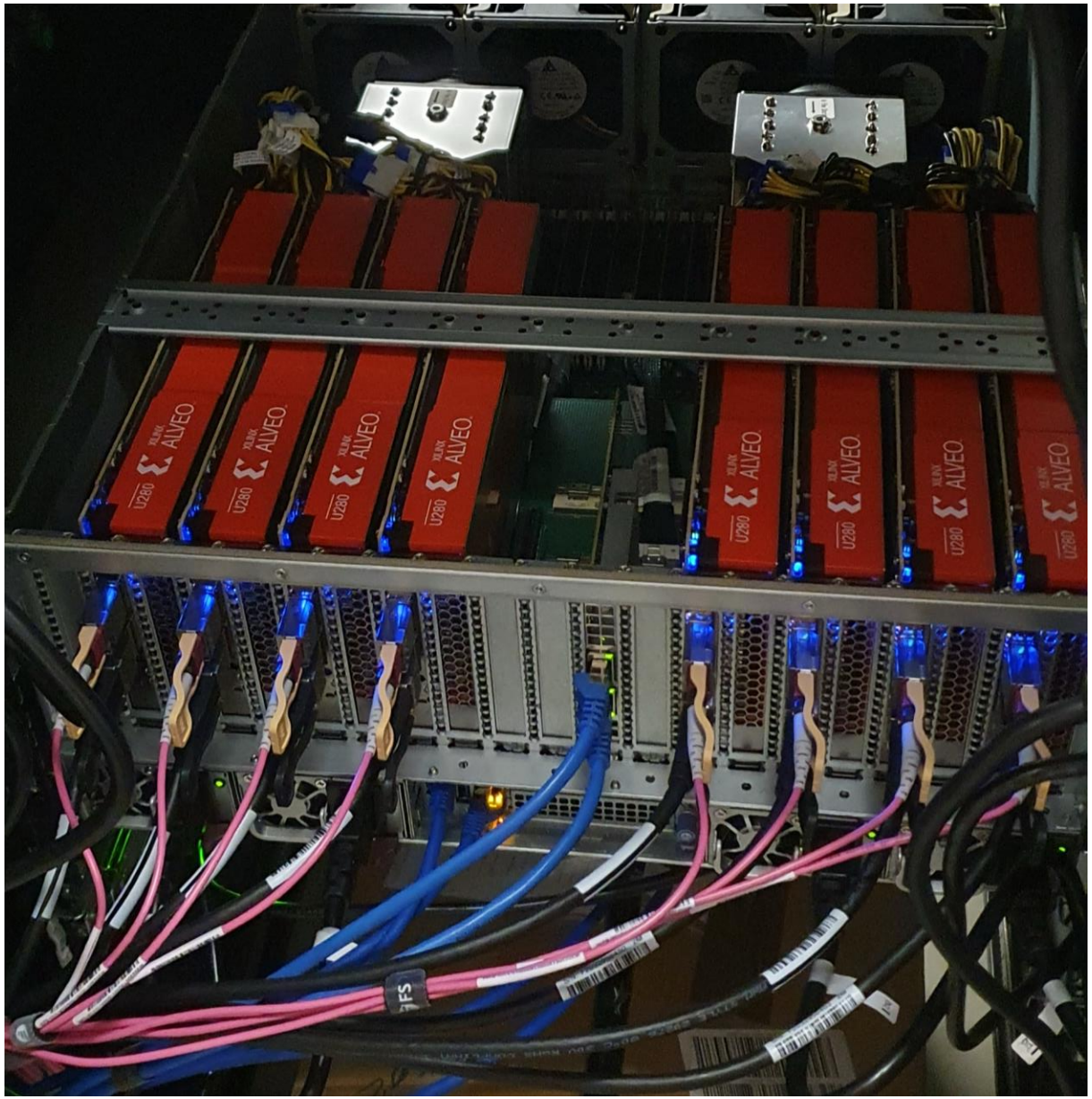
- New-ish FPGAs
- Standard 100Gb Ethernet
- Standard networking switch
- Variable on-chip topology
- Overlay customised for application
- Can we use HBM for messaging?

Just out of interest:

- Off-the shelf IP wherever possible
- Standard vendor AXI Interconnect
- Standard vendor CPU IPs

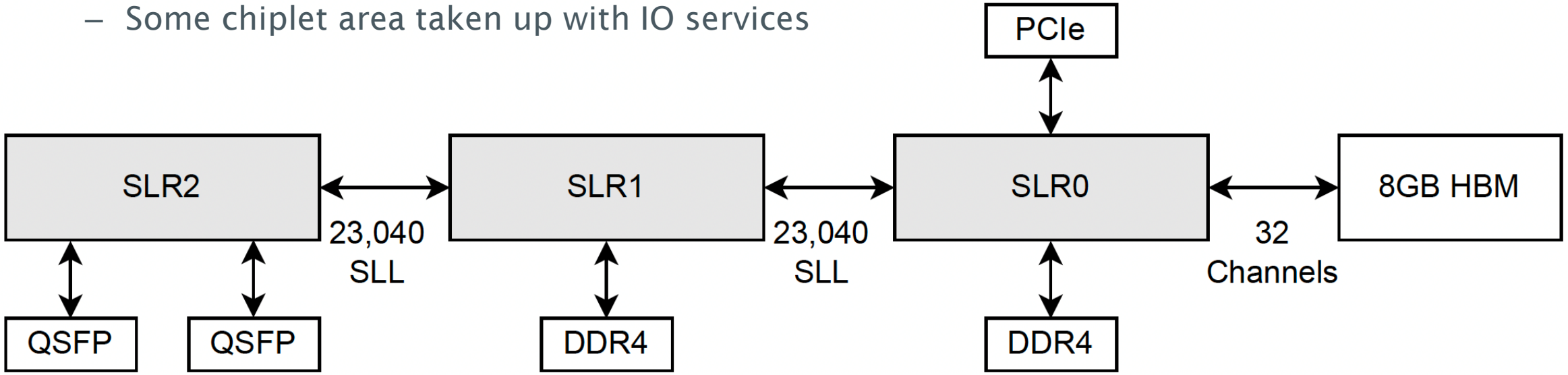
Hardware : Ennui Cluster

- 24 x U280 Alveo boards from Xilinx
  - 2x100Gb Ethernet
  - 1M LUTs
  - 8GB of HBM2
    - 460 GB/sec
    - 32 x 512bit channels
- Three host x86 servers
  - 8 U280s per server
- 48 port 100Gbe Switch



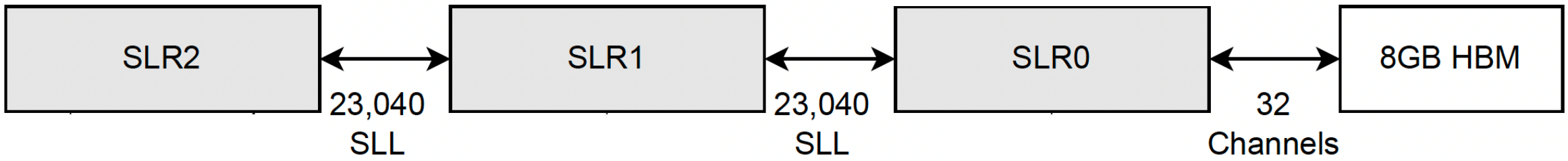
Initial problems: optimising a single FPGA full of CPUs

- Big FPGAs have become very heterogenous
  - Logic is split across SLRs (chiplets)
  - IO and RAM is attached to specific SLRs
  - SLLs between SLRs become a contested resource
  - Different chiplets are different sizes
  - Some chiplet area taken up with IO services



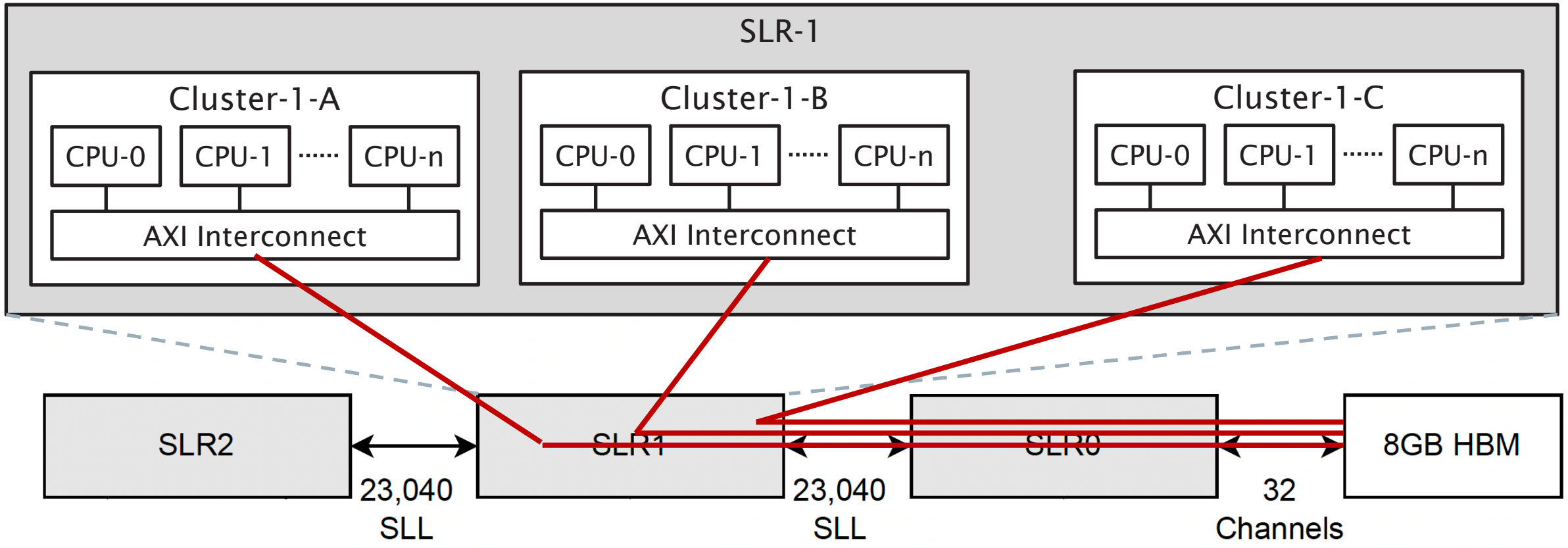
Initial problems: optimising a single FPGA full of CPUs

- Trying to organise access to HBM is a particular problem
  - Only SLR0 has direct access
  - All other SLRs must access through one or more SLL hops
- How do we choose how many CPUs per SLR?
- How do we allocate memory channels to SLLs?
  - How many? What width?



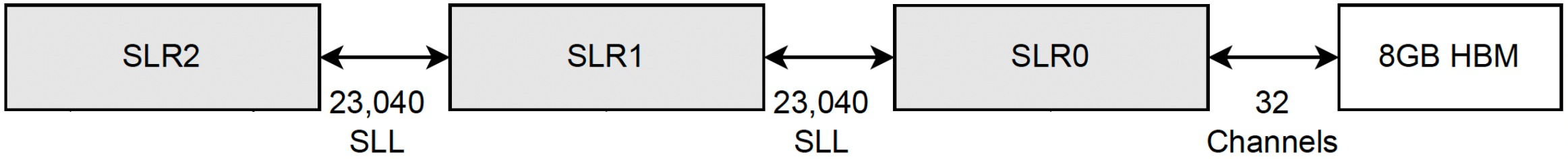
Design space exploration for a single FPGA

- SLRs are broken into clusters of cores
- Each cluster has an AXI memory mapped connection through to HBM



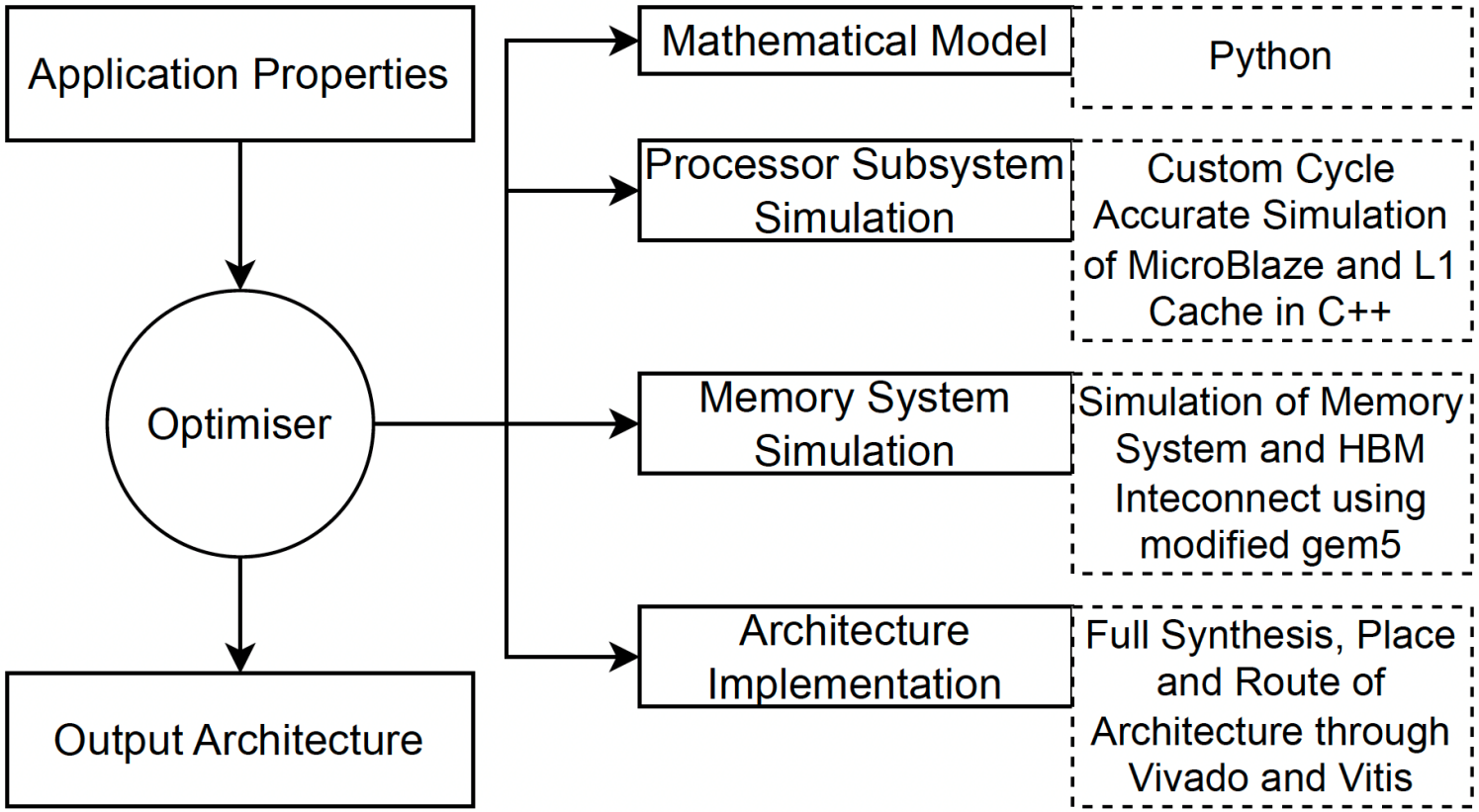
Design space exploration for a single FPGA

- SLRs are broken into clusters of cores
- Each cluster has an AXI memory mapped connection through to HBM
- Key design parameters
  - How many CPUs per cluster?
  - How many clusters per SLR?
  - What are the CPU parameters (cache size, FUs, depth, …)?
  - What are the bus parameters (width)?
- We need to optimise these parameters for a given application



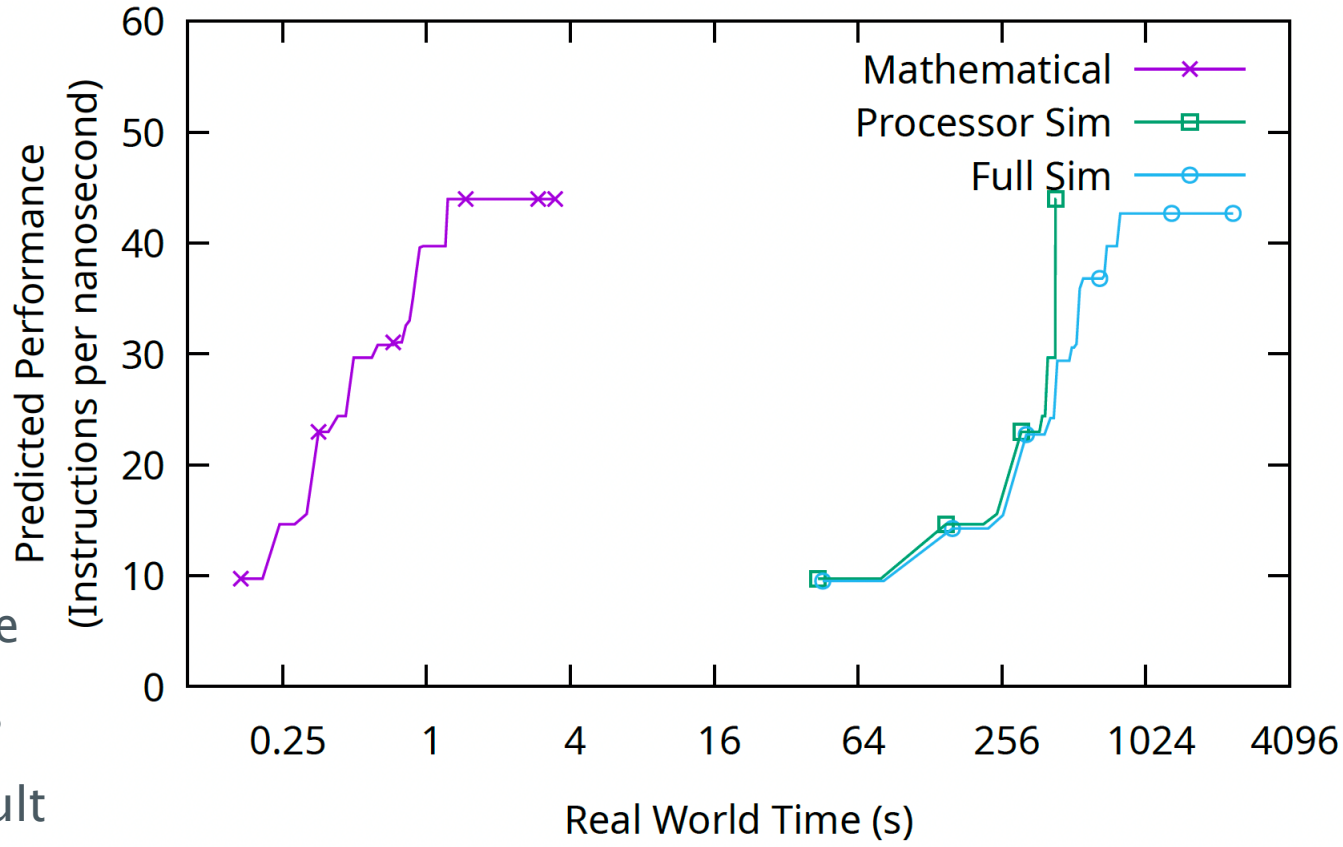
Design space exploration for a single FPGA

- Application properties are used to drive optimisation using three models (or reality)



Example : Optimising for a Shortest Path application

- Human initial estimate:
  - 32 clusters of 6 CPUs (192 CPUs)
  - Achieved 32B Instr/Sec
- Design space exploration solution:
  - 18 clusters of 14 CPUs (252 CPUs)
  - Achieved 42B Instr/Sec
- DSE gives rapid feedback
  - 4 seconds to an initial (good) estimate
  - 4 minutes to polish using simulations
  - 16 hours till first implementation result



Ongoing and future work

- Intra-FPGA communication : currently relies on HBM-based post-boxes
  - Dedicated messaging network is needed
- Inter-FPGA communication : so far we've only pinged between the U280s
  - Not clear what level of the network stack to target with very small messages
- CPU optimisation : we need to add back in latency tolerance
  - Currently getting an IPC of about 0.25, mainly due to memory stalls
  - Will probably go back to a Tinsel-like barrel scheduled CPU, e.g. BRISKI

Conclusion

- Processor overlays can be competitive, even with the FPGA tax
  - Need applications and toolchains that are designed to exploit them
- Tinsel: the full custom CPU overlay approach works well
  - Ideally suited to graph processing applications
  - Also works in less obvious domains such as particle dynamics
  - Weaknesses: difficult to deploy; difficult to customise for applications
- SONNETS : commodity everything works quite well too
  - Can usefully achieve 256+ cores using commodity IP for CPUs and interconnect
  - Strengths: very feasible to customise the overlay based on the application
  - Challenges: irregulatory of modern chiplet FPGAs makes DSE a must-have