They key to scalability: High-performance and energy efficient data movement

Integrated Systems Laboratory (ETH Zürich)

Tim Fischer  fischeti@iis.ee.ethz.ch
Thomas Benz  tbenz@iis.ee.ethz.ch

PULP Platform
Open Source Hardware, the way it should be!
Motivation – Scalability/Energy Efficiency

• Trend towards physically larger systems
  • Today’s application, especially driven by ML, requires
    • High memory bandwidth
    • Increasingly irregular accesses throughout the hierarchy
  • Increasing demand for compute increases the system size and heterogeneity
    • With {Moore, Dennard, Koomey, Kryder,...}’s law and node scaling coming to a halt
    • (The only law still going strongly: Murphy’s law...)
  • The memory system needs to keep up

• First part: Efficient data mover
  • Increased size and heterogeneity ➞ Latency tolerant operation using many protocols
  • Efficiently handle any transfer ➞ Maximizing bus utilization, irregular transfer support
  • PULP ecosystem is diverse ➞ Create one engine that fits all
Motivation – Scalability/Energy Efficiency

• **Our own scaleout study: Occamy**
  - DMA engines scaled flawlessly
  - **But:** AXI-crossbar-based memory system reached its limit
    - One large crossbar was unrouteable without internal cuts
    - Hierarchy of crossbars is complex and increases latency

• **Second part: Scalable Interconnect**
  - Most of the area needs to be occupied by compute logic
  - The interconnect **must not** become the bottleneck
    - Energy needs to go into compute
    - Sustained high-bandwidth data-flow
Why Should We Even Talk About DMAs?

• Well-established concept
  • Intel 8257

• Many DMAs exist

• BUT:
  • Closed-source or commercial
  • Special-purpose or entangled in systems
  • Technology-dependent or vendor-locked
  • Or all of the above...

• For our Research we need:

  An technology-independent, scalable, high-performance, low-overhead, latency-tolerant, energy-efficient, extendable, modular, configurable, multiprotocol, multiport, open-source DMA engine.
• System **binding**
• Defines interaction
• Given by the **platform**

• Transfer **modification**
• High-level (e.g. **repetition**)
• Given by the **application**

• Efficient **1D data transport**
• Multiple protocols and ports (v0.5)
• Given by the **memory system**
System Bindings: Available Front-ends

- **Register-based**
  - A set of registers is used to hold a transfer
  - Our *simplest* interface
  - Expensive in *multi-hart* systems

- **Instruction-based**
  - Custom stream of instructions (*Xdma*)
  - Extremely *agile*, 3 instructions to launch
  - Requires assembly or C intrinsics

- **Descriptor-based (Linux-compatible)**
  - Transfer descriptors in memory
  - *Arbitrary shapes*, atomic launches
  - Slower, requires memory bandwidth

---

### Comparison Table

<table>
<thead>
<tr>
<th>Operation</th>
<th>rs2</th>
<th>rs1</th>
<th>rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMSRC</td>
<td>ptrhi</td>
<td>ptrlo</td>
<td>-</td>
</tr>
<tr>
<td>DMDST</td>
<td>ptrhi</td>
<td>ptrlo</td>
<td>-</td>
</tr>
<tr>
<td>DMSTR</td>
<td>dststrd</td>
<td>srcstrd</td>
<td>-</td>
</tr>
<tr>
<td>DMREP</td>
<td>00000</td>
<td>reps</td>
<td>-</td>
</tr>
<tr>
<td>DMCPY</td>
<td>config</td>
<td>size</td>
<td>dest</td>
</tr>
<tr>
<td>DMSTAT</td>
<td>status</td>
<td>-</td>
<td>dest</td>
</tr>
</tbody>
</table>

---

**As diverse as our systems!**
As Flexible As Its Applications: Mid-ends

- **Going tensors**
  - Intrinsic support for N-D tensor
  - For each dimension: strides and repetitions

- **Repeated accesses**
  - Real time usecases (currently: ControlPULP)
  - Periodically launch data **gathering** operations
  - Up to 3D to cover **irregular** address maps

- **Distributed DMA**
  - Mempool cluster
  - One frontend controlling N back-ends
  - Act as one ultra-wide DMA

- **More to come!**
The Heart: The Engine (Back-end)

• Focus one 1D transfers
• Handle them as efficient as possible
  • Coalescing, realignment
  • Support many outstanding transfers
• Optional error handler
  • Reporting errors, replaying transfers
• Multiple protocols - Core insight
  • All protocols use ready-valid handshaking
  • The transport a stream of bytes
  • Protocol ports can be abstracted
    ➔ Managers
• One backend: one stream at a time
Modularity Has A Price

- Many variants to maintain
  - Every parameter multiplies the work
  - Multiport/protocol additions (v0.5)

- Verification is hell
  - Especially with AXI Stream support
  - Multiport testbench into single TB memory
  - Golden model in SystemVerilog

- Backend is hard to parameterize
  - Difficult at best to do in SystemVerilog
  - Assemble using Mario: clean result

```verilog
module idma_backend_rw_axi_rw_tilelink #(/* MANY PARAMETERS */)
    (/* DMA TRANSFER INTERFACE */
        // AXI4+ATOP read request
        output axi_req_t axi_read_req_o,
        // AXI4+ATOP read response
        input axi_rsp_t axi_read_rsp_i,
        // TileLink-UH read request
        output tilelink_req_t tilelink_read_req_o,
        // TileLink-UH read response
        input tilelink_rsp_t tilelink_read_rsp_i,
        // AXI4+ATOP write request
        output axi_req_t axi_write_req_o,
        // AXI4+ATOP write response
        input axi_rsp_t axi_write_rsp_i,
        // TileLink-UH write request
        output tilelink_req_t tilelink_write_req_o,
        // TileLink-UH write response
        input tilelink_rsp_t tilelink_write_rsp_i,
    );
```
Example: PULP Cluster
Example: PULP Cluster

- 64 bit DMA engine
- The cluster DMA has 3 interfaces
  - 64-bit AXI4 manager R/W port
  - 2x 32-bit OBI R port
  - 2x 32-bit OBI W port
  - Configuration port
- Different options
  - A pure AXI DMA and an AXI2OBI adapter
  - A multiprotocol DMA (v0.5)
- What is the preferred option?
How Do We Handle This Protocol Mess?

- **Pure AXI** DMA (v0.4)
- X-Bar and AXI2OBI
- Simple BE, adapters

- **Single** AXI – OBI BE
- Multiprotocol (v0.5)
- Either read or write

- **Dual** unidirectional BEs
- Multiprotocol (v0.5)
- Bidirectional copy
Comparison: Is Multiprotocol Support Worth It?

- Pure AXI (v0.4) to Multiprotocol (v0.5)
  - 14% improvement over (v0.4)
  - Mainly adapters
  - Slight reduction in speed

- Dual Backend (v0.5)
  - 8% improvement over (v0.4)
  - Two simpler BEs
  - Twice the bandwidth
  - No reduction in speed

HW spent on moving data, less latency!

The Future Of iDMA Is Exciting!

• In-stream operations (similar to HWPE)
  • Matrix transposition (dedicated buffer, TCDM)
  • Streamlined reshuffling operations in TCDM
  • Simple arithmetic instructions

• Virtual memory support
  • Direct translation to VA in a mid-end

• AXI-Pack™ support
  • Efficient transport of strided and irregular streams

• Near-memory smart DMA (PIM)
  • Moving DMA engines closer to memory
  • Adding intelligence directly to these DMAs
  • Data streams as service

10 Years Of PULP, Lugano June 2023
Occamy is powered by the iDMA!
Occamy is powered by the iDMA!

- **Chiplet-based system**
  - Compute Chiplet
  - HBM Chiplet
  - Off-chip Serial Link

- **24 Compute Clusters**
  - Each with their own DMA
  - 128KB SPM
  - Wide 512-bit buses

- **Programmable DMAs**
  - Low-latency Cluster2Cluster transfers
  - High-latency HBM/Off-chip data-movement

**Most efficient** way of data movement!

Use burst-based **latency-tolerant** transfers + double buffering
There is one remaining issue in Occamy...

- Occamy Interconnect
  - AXI
  - Crossbar-based
  - Deep Memory Hierarchy
    - Cluster, Quadrant, Top-level
  - Multi-hop

...It didn’t scale well

Top-level interconnect is huge*

*After optimization and splitting it up!
AXI has some problems

**Scalability**
- Tracking outstanding transactions based on ID
- ID width increase after each hop
- Exponential ID Scaling

**Performance**
- Strict ordering requirements on ID
- Stall to guarantee ordering

**Overhead**
- Logic for tracking outstanding requests
- Included in every switch
- ID Remappers to reduce ID width

AXI does not scale to deep many-hop memory hierarchies

How to solve the scalability issue?
With a real NoC...

...how it was proposed 21 years ago...

...FlooNoC was born
Decouple AXI with link-level protocol

**Routers**
- Custom Link-level protocol, agnostic of AXI
- “State-less“
- **Simple, low complexity** router microarchitecture
- **No tracking** of any transactions
- Better **scalability** to multi-hop networks

**Network Interface**
- Converts AXI to link-level protocol
- Supports all nice AXI features
  - **Burst** transfers
  - **Outstanding transactions**
  - **Atomics**
- **Reorder Buffers** to guarantee ordering
  - Move complexity to endpoints
  - Amortized with simpler routers
The traditional approach: Serialization
The traditional approach: Serialization

Initiator

Narrow Link

Target

10 Years Of PULP, Lugano June 2023
The traditional approach: Serialization
Avoid serialization with **wide** links
Send everything in **parallel**

Link frequency needs to increase frequency to match bandwidth of initiator

Routing resources are **plentiful** in modern technologies

Not possible anymore if bandwidth injection is too high
How to handle heterogeneous traffic

**High Bandwidth traffic**
- DMA traffic
- Wide busses (512-bit)
- Burst-based
- Large messages (few kBs)
- Latency tolerant

**Latency-sensitive traffic**
- Core traffic
- Atomics
- Synchronization
- Single-word messages

**Auxiliary traffic**
- Request messages
- Acknowledgment messages (AXI write responses)
- Small messages

We need **wide links** with **high utilization**
We need **Narrow links** with **low utilization**
Low-priority traffic, **minimize interference**

Narrow Req
Wide
Narrow Rsp
Case Study: Snitch Cluster

- **Occamy Snitch Cluster**
  - 9 RISC-V + DMA
  - **128KB** TCDM
  - **Two** bidir AXI interfaces
    - **Narrow** 64-bit
    - **Wide** 512-bit
Case Study: Snitch Cluster

- **Occamy Snitch Cluster**
  - 9 RISC-V + DMA
  - 128KB TCDM
  - Two bidir AXI interfaces
    - Narrow 64-bit
    - Wide 512-bit

- **Compute Tile**
  - Wraps a Snitch Cluster
  - Network Interface with 8KB Reorder Buffer
  - 5x5 Multi-link Router
Case Study: Snitch Cluster

• Occamy Snitch Cluster
  • 9 RISC-V + DMA
  • 128KB TCDM
  • Two bidir AXI interfaces
    • Narrow 64-bit
    • Wide 512-bit

• Compute Tile
  • Wraps a Snitch Cluster
  • Network Interface with 8KB Reorder Buffer
  • 5x5 Multi-link Router
  • Can be arranged in a mesh
Case Study: is physical implementation feasible?

- Hard Router Macro
- Flattened Network Interface + Reorder Buffer (SRAM)
Case Study: is physical implementation feasible?

- Hard Router Macro
- Flattened Network Interface + Reorder Buffer (SRAM)
- Tiled design methodology
  - N/E/S/W ports
  - Tiles can be directly abutted
  - Mesh of compute tiles

Only 10% area overhead with an integrated NoC
How to route 1000s of wires

- **Routing Channels**
  - #wires: ~1600
  - Distance: 800-900 μm

- **Methodology**
  - Use upper layers
    - Global routing
    - Not blocked by SRAMS
  - 4 metal layers suffice with min. pitch

**Area-efficient** routing over memory and logic
How many cuts do we need to meet timing?

• Router have input/output buffers…
• … But timing is poor
• High output load ➞ very high transition times
• Buffers are the solution, of course…
• … But SRAMs obstruct the lower layers
• Manual placement of Buffer Islands

Routing channels are quite long…

No cuts needed to meet timing!
Low-Latency & High-Bandwidth

Latency

- **18 cycles** access latency to neighboring tile
- Round-trip time
- **2-cycle** hop latency
- Even for large distances!

Bandwidth

- High Bandwidth thanks to **wide** links
  - **629Gpbs** per link
  - ... at a modest **frequency**
  - **1.23 GHz** (TT, 0.8V, 25°)

Effective for **latency-sensitive** traffic
  e.g. many-core synchronization

Very efficient for **high-bandwidth & latency-tolerant** DMA transfers
Energy-efficiency is key for data-movement

- The NoC consumes only 7% of the power
- Data movement is very cheap with only 0.19 pJ/byte/hop

The NoC will not become the energy bottleneck which is key for scalability to large systems.
We’ve only just begun...

• **Still Work in Progress**
  - Set of IPs are ready
  - System-level verification to come

• **AXI-Pack**
  - Sparse workloads on an NoC
We’ve only just begun...

- **Still Work in Progress**
  - Set of IPs are ready
  - System-level verification to come

- **Chi’s talk: AXI-Pack**
  - Sparse workloads on an NoC

- **Scale up**
  - How does it compare to Occamy?

- **Network-on-Package (NoP)**
  - Fuse with off-chip Serial Link
Scalability is key for data movement

- **Efficient data movement** is a necessity in High-Performance systems
- **Bandwidth** requirements are always growing
- **Off-chip** communication have 10x higher latency

- **AXI** does **not scale well** for deep memory hierarchy
  - A NoC scales much better
- Modern technologies enable **wide** links can deliver **high-bandwidth** ...
- ...separate physical channels for different traffic provides **low-latency** communication

The iDMA is **highly scalable** for **high-bandwidth** & **latency-tolerant** use cases with **minimal overhead**

FlooNoC demonstrates the feasibility and **scalability** of **wide links** while consuming only a **fraction** of the **power**
Where Can I Get It?

- **iDMA on Github**
  - Currently v0.4
  - PR pending for v0.5: Multiprotocol
  - DMA Paper on arXiv

- **FlooNoC**
  - Still in the development phase
  - Focus on verification
  - Paper on arXiv
  - Soon on GitHub!

- **Related: AXI paper (IEEE TC)**
PULP Platform
Open Source Hardware, the way it should be!

Tim Fischer  fischeti@iis.ee.ethz.ch
Thomas Benz  tbenz@iis.ee.ethz.ch

Institut für Integrierte Systeme – ETH Zürich
Gloriastrasse 35
Zürich, Switzerland

DEI – Università di Bologna
Viale del Risorgimento 2
Bologna, Italy