General in-network processing – time is ripe!
Keynote at the High-Performance Interconnects Forum with HPC China 2020
The Development of High-Performance Networking Interfaces

- Scalable Coherent Interface
- Myrinet GM+MX
- Virtual Interface Architecture
- OFED
- libfabric
- Ethernet+TCP/IP
- Fast Messages
- Quadrics QsNet
- Cray Gemini
- IB Verbs
- Portals 4

1980: Scalable Coherent Interface
1990: Myrinet GM+MX
2000: Virtual Interface Architecture
2010: OFED
2020: libfabric

- sockets
- (active) message based
- protocol offload
- remote direct memory access (RDMA)
- coherent memory access
- OS bypass
- zero copy
- triggered operations

- ARM cores (with full OS, outside packet pipe)
- Flow Processors (limited flexibility, P4)
- Microsoft Catapult
- Xilinx
- BAREFOOT NETWORKS
- FPGAs (limited productivity, silicon efficiency)

Smart NIC

- Catapult
- FPGAs (limited productivity, silicon efficiency)
Data Processing in modern RDMA networks

Remote Nodes (via network)

Mellanox Connect-X5: 1 packet/5ns
Tomorrow (400G): 1 packet/1.2ns

RDMA NIC

arriving packets

Local Node

Core i7 Haswell

Regs
4 cycles ~1.3ns

11 cycles ~ 3.6ns

34 cycles ~11.3ns

L1

L2

L3

Main Memory

125 cycles ~41.6ns

RDMA Processing

DMA Unit

明天 (400G): 1 packet/1.2ns
The future of High-Performance Networking Interfaces

Established Principles for Compute Acceleration

- OpenGL
- DirectX11
- OpenMP 4.0
- NVIDIA CUDA
- OpenCL

Generalization
Revolutionizes Acceleration

Where do we stand in Network Acceleration?

- eBPF
- Data Acceleration

sPIN NIC – Architecture for fast Network Processing

Matching Entry (ME):
- Host Mem. Address
- Matching Bits
- sPIN Handlers (optional)

sPIN NIC - Abstract Machine Model for Packet Processing

Packet Scheduler

arriving packets

Fast shared memory
(packet input buffer)

HPU 0  HPU 1
HPU 2  HPU 3

DMA Unit

upload
handlers
manage
memory

CPU

MEM

R/W

RDMA vs. sPIN in action: Simple Ping Pong

RDMA vs. sPIN in action: Streaming Ping Pong

sPIN – Programming Interface

**Header handler**

```c
__handler int pp_header_handler(const ptl_header_t h, void *state) {
    pingpong_info_t *i = state;
    i->source = h.source_id;
    return PROCESS_DATA; // execute payload handler to put from device
}
```

**Payload handler**

```c
__handler int pp_payload_handler(const ptl_payload_t p, void * state) {
    pingpong_info_t *i = state;
    PtlHandlerPutFromDevice(p.base, p.length, 1, 0, i->source, 10, 0, NULL, 0);
    return SUCCESS;
}
```

**Completion handler**

```c
__handler int pp_completion_handler(int dropped_bytes, bool flow_control_triggered, void *state) {
    return SUCCESS;
}
```

```c
connect(peer, /* ... */ , &pp_header_handler, &pp_payload_handler, &pp_completion_handler);
```
Talk roadmap

Motivation and Overview → Data Layout Transformation → Hardware Implementation

further use cases

Network Group Communication
Distributed Data Management
Application domain

Memory layout

Structured Exchange

Reshaping

Unstructured Exchange


https://specfem3d.readthedocs.io/en/latest/

http://fourier.eng.hmc.edu/e161/lectures/fourier/node10.html
Programming Support for Non-Contiguous Transfers

ARMCI
CAF
Chapel
Portals 4

SHMEM
UPC
X10

I/O Vectors
Strided transfers
Compiler-Assisted Aggregation
Support for multiple strides (e.g., 3D faces)

MPI
Derived Datatypes

vector
indexed
struct

State of the Art in MPI Datatypes Processing

Gropp, W., et al., March. Improving the performance of MPI derived datatypes. *MPIDC’99*
Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. 2017. sPIN: High-performance streaming Processing In the Network. *SC’17*
State of the Art in MPI Datatypes Processing

Run **user-defined packet-processing kernels on the NIC**

- 1 K non-contig. regions.
- 256 non-contig. regions.
- 4 MiB message; stride = 2 x block_size

Gropp, W., et al., March. Improving the performance of MPI derived datatypes. *MPIDC'99*
A simple vectorize scatter datatype

```c
vector_payload_handler(
    handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t*) args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;
    uint8_t *pkt_payload = args->pkt_payload_ptr;
    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (args->pkt_offset / ddt_descr->block_size) * stride;
    uint8_t *host_address = host_base_ptr + host_offset;

    for (uint32_t i = 0; i < num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }
    return SPIN_SUCCESS;
}
```

Can we define a general handler to process arbitrary datatypes?
Porting the MPI Types Library [1] to sPIN

MPI Types Library on sPIN: Read-Write Checkpoints

NIC Memory

V-HPUs:
- V-HPU 0
  - Index:
  - Vector:
- V-HPU 1
  - Index:
  - Vector:
- V-HPU 2
  - Index:
  - Vector:
- V-HPU 3
  - Index:
  - Vector:
- V-HPU 4
  - Index:
  - Vector:
- V-HPU 5
  - Index:
  - Vector:

Index (blocks: 2, blocklen: 1, offsets: {0, x}, basetype: *)

Vector (blocks: 3, blocklen: 2, stride: 3, basetype: )

Δt = 2

HPU Memory

HPU 0
HPU 1
HPU 2
HPU 3

Packet Scheduler

Index

Vector

Index

Vector

Packet Scheduler

Line rate

Specialized

RW Checkpoints

HPU-Local

RO Checkpoints

Host Unpack

Checkpoint Interval Selection

**Network**

- **HPU 0**
- **HPU 1**
- **HPU 2**

\[ T_C = T_{pkt} + \left[ \frac{\Delta r}{k} \right] \cdot (P - 1) \cdot T_{pkt} + \left[ \frac{n_{pkt}}{P} \right] \cdot T_{PH}(\gamma) \]

1. **Limit the impact of the scheduling overhead**
2. **Do not saturate NIC memory with checkpoints**
3. **Do not saturate the packet buffer**

Cray Slingshot Simulator

32 Cortex A15 @800 MHz, single-cycle access memory

Real Application DDTs

Real Applications DDTs

Checkpoints Overhead

Data Movement

75% of the analyzed DDTs amortized after 4 reuses

Handler Analysis

Full app speedup (FFT2D)

Up to 3.8x less moved data volume

PsPIN hardware implementation: sPIN on PULP
Circuit Complexity and Power Estimations

- Processor synthesized in GlobalFoundries 22nm fully depleted silicon on insulator (FDSOI) technology
  - Timing: 1 GHz

- Accelerator complexity: ~95 MGE
  - 18.5 mm² area (assuming layout density 85%)
  - Mellanox BlueField: 16 A72 64bit cores
    Estimated area: 51 mm²

- Power consumption (100% toggle rate): 6 W (not including I/O and PHY power).
Why choosing PULP for sPIN?

Architectures:
- **zynq**: ARM Cortex-A53, 64-bit, 2-way superscalar, 1.2 GHz
- **ault**: Intel Skylake Gold 6154, 64-bit, out-of-order execution, 3 GHz
- **PsPIN/RISCY**: RISC-V based, 32-bit, in-order, 1 GHz

Use cases:
- Data reduction
- Single message aggregation
- Packet filtering/rewriting
- KV store cache
- Strided datatypes
- Histogram

<table>
<thead>
<tr>
<th>Arch.</th>
<th>Tech.</th>
<th>Die area</th>
<th>PEs</th>
<th>Memory</th>
<th>Area/PE</th>
<th>Area/PE (scaled)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ault</td>
<td>14 nm</td>
<td>485 mm²</td>
<td>18</td>
<td>43.3 MiB</td>
<td>17.978 mm²</td>
<td>35.956 mm²</td>
</tr>
<tr>
<td>zynq</td>
<td>16 nm</td>
<td>3.27 mm²</td>
<td>4</td>
<td>1.125 MiB</td>
<td>0.876 mm²</td>
<td>1.752 mm²</td>
</tr>
<tr>
<td>PsPIN</td>
<td>22 nm</td>
<td>18.5 mm²</td>
<td>32</td>
<td>12 MiB</td>
<td>0.578 mm²</td>
<td>0.578 mm²</td>
</tr>
</tbody>
</table>

Actual throughput on PsPIN:
Illustrating broadcast acceleration with sPIN

Illustrating broadcast acceleration with sPIN

Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. *HOTI’11*

Illustrating broadcast acceleration with sPIN

Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. *HOTI*’11

Further results and use-cases
Further results and use-cases

### Use Case 4: MPI Rendezvous Protocol

<table>
<thead>
<tr>
<th>program</th>
<th>p</th>
<th>msgs</th>
<th>ovhd</th>
<th>ovhd</th>
<th>red</th>
</tr>
</thead>
<tbody>
<tr>
<td>MILC</td>
<td>64</td>
<td>5.7M</td>
<td>5.5%</td>
<td>1.9%</td>
<td>65%</td>
</tr>
<tr>
<td>POP</td>
<td>64</td>
<td>772M</td>
<td>3.1%</td>
<td>2.4%</td>
<td>22%</td>
</tr>
<tr>
<td>coMD</td>
<td>72</td>
<td>5.3M</td>
<td>6.1%</td>
<td>2.4%</td>
<td>60%</td>
</tr>
<tr>
<td>coMD</td>
<td>360</td>
<td>28.1M</td>
<td>5.5%</td>
<td>2.8%</td>
<td>58%</td>
</tr>
<tr>
<td>Cloverleaf</td>
<td>72</td>
<td>2.7M</td>
<td>5.2%</td>
<td>2.4%</td>
<td>53%</td>
</tr>
<tr>
<td>Cloverleaf</td>
<td>360</td>
<td>15.3M</td>
<td>5.6%</td>
<td>3.2%</td>
<td>42%</td>
</tr>
</tbody>
</table>
Further results and use-cases

### Use Case 4: MPI Rendezvous Protocol

<table>
<thead>
<tr>
<th>program</th>
<th>p</th>
<th>msgs</th>
<th>ovhd</th>
<th>ovhd</th>
<th>red</th>
</tr>
</thead>
<tbody>
<tr>
<td>MILC</td>
<td>64</td>
<td>5.7M</td>
<td>5.5%</td>
<td>1.9%</td>
<td>65%</td>
</tr>
<tr>
<td>POP</td>
<td>64</td>
<td>772M</td>
<td>3.1%</td>
<td>2.4%</td>
<td>22%</td>
</tr>
<tr>
<td>coMD</td>
<td>72</td>
<td>5.3M</td>
<td>6.1%</td>
<td>2.4%</td>
<td>60%</td>
</tr>
<tr>
<td>coMD</td>
<td>360</td>
<td>28.1M</td>
<td>6.5%</td>
<td>2.8%</td>
<td>58%</td>
</tr>
<tr>
<td>Cloverleaf</td>
<td>72</td>
<td>2.7M</td>
<td>5.2%</td>
<td>2.4%</td>
<td>53%</td>
</tr>
<tr>
<td>Cloverleaf</td>
<td>360</td>
<td>15.3M</td>
<td>5.6%</td>
<td>3.2%</td>
<td>42%</td>
</tr>
</tbody>
</table>

### Use Case 5: Distributed KV Store

41% lower latency

Further results and use-cases

### Use Case 4: MPI Rendezvous Protocol

<table>
<thead>
<tr>
<th>program</th>
<th>p</th>
<th>msgs</th>
<th>ovhd</th>
<th>ovhd</th>
<th>red</th>
</tr>
</thead>
<tbody>
<tr>
<td>MILC</td>
<td>64</td>
<td>5.7M</td>
<td>5.5%</td>
<td>1.9%</td>
<td>65%</td>
</tr>
<tr>
<td>POP</td>
<td>64</td>
<td>772M</td>
<td>3.1%</td>
<td>2.4%</td>
<td>22%</td>
</tr>
<tr>
<td>coMD</td>
<td>72</td>
<td>5.3M</td>
<td>6.1%</td>
<td>2.4%</td>
<td>60%</td>
</tr>
<tr>
<td>coMD</td>
<td>360</td>
<td>28.1M</td>
<td>6.5%</td>
<td>2.8%</td>
<td>58%</td>
</tr>
<tr>
<td>Cloverleaf</td>
<td>72</td>
<td>2.7M</td>
<td>5.2%</td>
<td>2.4%</td>
<td>53%</td>
</tr>
<tr>
<td>Cloverleaf</td>
<td>360</td>
<td>15.3M</td>
<td>5.6%</td>
<td>3.2%</td>
<td>42%</td>
</tr>
</tbody>
</table>

### Use Case 5: Distributed KV Store

41% lower latency


### Use Case 6: Conditional Read

Further results and use-cases

**Use Case 4: MPI Rendezvous Protocol**

<table>
<thead>
<tr>
<th>Program</th>
<th>p</th>
<th>msgs</th>
<th>ovhd</th>
<th>ovhd</th>
<th>red</th>
</tr>
</thead>
<tbody>
<tr>
<td>MILC</td>
<td>64</td>
<td>5.7M</td>
<td>5.5%</td>
<td>1.9%</td>
<td>65%</td>
</tr>
<tr>
<td>POP</td>
<td>64</td>
<td>772M</td>
<td>3.1%</td>
<td>2.4%</td>
<td>22%</td>
</tr>
<tr>
<td>coMD</td>
<td>72</td>
<td>5.3M</td>
<td>6.1%</td>
<td>2.4%</td>
<td>60%</td>
</tr>
<tr>
<td>coMD</td>
<td>360</td>
<td>28.1M</td>
<td>6.5%</td>
<td>2.8%</td>
<td>58%</td>
</tr>
<tr>
<td>Cloverleaf</td>
<td>72</td>
<td>2.7M</td>
<td>5.2%</td>
<td>2.4%</td>
<td>53%</td>
</tr>
<tr>
<td>Cloverleaf</td>
<td>360</td>
<td>15.3M</td>
<td>5.6%</td>
<td>3.2%</td>
<td>42%</td>
</tr>
</tbody>
</table>

**Use Case 5: Distributed KV Store**

41% lower latency


**Use Case 6: Conditional Read**


**Use Case 7: Distributed Transactions**

Dragojević, A, et al., No compromises: distributed transactions with consistency, availability, and performance. *SOSP’15*
Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

<table>
<thead>
<tr>
<th>program</th>
<th>p</th>
<th>msgs</th>
<th>ovhd</th>
<th>ovhd red</th>
</tr>
</thead>
<tbody>
<tr>
<td>MILC</td>
<td>64</td>
<td>5.7M</td>
<td>5.5%</td>
<td>1.9% 65%</td>
</tr>
<tr>
<td>POP</td>
<td>64</td>
<td>772M</td>
<td>3.1%</td>
<td>2.4% 22%</td>
</tr>
<tr>
<td>coMD</td>
<td>72</td>
<td>5.3M</td>
<td>6.1%</td>
<td>2.4% 60%</td>
</tr>
<tr>
<td>coMD</td>
<td>360</td>
<td>28.1M</td>
<td>6.5%</td>
<td>2.8% 58%</td>
</tr>
<tr>
<td>Cloverleaf</td>
<td>72</td>
<td>2.7M</td>
<td>5.2%</td>
<td>2.4% 53%</td>
</tr>
<tr>
<td>Cloverleaf</td>
<td>360</td>
<td>15.3M</td>
<td>5.6%</td>
<td>3.2% 42%</td>
</tr>
</tbody>
</table>

Use Case 5: Distributed KV Store

41% lower latency


Use Case 6: Conditional Read


Use Case 7: Distributed Transactions

Dragojević, A., et al., No compromises: distributed transactions with consistency, availability, and performance. SOSP’15

Use Case 8: FT Broadcast

Bosilca, G., et al., Failure Detection and Propagation in HPC systems. SC’16
Further results and use-cases

<table>
<thead>
<tr>
<th>Use Case 4: MPI Rendezvous Protocol</th>
<th>Use Case 5: Distributed KV Store</th>
<th>Use Case 6: Conditional Read</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>program</strong></td>
<td><strong>p</strong></td>
<td><strong>msgs</strong></td>
</tr>
<tr>
<td>MILC</td>
<td>64</td>
<td>5.7M</td>
</tr>
<tr>
<td>POP</td>
<td>64</td>
<td>772M</td>
</tr>
<tr>
<td>coMD</td>
<td>72</td>
<td>5.3M</td>
</tr>
<tr>
<td>coMD</td>
<td>360</td>
<td>28.1M</td>
</tr>
<tr>
<td>Cloverleaf</td>
<td>72</td>
<td>2.7M</td>
</tr>
<tr>
<td>Cloverleaf</td>
<td>360</td>
<td>15.3M</td>
</tr>
</tbody>
</table>

**The Next 700 sPIN use-cases**

... just think about sPIN graph kernels ....

41% lower latency


Use Case 7: Distributed Transactions

Dragojević, A., et al., No compromises: distributed transactions with consistency, availability, and performance. SOSP’15

Use Case 8: FT Broadcast

Bosilca, G., et al., Failure Detection and Propagation in HPC systems. SC’16

Use Case 9: Distributed Consensus

István, Z., et al., Consensus in a Box: Inexpensive Coordination in Hardware. NSDI’16
Next step - pushing sPIN into network switches?

- Needs to be carefully vetted!
  - Can we achieve our goals with P4?

- What else needs to be fixed before we go into the network?

- We chose to investigate network noise first
  - Not enough time here but let me give you a brief overview.
Network noise analysis and mitigation

Analysis of the impact of adaptive routing on network noise

Design and implementation of a transparent solution

Improvements up to 55% on real applications

Daniele De Sensi et al.: “Mitigating Network Noise on Dragonfly Networks through Application-Aware Routing”, IEEE/ACM SC19
Slingshot the Exascale Interconnect

Description of the main features of the interconnect

In-depth benchmarking procedure that can be ported to other interconnect

Detailed results on performance, congestion control, and quality of service, on microbenchmarks, HPC, and DC applications

Daniele De Sensi et al.: “An In-Depth Analysis of the Slingshot Interconnect”, IEEE/ACM SC20
SPCL is hiring PhD students and highly-qualified postdocs to reach new heights!

https://spcl.inf.ethz.ch/Jobs/
sPIN Streaming Processing in the Network for Network Acceleration

Full specification: https://arxiv.org/abs/1709.05483

Try it out: https://spcl.inf.ethz.ch/Research/Parallel_Programming/sPIN/
Backup Slides
But why PULP/RISC-V?

- RISC-V is an open source ISA
  - Allows and supports extensions
    - *Doing this in ARM may be complex and expensive*

- PULP aims to provide high performance per Watt
  - Energy efficient
  - Provides tight control over compute and data movement schedule
  - Fits well the sPIN abstract machine model (e.g., removing cache coherency on ARM could be painful)
  - PULP is actively researched + we can leverage ISS group expertise at ETH