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:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area (Stratix V ALMs)</td>
<td>500</td>
</tr>
<tr>
<td>Fmax (MHz)</td>
<td>450</td>
</tr>
<tr>
<td>MIPS/LUT*</td>
<td>0.9</td>
</tr>
</tbody>
</table>

*assuming a highly-threaded workload
Tinsel tile: FPUs, caches, mailboxes

- Custom instructions for message-passing
- Data cache: no global shared memory
- Mixed-width memory-mapped scratchpad

Diagram:
- Off-chip RAM network
- Core (16 threads) with FPU and partitioned data cache
- Mailbox connected to NoC router
- Debug bus
Tinsel network-on-chip

2D dimension-ordered router

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

2 x DDR3 DRAM and 4 x 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:

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

POLite

Tinsel cluster
Template:

```
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);
};
```
// 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 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;
    }
};

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

// POLite SSSP (asynchronous)
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:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Sync</th>
<th>GALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>0.49</td>
<td>0.59</td>
</tr>
<tr>
<td>Cache hit rate (%)</td>
<td>91.5</td>
<td>93.9</td>
</tr>
<tr>
<td>Off-chip memory (GB/s)</td>
<td>125.8</td>
<td>127.7</td>
</tr>
<tr>
<td>CPU utilisation (%)</td>
<td>56.4</td>
<td>71.3</td>
</tr>
<tr>
<td>NoC messages (GB/s)</td>
<td>32.2</td>
<td>27.2</td>
</tr>
<tr>
<td>Inter-FPGA messages (Gbps)</td>
<td>58.4</td>
<td>48.8</td>
</tr>
</tbody>
</table>
# Comparing features, area, Fmax

<table>
<thead>
<tr>
<th>Feature</th>
<th>Tinsel-64</th>
<th>Tinsel-128</th>
<th>μaptive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>64</td>
<td>128</td>
<td>120</td>
</tr>
<tr>
<td>Threads</td>
<td>1024</td>
<td>2048</td>
<td>120</td>
</tr>
<tr>
<td>DDR3 controllers</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>QDRII+ controllers</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Data caches</td>
<td>16 × 64KB</td>
<td>16 × 64KB</td>
<td>0</td>
</tr>
<tr>
<td>FPUs</td>
<td>16</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>NoC</td>
<td>2D mesh</td>
<td>2D mesh</td>
<td>Hoplite</td>
</tr>
<tr>
<td>Inter-FPGA comms</td>
<td>4 × 10Gbps</td>
<td>4 × 10Gbps</td>
<td>0</td>
</tr>
<tr>
<td>Termination detection</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Fmax (MHz)</td>
<td>250</td>
<td>210</td>
<td>94</td>
</tr>
<tr>
<td>Area (% of DE5-Net)</td>
<td>61%</td>
<td>88%</td>
<td>100%</td>
</tr>
</tbody>
</table>
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.
POETS partners
Extras
## Parameterisation

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Parameter</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>ThreadsPerCore</td>
<td>16</td>
</tr>
<tr>
<td>Core</td>
<td>CoresPerFPU</td>
<td>4</td>
</tr>
<tr>
<td>Core</td>
<td>CoresPerDCache</td>
<td>4</td>
</tr>
<tr>
<td>Core</td>
<td>CoresPerMailbox</td>
<td>4</td>
</tr>
<tr>
<td>Core</td>
<td>BytesPerInstrMem</td>
<td>16,384</td>
</tr>
<tr>
<td>Cache</td>
<td>DCachesPerDRAM</td>
<td>8</td>
</tr>
<tr>
<td>Cache</td>
<td>BytesPerBeat</td>
<td>32</td>
</tr>
<tr>
<td>Cache</td>
<td>BeatsPerLine</td>
<td>1</td>
</tr>
<tr>
<td>Cache</td>
<td>DCacheSetsPerThread</td>
<td>4</td>
</tr>
<tr>
<td>Cache</td>
<td>DCacheNumWays</td>
<td>8</td>
</tr>
<tr>
<td>NoC</td>
<td>MailboxMeshXLen</td>
<td>4</td>
</tr>
<tr>
<td>NoC</td>
<td>MailboxMeshYLen</td>
<td>4</td>
</tr>
<tr>
<td>NoC</td>
<td>BytesPerFlit</td>
<td>16</td>
</tr>
<tr>
<td>NoC</td>
<td>MaxFlitsPerMsg</td>
<td>4</td>
</tr>
<tr>
<td>Mailbox</td>
<td>MsgSlotsPerThread</td>
<td>16</td>
</tr>
</tbody>
</table>
Area breakdown (default configuration)

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

(On the DE5-Net at 250MHz.)
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;
    }

    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;
    }
};

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;
}