Tinsel: a manythread overlay for FPGA clusters

POETS Project (EPSRC)

Matthew Naylor,
Simon Moore,
University of Cambridge

David Thomas,
Imperial College London

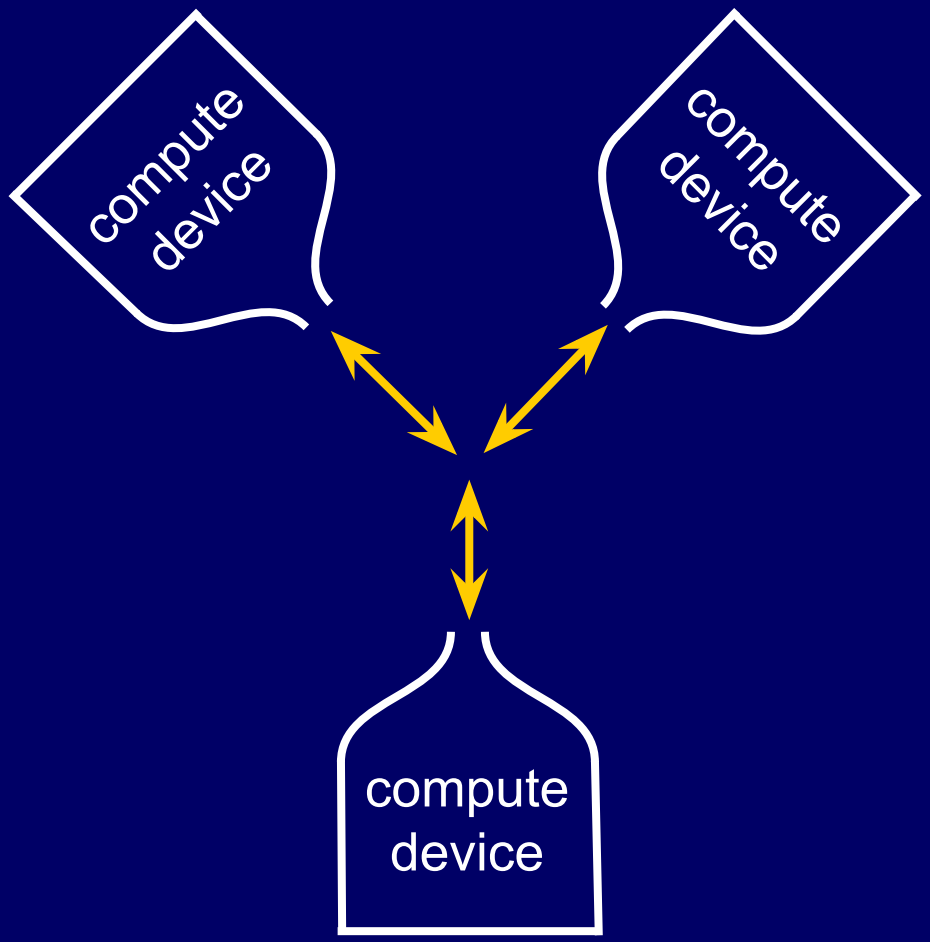
New compute devices allow ever-larger problems to be solved.

But there’s always a larger problem!

And **clusters** of these devices arise.

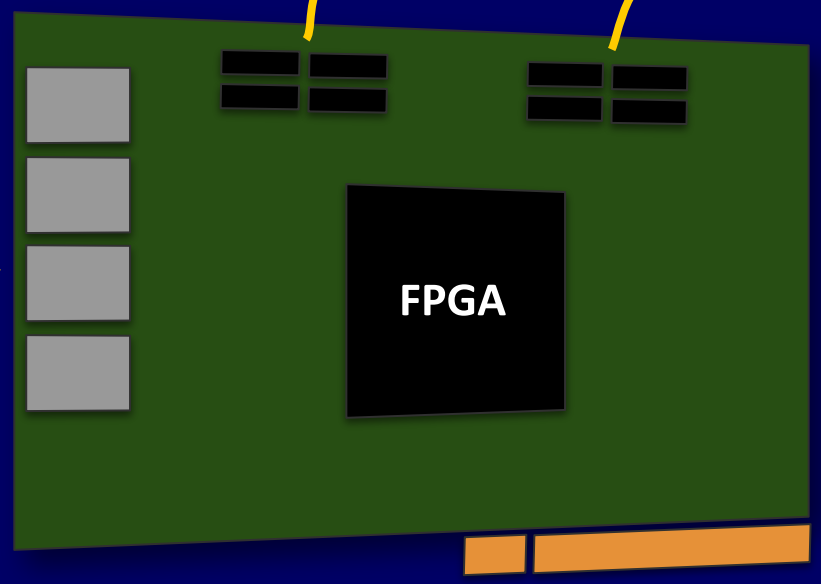
(Not just size: fault-tolerance, cost, reuse)

The communication bottleneck



Communication: an FPGA's speciality

SATA connectors, **6 Gbps** each



state-of-the-art network interfaces,
10-100 Gbps each

8-16x PCIe lanes,
10 Gbps each

Developer productivity

is a major factor blocking wider adoption of FPGA-based systems:

- FPGA knowledge & expertise
- Low-level design tools
- Long synthesis times

This paper

To what extent can a distributed **soft-processor overlay*** provide a useful level of performance for FPGA clusters?

* programmed in software at a **high-level of abstraction**

The Tinsel overlay

How to tolerate latency?

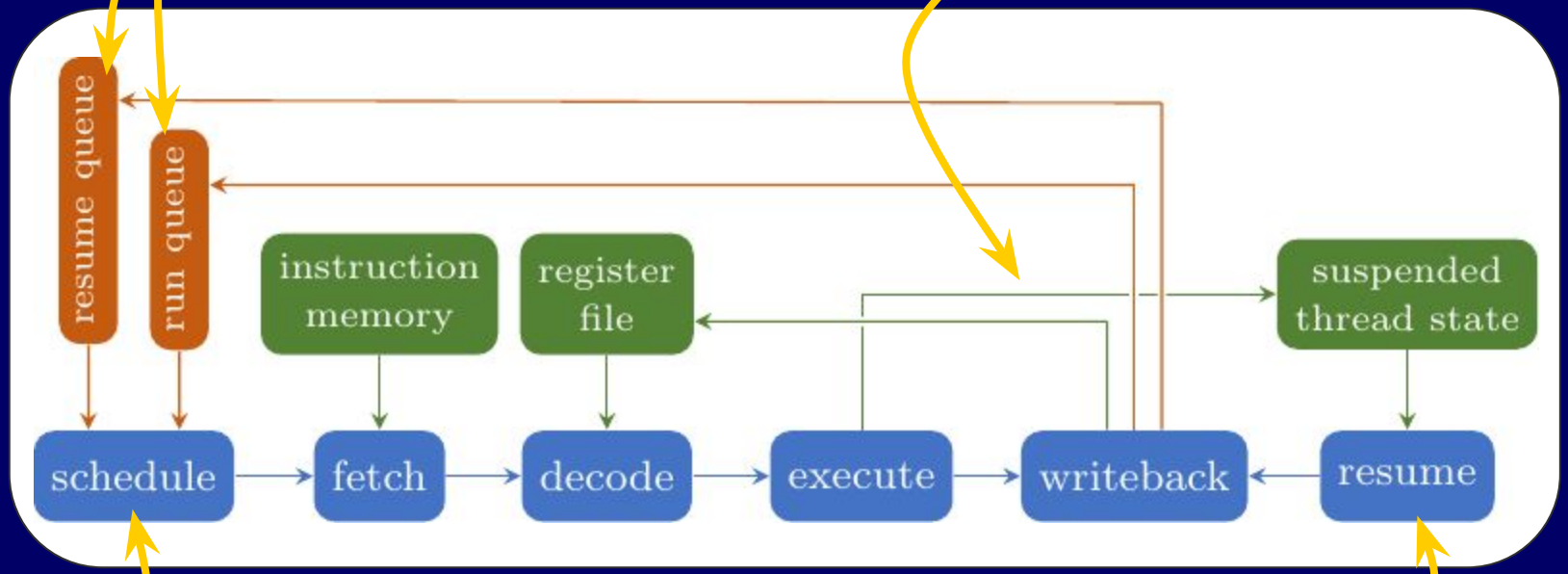
Many sources of latency to a soft-processor:

- Floating-point
- Off-chip memory
- Parameterisation & resource sharing
- Pipelined uncore to keep Fmax high

Tinsel core: multithreaded RV32IMF

16 or 32 threads
per core
(barrel scheduled)

Latent instructions
are **suspended**



One instruction per thread in pipeline
at any time: **no control / data hazards**

Latent instructions
are **resumed**

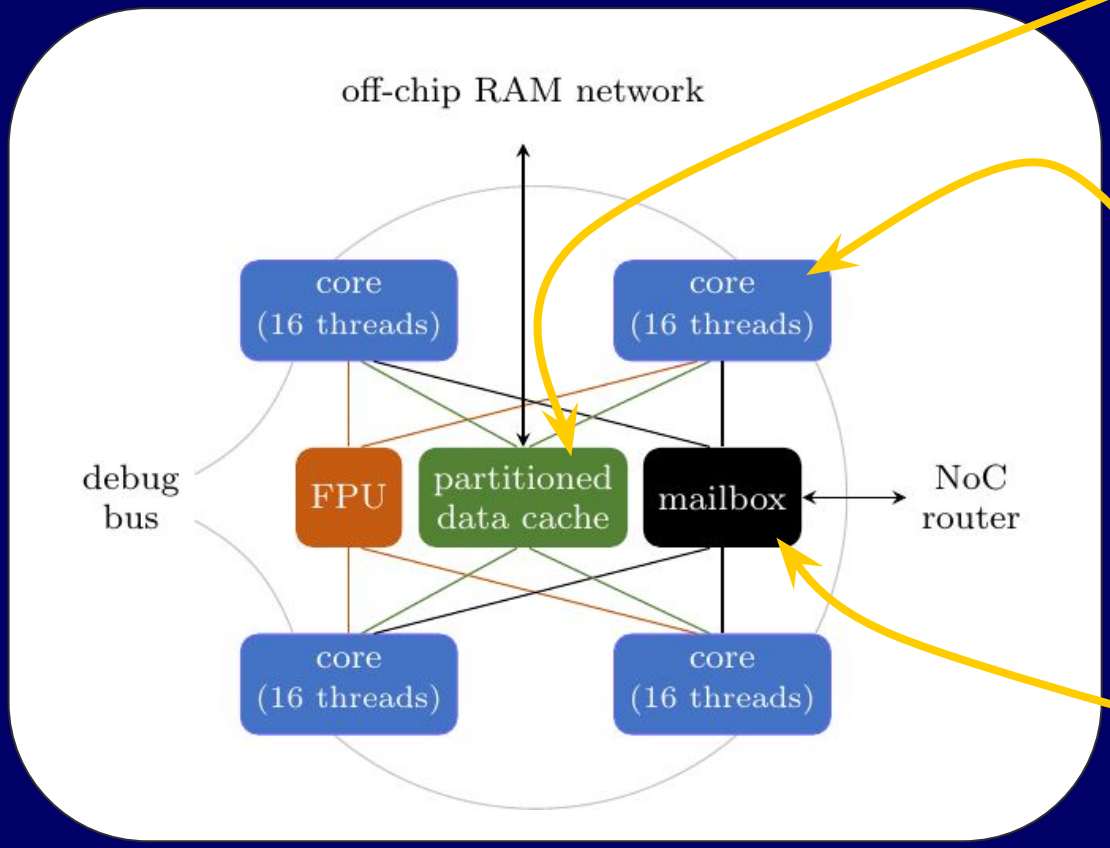
No hazards ⇒ small and fast

A single RV32I 16-thread Tinsel core with tightly-coupled memories:

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

*assuming a highly-threaded workload

Tinsel tile: FPUs, caches, mailboxes



Data cache: **no global shared memory**

Custom instructions for **message-passing**

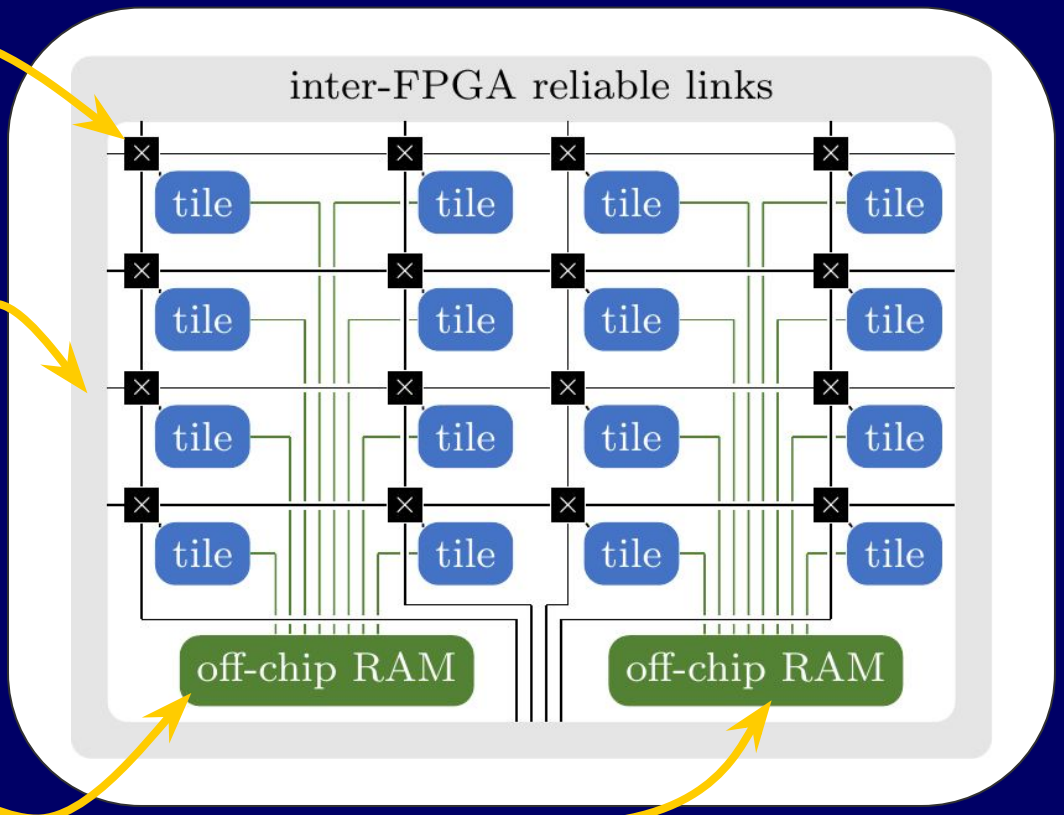
Mixed-width memory-mapped scratchpad

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
(now 8 boxes)

3 × 4 FPGA mesh
over 10G SFP+

Distributed termination detection

Custom instruction for fast distributed termination detection over the entire cluster:

```
int tinselIdle(bool vote);
```

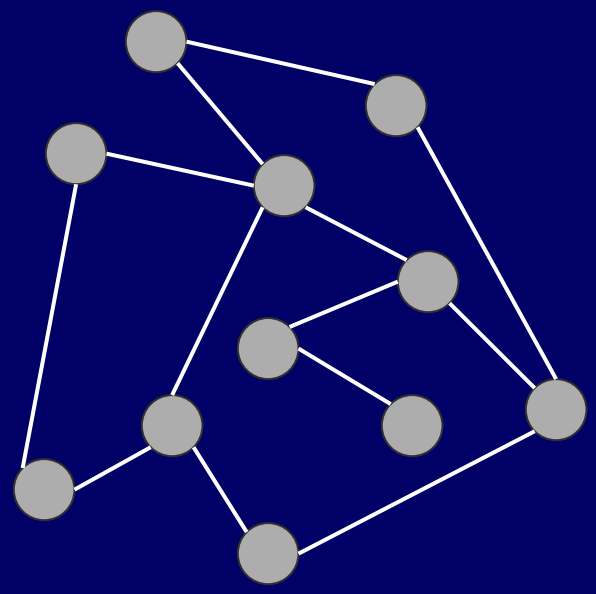
Returns `true` if all threads are in a call to `tinselIdle()` and no messages are in-flight.

Greatly simplifies and accelerates both **synchronous** and **asynchronous** message-passing applications.

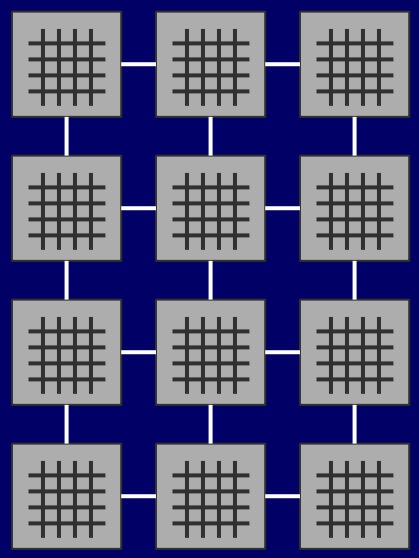
POLite: high-level API

POLite

Application graph
defined by POLite API
(**vertex-centric** paradigm)



Tinsel cluster



POLite: Types

Vertex state

Edge properties

Message type

```
template <typename S, typename E, typename M>
  struct PVertex {
    // State
    S* s;
    PPin* readyToSend;

    // Event handlers
    void init();
    void send(M* msg);
    void recv(M* msg, E* edge);
    bool step();
    bool finish(M* msg);
 };
```

No: the vertex doesn't want to send.

Pin(p): the vertex wants to send on pin p.

HostPin: the vertex wants to send to the host.

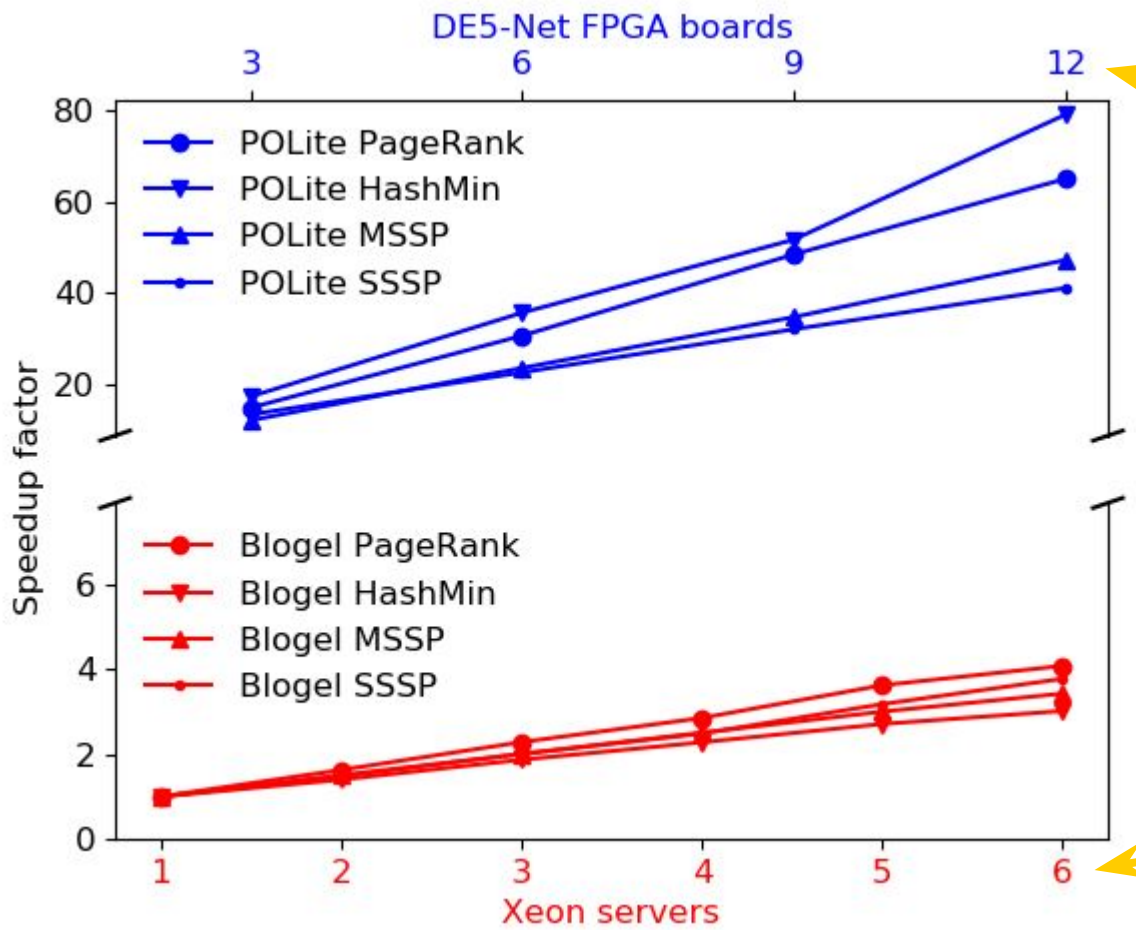
POLite SSSP (asynchronous)

```
// Each vertex maintains an int
// representing the distance of
// the shortest known path to it
//
// Source vertex triggers a
// series of sends, ceasing
// when all shortest paths
// have been found.

// Vertex state
struct SSSPState {
  // Is this the source vertex?
  bool isSource;
  // The shortest known
  // distance to this vertex
  int dist;
};
```

```
// Vertex behaviour
struct SSSPVertex : PVertex<SSSPState,int,int> {
  void init() {
    *readyToSend = s->isSource ? Pin(0) : No;
  }
  void send(int* msg) {
    *msg = s->dist;
    *readyToSend = No;
  }
  void recv(int* dist, int* weight) {
    int newDist = *dist + *weight;
    if (newDist < s->dist) {
      s->dist = newDist;
      *readyToSend = Pin(0);
    }
  }
  bool step() { return false; }
  bool finish(int* msg) {
    *msg = s->dist;
    return true;
  }
};
```

Performance results

Xeon cluster versus FPGA cluster



12 DE5s and 6 Xeons consume same power

Performance counters

From POLite versions of PageRank on 12 FPGAs:

Metric	Sync	GALS
Time (s)	0.49	0.59
Cache hit rate (%)	91.5	93.9
Off-chip memory (GB/s)	125.8	127.7
CPU utilisation (%)	56.4	71.3
NoC messages (GB/s)	32.2	27.2
Inter-FPGA messages (Gbps)	58.4	48.8

Comparing features, area, Fmax

Feature	Tinsel-64	Tinsel-128	µaptive
Cores	64	128	120
Threads	1024	2048	120
DDR3 controllers	2	2	0
QDRII+ controllers	4	4	0
Data caches	16 × 64KB	16 × 64KB	0
FPUs	16	16	0
NoC	2D mesh	2D mesh	Hoplite
Inter-FPGA comms	4 × 10Gbps	4 × 10Gbps	0
Termination detection	Yes	Yes	No
Fmax (MHz)	250	210	94
Area (% of DE5-Net)	61%	88%	100%

Conclusion 1

Many advantages of a **multithreading** on FPGA:

- No hazard avoidance logic (small, high Fmax)
- No hazards (high throughput)
- Latency tolerance (high throughput, resource sharing, deeply pipelined uncore e.g. FPUs, caches)

Conclusion 2

Good performance possible from an FPGA cluster programmed in software at a high-level when:

- the off-FPGA bandwidth limits (memory & comms) are approached by a modest amount of compute;
- e.g. the distributed vertex-centric computing paradigm.

Funded by

EPSRC

Contact: *matthew.naylor@cl.cam.ac.uk*

Website: *https://github.com/POETSII/tinsel*

POETS partners

Extras

Parameterisation

Subsystem	Parameter	Default value
Core	ThreadsPerCore	16
Core	CoresPerFPU	4
Core	CoresPerDCache	4
Core	CoresPerMailbox	4
Core	BytesPerInstrMem	16,384
Cache	DCachesPerDRAM	8
Cache	BytesPerBeat	32
Cache	BeatsPerLine	1
Cache	DCacheSetsPerThread	4
Cache	DCacheNumWays	8
NoC	MailboxMeshXLen	4
NoC	MailboxMeshYLen	4
NoC	BytesPerFlit	16
NoC	MaxFlitsPerMsg	4
Mailbox	MsgSlotsPerThread	16

Area breakdown (default configuration)

Subsystem	Quantity	ALMs	% of DE5
Core	64	51,029	21.7
FPU	16	15,612	6.7
DDR3 controller	2	7,928	3.5
Data cache	16	7,522	3.2
NoC router	16	7,609	3.2
QDRII+ controller	4	5,623	2.4
10G Ethernet MAC	4	5,505	2.3
Mailbox	16	4,783	2.0
Interconnect etc.	1	37,660	16.0
Total		143,271	61.0

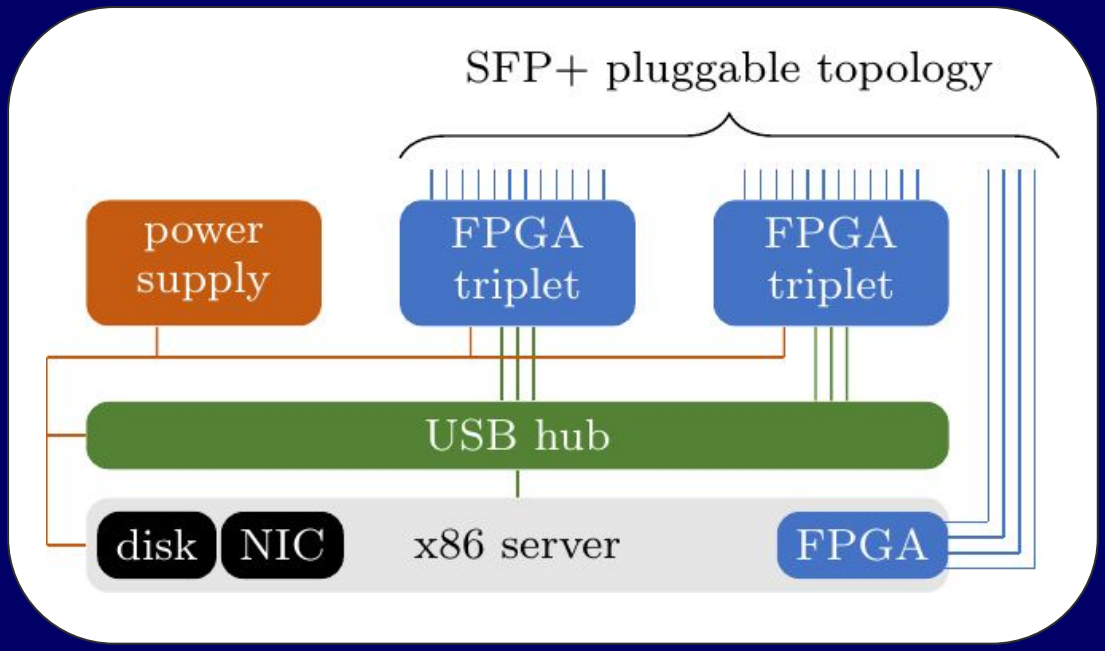
(On the DE5-Net at 250MHz.)

POLite: Event handlers

`void init();`	Called once at start of time.
`void send(M* msg);`	Called when network capacity available, and `readyToSend != No`.
`void recv(M* msg, E* edge);`	Called when message arrives.
`bool step();`	Called when no vertex wishes to send and no messages in-flight (stable state). Return true to start a new time-step.
`bool finish(M* msg);`	Like `step()`, but only called when no vertex has indicated a desire to start a new time step. Optionally send a message to the host.

POLite SSSP (synchronous)

```
// Similar to async version, but
// each vertex sends at most
// one message per time step

// Vertex state
struct SSSPState {
  // Is this the source vertex?
  bool isSource;
  // The shortest known
  // distance to this vertex
  int dist;
};

struct SSSPVertex :
        PVertex<SSSPState,int,int> {

  void init() {
    *readyToSend =
       s->isSource ? Pin(0) : No;
  }

  void send(int* msg) {
    *msg = s->dist; *readyToSend = No;
  }

  void recv(int* dist, int* weight) {
    int newDist = *dist + *weight;
    if (newDist < s->dist) {
      s->dist = newDist;
      s->changed = true;
    }
  }

  bool step() {
    if (s->changed) {
      s->changed = false;
      *readyToSend = Pin(0);
      return true;
    }
    else return false;
  }

  bool finish(int* msg) {
    *msg = s->dist; return true;
  }
};
```


SFP+ pluggable topology
power supply
FPGA triplet
FPGA triplet
USB hub
disk
NIC
x86 server
FPGA