sPIN: High-performance streaming Processing in the Network

S. Di Girolamo, D. De Sensi, T. Hoefler, and the sPIN team @SPCL
Workloads

Systems
Workloads

Systems
Workloads

Systems
Workloads

Systems
Low latency, high throughput

Workloads

Systems
Data Processing in modern RDMA networks

Remote Nodes (via network)

Local Node
Core i7 Haswell

Regs
4 cycles ~1.3ns
L1
11 cycles ~ 3.6ns
L2
34 cycles ~11.3ns
L3
125 cycles ~41.6ns

Main Memory
Input buffer

DMA Unit
PCIe bus
~ 250ns

arriving packets
RDMA NIC
RDMA Processing

34 cycles ~11.3ns
11 cycles ~ 3.6ns
4 cycles ~1.3ns
Data Processing in modern RDMA networks

Remote Nodes (via network)

Local Node

Core i7 Haswell

Regs
4 cycles ~1.3ns
L1
11 cycles ~ 3.6ns
L2
34 cycles ~11.3ns
L3

Input buffer

PCIe bus
~ 250ns

arriving packets

RDMA Processing

DMA Unit

RDMA NIC

125 cycles ~41.6ns

Main Memory

arriving packets

RDMA Processing

DMA Unit

RDMA NIC
Data Processing in modern RDMA networks

Remote Nodes (via network)

Local Node

Core i7 Haswell

Regs
4 cycles ~1.3ns
L1
11 cycles ~ 3.6ns
L2
34 cycles ~11.3ns
L3
34 cycles ~11.3ns

Remote Nodes (via network)

Local Node

Core i7 Haswell

Regs
4 cycles ~1.3ns
L1
11 cycles ~ 3.6ns
L2
34 cycles ~11.3ns
L3
34 cycles ~11.3ns

PCIe bus
~ 250ns

arriving packets

RDMA NIC

DMA Unit

Input buffer

Main Memory

~ 250ns

~ 41.6ns

34 cycles ~11.3ns

11 cycles ~ 3.6ns

4 cycles ~1.3ns

~ 250ns

~ 250ns
Data Processing in modern RDMA networks

Remote Nodes (via network)

RDMA Processing
DMA Unit

arriving packets

RDMA NIC

PCle bus
~ 250ns

Local Node
Core i7 Haswell

Regs
4 cycles ~ 1.3ns
L1
11 cycles ~ 3.6ns
L2
34 cycles ~ 11.3ns
L3
125 cycles ~ 41.6ns

Main Memory

Input buffer

arriving packets

34 cycles ~ 11.3ns
11 cycles ~ 3.6ns
4 cycles ~ 1.3ns

~ 250ns
Data Processing in modern RDMA networks

Mellanox Connect-X5: 1 packet/5ns
Mellanox Connect-X7 (400G): 1 packet/1.2ns
Data Processing in modern RDMA networks

Mellanox Connect-X5: 1 packet/5ns
Mellanox Connect-X7 (400G): 1 packet/1.2ns
Data Processing in modern RDMA networks

Mellanox Connect-X5: **1 packet/5ns**
Mellanox Connect-X7 (400G): **1 packet/1.2ns**
Data Processing in modern RDMA networks

Remote Nodes (via network)

Mellanox Connect-X5: 1 packet/5ns
Mellanox Connect-X7 (400G): 1 packet/1.2ns

Local Node
Core i7 Haswell

RDMA NIC

PCIe bus

Main Memory

Input buffer

arriving packets

34 cycles ~ 11.3ns
11 cycles ~ 3.6ns
4 cycles ~ 1.3ns

125 cycles ~ 41.6ns
34 cycles ~ 11.3ns
11 cycles ~ 3.6ns
4 cycles ~ 1.3ns

~ 250ns
Data Processing in modern RDMA networks

Mellanox Connect-X5: 1 packet/5ns
Mellanox Connect-X7 (400G): 1 packet/1.2ns
sPIN: High-performance streaming Processing in the Network

The programming model

sPIN NIC - Abstract Machine Model

The hardware accelerator

Architectural principles for in-network compute

Use cases
sPIN NIC - Abstract Machine Model
sPIN NIC - Abstract Machine Model
sPIN NIC - Abstract Machine Model

Fast shared memory (packet input buffer)

sPIN NIC - Abstract Machine Model

Packet Scheduler

arriving packets

Fast shared memory (packet input buffer)
sPIN NIC - Abstract Machine Model

Packet Scheduler

arriving packets

Fast shared memory (packet input buffer)

HPU 0  HPU 1

HPU 2  HPU 3
sPIN NIC - Abstract Machine Model

Fast shared memory (packet input buffer)

- HPU 0
- HPU 1
- HPU 2
- HPU 3

DMA Unit

arriving packets

sPIN NIC - Abstract Machine Model

- Arriving packets are scheduled by the Packet Scheduler.
- Fast shared memory (packet input buffer) is accessed by HPU 0, HPU 1, HPU 2, and HPU 3.
- DMA Unit facilitates data transfer between the shared memory and MEM (Memory).

sPIN NIC - Abstract Machine Model

- Packet Scheduler
- Fast shared memory (packet input buffer)
- HPU 0
- HPU 1
- HPU 2
- HPU 3
- DMA Unit

arriving packets

R/W

CPU

MEM

sPIN NIC - Abstract Machine Model

Packet Scheduler

arriving packets

Fast shared memory
(packet input buffer)

HPU 0  HPU 1
HPU 2  HPU 3

DMA Unit

upload handlers

CPU

MEM

R/W

sPIN NIC - Abstract Machine Model

- **Packet Scheduler**
- **Fast shared memory** (packet input buffer)
- **DMA Unit**
- **upload**
- **handlers**
- **manage**
- **memory**
- **CPU**
- **MEM**

- **arriving packets**
- **HPU 0**
- **HPU 1**
- **HPU 2**
- **HPU 3**

- **R/W**
sPIN NIC - Abstract Machine Model

Fast shared memory (packet input buffer)

- HPU 0
- HPU 1
- HPU 2
- HPU 3

Packet Scheduler

arriving packets

upload

handlers

manage

memory

CPU

MEM

DMA Unit

R/W
sPIN NIC - Abstract Machine Model

- **Packet Scheduler**: Fast shared memory (packet input buffer)
  - HPU 0
  - HPU 1
  - HPU 2
  - HPU 3

- **DMA Unit**: upload, handlers, manage, memory

- **CPU**:

- **MEM**: R/W

Arriving packets

sPIN NIC - Abstract Machine Model

arriving packets
Packet Scheduler

Fast shared memory (packet input buffer)

HPU 0
HPU 1
HPU 2
HPU 3

DMA Unit

upload
handlers
manage
memory

CPU

MEM

R/W

sPIN NIC - Abstract Machine Model

Packet Scheduler

Fast shared memory (packet input buffer)

HPU 0  HPU 1
HPU 2  HPU 3

DMA Unit

upload
handlers
manage
memory

R/W

CPU

MEM

arriving packets

sPIN – Programming Interface

Incoming message/flow

sPIN – Programming Interface

Incoming message/flow

Tail

Payload

Header

sPIN – Programming Interface

sPIN – Programming Interface

Header handler

```c
int hh(const handler_args_t * args){
    return 0;
}
```

sPIN – Programming Interface

**Header handler**

```c
int hh(const handler_args_t * args){
    return 0;
}
```

**Payload handler**

```c
int ph (const handler_args_t * args){
    spin_pkt_send(args->pkt_pld, 0, args->source, args->pkt_offset,
                  args->rlength, args->match_bits, args->pkt_pld_len);
    return 0;
}
```
sPIN – Programming Interface

**Header handler**

```c
int hh (const handler_args_t * args) {
    return 0;
}
```

**Payload handler**

```c
int ph (const handler_args_t * args) {
    spin_pkt_send (args->pkt_pld, 0, args->source, args->pkt_offset,
                   args->rlength, args->match_bits, args->pkt_pld_len);
    return 0;
}
```

**Completion handler**

```c
int th (const handler_args_t * args) {
    return 0;
}
```
RDMA vs. sPIN in action: Streaming Ping Pong

RDMA vs. sPIN in action: Streaming Ping Pong

RDMA vs. sPIN in action: Streaming Ping Pong

RDMA vs. sPIN in action: Streaming Ping Pong

Architectural principles for in-network compute

- Low latency, full throughput
- Support for wide range of use cases
- Easy to integrate

Architectural principles for in-network compute
Architectural principles for in-network compute

Low latency, full throughput
Architectural principles for in-network compute

- Low latency, full throughput
- Highly parallel
Architectural principles for in-network compute

Low latency, full throughput

Highly parallel

![Graph showing the relationship between Handler duration (us) and # HPUs to line rate for 200 Gbit/s and 400 Gbit/s.]
Architectural principles for in-network compute

- Low latency, full throughput
- Highly parallel
- Fast scheduling

Graph showing the relationship between handler duration (us) and the number of HPUs to line rate, with 400 Gbit/s and 200 Gbit/s lines.
Architectural principles for in-network compute

- Low latency, full throughput
- Highly parallel
- Fast scheduling
- Fast explicit memory access

Graph showing the relationship between Handler duration (us) and the number of HPUs to line rate, with a line for 400 Gbit/s and another for 200 Gbit/s.
Architectural principles for in-network compute
Architectural principles for in-network compute

Support for wide range of use cases
Architectural principles for in-network compute

Support for wide range of use cases

Network-accelerated datatypes

sPIN-FS

Zoo-sPINNER consensus protocols

Quantization

Allreduce and other collectives

Packet classification and pattern matching

Erasure coding

Serverless
Architectural principles for in-network compute

Support for wide range of use cases

- Stateful computation support
- Handlers isolation

- Network-accelerated datatypes
- sPIN-FS
- Zoo-sPINNER consensus protocols
- Quantization
  Allreduce and other collectives
- Packet classification and pattern matching
- Serverless
- Erasure coding
Architectural principles for in-network compute
Architectural principles for in-network compute

Easy to integrate
Architectural principles for in-network compute

Easy to integrate
Architectural principles for in-network compute

- Easy to integrate
- Area and power efficiency
- Configurability
PsPIN: A PULP-powered implementation of sPIN

![Diagram of PsPIN architecture]

PsPIN: A PULP-powered implementation of sPIN

PsPIN: A PULP-powered implementation of sPIN

PsPIN: A PULP-powered implementation of sPIN

PsPIN: A PULP-powered implementation of sPIN

PsPIN: A PULP-powered implementation of sPIN

Network Interface
- Outbound Engine
- Inbound Engine
- PsPIN Unit

Host Interface
- Command Unit

Packet Scheduler
- DMA engine (off-cluster)

Command unit

Monitoring & control

Cluster 0
- Scratchpad
- DMA
- CSCHED

Cluster 1
- Scratchpad
- DMA
- CSCHED

Cluster 2
- Scratchpad
- DMA
- CSCHED

Cluster 3
- Scratchpad
- DMA
- CSCHED

Packet buffer

Program memory

Handler memory

H cv32e40p (aka RI5CY): RISC-V, 4 stage pipeline, in-order, 32 bit (40 kGE)

Application perspective

- Network Interface
  - Outbound Engine
  - Inbound Engine
  - Command Unit
  - PsPIN Unit

- Host Interface

- Packet Scheduler
  - DMA engine (off-cluster)
  - Command unit
  - Monitoring & control

- Cluster 0
  - DMA
  - CSCHED
  - Scratchpad

- Cluster 1
  - DMA
  - CSCHED
  - Scratchpad

- Cluster 2
  - DMA
  - CSCHED
  - Scratchpad

- Cluster 3
  - DMA
  - CSCHED
  - Scratchpad

- Program memory
- Handler memory
- Packet buffer
Application perspective

1 Define and offload handlers

**Telemetry:**
- telemetry\_hh(), telemetry\_ph(), telemetry\_th();

**Filtering:**
- filter\_hh(), filter\_ph, filter\_th();
Application perspective

1. Define and offload handlers

   Telemetry:
   telemetry_hh(), telemetry_ph(), telemetry_th();

   Filtering:
   filter_hh(), filter_ph, filter_th();

2. Define an execution context

   Execution context: EC_filter
   filter_hh(), filter_ph(), filter_th();
   NIC memory: **STATE**
   Host buffer: **BUF**
Application perspective

1. Define and offload handlers

   - Telemetry:
     
     telemetry_hh(), telemetry_ph(), telemetry_th();
   
   - Filtering:
     
     filter_hh(), filter_ph, filter_th();

2. Define an execution context

   **Execution context: EC_filter**

   - filter_hh(), filter_ph(), filter_th();
   - NIC memory: **STATE**
   - Host buffer: **BUF**
Application perspective

1. Define and offload handlers

   Telemetry:
   telemetry_hh(), telemetry_ph(), telemetry_th();

   Filtering:
   filter_hh(), filter_ph, filter_th();

2. Define an execution context

   Execution context: EC_filter
   filter_hh(), filter_ph(), filter_th();
   NIC memory: STATE
   Host buffer: BUF

   Packet buffer
   Program memory
   Handler memory
   STATE
Application perspective

1. Define and offload handlers
   - Telemetry: telemetry_hh(), telemetry_ph(), telemetry_th();
   - Filtering: filter_hh(), filter_ph, filter_th();

2. Define an execution context
   - Execution context: EC_filter
     - filter_hh(), filter_ph(), filter_th();
     - NIC memory: STATE
     - Host buffer: BUF

3. Define matching rule:
   - e.g., (IP packets) -> EC_filter
Network perspective

Execution context: EC_filter

filter_hh(), filter_ph(), filter_th();
NIC memory: STATE
Host buffer: BUF
Network perspective

1. Match packet to execution context e.g., (IP packets) -> EC_filter

**Execution context: EC_filter**

- `filter_hh()`, `filter_ph()`, `filter_th()`;
- NIC memory: **STATE**;
- Host buffer: **BUF**;

**Network perspective**

1. Match packet to execution context e.g., (IP packets) -> EC_filter
Network perspective

1. Match packet to execution context e.g., (IP packets) -> EC_filter

2. Write **PKT** to L2 pkt buffer and inform PsPIN of the new packet to process
Network perspective

1. Match packet to execution context e.g., (IP packets) -> EC_filter

2. Write **PKT** to L2 pkt buffer and inform PsPIN of the new packet to process

3. Schedule the packet to a cluster (task: pkt pointer, handler fun)
Network perspective

1. Match packet to execution context e.g., (IP packets) -> EC_filter

2. Write \textbf{PKT} to L2 pkt buffer and inform PsPIN of the new packet to process

3. Schedule the packet to a cluster (task: pkt pointer, handler fun)

4. Copy packet to L1 and run the handler

**Execution context: EC_filter**

- filter\_hh(), filter\_ph(), filter\_th();
- NIC memory: \textbf{STATE}
- Host buffer: \textbf{BUF}
Network perspective

1. Match packet to execution context e.g., (IP packets) \(\rightarrow\) EC_filter

2. Write \(\text{PKT}\) to L2 pkt buffer and inform PsPIN of the new packet to process

3. Schedule the packet to a cluster (task: pkt pointer, handler fun)

4. Copy packet to L1 and run the handler

Execution context: EC_filter
- filter\_hh(), filter\_ph(), filter\_th();
- NIC memory: STATE
- Host buffer: BUF

Scheduling overhead:
- 64 B packets: 12 ns
- 1 KiB packets: 26 ns
# Circuit Complexity, Performance, and Efficiency

GlobalFoundries 22nm FDSOI @ 1GHz

<table>
<thead>
<tr>
<th>Component</th>
<th>Cluster 0</th>
<th>Cluster 1</th>
<th>Cluster 2</th>
<th>Cluster 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet Scheduler</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMA CSCHED</td>
<td>H H H H</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scratchpad</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMA engine (off-cluster)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command unit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMA CSCHED</td>
<td>H H H H</td>
<td>H H H H</td>
<td>H H H H</td>
<td></td>
</tr>
<tr>
<td>Scratchpad</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitoring &amp; control</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DMA CSCHED</td>
<td>H H H H</td>
<td>H H H H</td>
<td>H H H H</td>
<td>H H H H</td>
</tr>
<tr>
<td>Scratchpad</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Packet buffer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Handler memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Circuit Complexity, Performance, and Efficiency

GlobalFoundries 22nm FDSOI @ 1GHz

Area:
95 MGE (18.5 mm², 70% layout density)

Power:
6.1 W (98% dynamic power, worst case)
Circuit Complexity, Performance, and Efficiency

GlobalFoundries 22nm FDSOI @ 1GHz

Area:
95 MGE (18.5 mm², 70% layout density)

Power:
6.1 W (98% dynamic power, worst case)
Circuit Complexity, Performance, and Efficiency

GlobalFoundries 22nm FDSOI @ 1GHz

Area:
95 MGE (18.5 mm², 70% layout density)

Power:
6.1 W (98% dynamic power, worst case)

Mellanox BlueField: 16 A72 64bit cores
Estimated area: 51 mm²
Circuit Complexity, Performance, and Efficiency

GlobalFoundries 22nm FDSOI @ 1GHz

**Area:**
- 95 MGE (18.5 mm², 70% layout density)

**Power:**
- 6.1 W (98% dynamic power, worst case)

Mellanox BlueField: 16 A72 64bit cores
Estimated area: 51 mm²
Circuit Complexity, Performance, and Efficiency

GlobalFoundries 22nm FDSOI @ 1GHz

**Area:**
95 MGE (18.5 mm², 70% layout density)

**Power:**
6.1 W (98% dynamic power, worst case)

Cycle-accurate simulations

Mellanox BlueField: 16 A72 64bit cores
Estimated area: 51 mm²

> 10x area efficiency (Gb/s/area)

ARM Cortex-A53 @ 1.2 GHz
(4 cores, 2-way superscalar, 64-bit)
Circuit Complexity, Performance, and Efficiency

GlobalFoundries 22nm FDSOI @ 1GHz

Area:
95 MGE (18.5 mm², 70% layout density)

Power:
6.1 W (98% dynamic power, worst case)

Cycle-accurate simulations

Mellanox BlueField: 16 A72 64bit cores
Estimated area: 51 mm²

zynq

> 10x area efficiency (Gb/s/area)

ARM Cortex-A53 @ 1.2 GHz
(4 cores, 2-way superscalar, 64-bit)

ault @ CSCS

> 100x area efficiency (Gb/s/area)

Xeon Gold @ 3 GHz
(18-core, 4-way superscalar, OOO, 64-bit)
Network-accelerated datatypes

Quantization

Erasure coding

Distributed File Systems

Zoo-sPINNER

consensus on sPIN

Network Group Communication

Packet classification and pattern matching

In-network allreduce

Serverless sPIN
Network-accelerated datatypes

Quantization

Erasure coding

Distributed File Systems

Zoo-sPINNER
consensus on sPIN

Network Group Communication

Packet classification and pattern matching

In-network allreduce

Serverless sPIN
Network-accelerated datatypes
Quantization
Erasure coding
Distributed File Systems
Zoo-sPINNER consensus on sPIN
Network Group Communication
Packet classification and pattern matching
In-network allreduce
Serverless sPIN
Network-accelerated non-contiguous memory transfers
Network-accelerated non-contiguous memory transfers
Network-accelerated non-contiguous memory transfers

Salvatore Di Girolamo, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler.

"Network-accelerated non-contiguous memory transfers." SC ’19
Network-accelerated non-contiguous memory transfers

Salvatore Di Girolamo, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. "Network-accelerated non-contiguous memory transfers." SC ’19
Network-accelerated non-contiguous memory transfers
Network-accelerated non-contiguous memory transfers

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


Salvatore Di Girolamo, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. "Network-accelerated non-contiguous memory transfers." SC ‘19
Network-accelerated non-contiguous memory transfers


Salvatore Di Girolamo, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefer. "Network-accelerated non-contiguous memory transfers." SC ’19


https://specfem3d.readthedocs.io/en/latest/
http://fourier.eng.hmc.edu/e161/lectures/fourier/node10.html
Network-accelerated non-contiguous memory transfers


Salvatore Di Girolamo, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler.

“Network-accelerated non-contiguous memory transfers.” SC ’19
Network-accelerated non-contiguous memory transfers

Salvatore Di Girolamo, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler.

"Network-accelerated non-contiguous memory transfers." SC '19


http://fourier.eng.hmc.edu/e161/lectures/fourier/node10.html

https://specfem3d.readthedocs.io/en/latest/
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*
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*
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
Specialized Handlers
Specialized Handlers

vector
Specialized Handlers

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int
Specialized Handlers

- **spin_vec_t**
  - num_blocks: 3
  - block_size: 2
  - stride: 3
  - base_type: int
Specialized Handlers

NIC Memory

vector

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int
Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int
Specialized Handlers

**NIC Memory**

```plaintext
spin_vec_t:
  num_blocks: 3
  block_size: 2
  stride: 3
  base_type: int
```

**vector**

**Handler**
Specialized Handlers

**NIC Memory**

```
spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int
```

Handler

vector
Specialized Handlers

**NIC Memory**

```c
spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int
```

```c
_handler 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_addr + 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;
}
```
Specialized Handlers

spin_vec_t:
  num_blocks: 3
  block_size: 2
  stride: 3
  base_type: int

vector

_handler vector_payload_handler(handler_args_t *args)
{
  spin_vec_t *ddt_descr = (spin_vec_t *)args->pkt_offset;
  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;
}
Specialized Handlers

**NIC Memory**

- `spin_vec_t`: num_blocks: 3
- `block_size`: 2
- `stride`: 3
- `base_type`: int

```c
_handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->ddt_descr;
    uint32_t num_blocks = args->num_blocks;
    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;
}
```

Load DDT info

Compute host memory destination address
Specialized Handlers

NIC Memory

- **spin_vec_t:**
  - num_blocks: 3
  - block_size: 2
  - stride: 3
  - base_type: int

- **vector:**

  ```c
  vector_payload_handler(handler_args_t *args) {
    spin_vec_t *ddt_descr = (spin_vec_t *) args->args;
    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;
  }
  ```

- **Load DDT info**
- **Compute host memory destination address**
- **DMA all contig. regions contained in the packet**
Specialized Handlers

NIC Memory

**spin_vec_t:**
- num_blocks: 3
- block_size: 2
- stride: 3
- base_type: int

**vector**

Handler

```c
vector_payload_handler(handler_args_t *args) {
    spin_vec_t *ddt_descr = (spin_vec_t *)args->ddt_descr;
    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;
}
```

1. Load DDT info
2. Compute host memory destination address
3. DMA all contig. regions contained in the packet
Specialized Handlers

```
vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args;
    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, stride, DMA_NO_EVENT);
        pkt_payload += stride;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```
Specialized Handlers

vector

indexed

struct

NIC Memory

spin_vec_t:
	num_blocks: 3
	block_size: 2
	stride: 3

type: int

vector_payload_handler(handler_args_t *args)

Load DDT info

Compute host memory destination address

DMA all contig. regions contained in the packet

return SPIN_SUCCESS;
Specialized Handlers

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

Load DDT info

- num_blocks: 3
- block_size: 2
- stride: 3
- base_type: int

Compute host memory destination address

DMA all contig. regions contained in the packet

Specialized Line rate

Throughput (Gbps)

Block Size (B)
Specialized Handlers

```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, stride, DMA_NO_EVENT);
        pkt_payload += stride;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

- **Specialized Line rate**
- **Load DDT info**
- **Compute host memory destination address**
- **DMA all contig. regions contained in the packet**
Specialized Handlers

Load DDT info

Compute host memory destination address

DMA all contig. regions contained in the packet

vector

indexed

struct

Need a different handlers for each possible derived datatype!
Specialized Handlers

Load DDT info
Compute host memory destination address
DMA all contig. regions contained in the packet

Can we define a general handler to process any datatype?
MPI Types Library on sPIN

NIC Memory

Index:
Vector:
Snapshot 0

Index:
Vector:
Snapshot 1

Index:
Vector:
Snapshot 2

Index:
Vector:
Snapshot 3

Index:
Vector:
Snapshot 4

Index:
Vector:
Snapshot 5

Δτ = 2

HuP-Local: each HPU has its own state

Packet Scheduler

HPU 0  HPU 1  HPU 2  HPU 3

20
MPI Types Library on sPIN

HPU-Local: each HPU has its own state

RO-checkpoints: pre-computed checkpoints shared by multiple HPUs (read-only)
MPI Types Library on sPIN

HPU-Local: each HPU has its own state

RO-checkpoints: pre-computed checkpoints shared by multiple HPUs (read-only)
MPI Types Library on sPIN

**HPU-Local**: each HPU has its own state

**RO-checkpoints**: pre-computed checkpoints shared by multiple HPUs (read-only)

**RW-checkpoints**: pre-computed checkpoints shared by multiple HPUs (read/write, fine-grain synchronization)

---

**NIC Memory**

Snapshot 0

Index:
Vector:

Snapshot 1

Index:
Vector:

Snapshot 2

Index:
Vector:

Snapshot 3

Index:
Vector:

Snapshot 4

Index:
Vector:

Snapshot 5

Index:
Vector:

**Packet Scheduler**

**HPU 0**

**HPU 1**

**HPU 2**

**HPU 3**

---

**Graph**

- **Line rate**
  - Specialized
  - HPU-Local
  - RO Checkpoints

- **Throughput (Gbps)**

- **Block Size (B)**
  - 4
  - 16
  - 32
  - 64
  - 128
  - 256
  - 512
  - 1K
  - 2K
  - 4K
  - 8K
  - 16K

**Host Unpack**
**MPI Types Library on sPIN**

**NIC Memory**

- **Index:** #blocks: 2, blocklen: 1, offsets: (0, x), basetype: *
- **Vector:** #blocks: 3, blocklen: 2, stride: 3, basetype: 

**Snapshot 0**
- **Index:**
- **Vector:**

**Snapshot 1**
- **Index:**
- **Vector:**

**Snapshot 2**
- **Index:**
- **Vector:**

**Snapshot 3**
- **Index:**
- **Vector:**

**Snapshot 4**
- **Index:**
- **Vector:**

**Snapshot 5**
- **Index:**
- **Vector:**

**Packet Scheduler**

- **HPU 0**
- **HPU 1**
- **HPU 2**
- **HPU 3**

**Δt = 2**

**HPU-Local:** each HPU has its own state

**RO-checkpoints:** pre-computed checkpoints shared by multiple HPUs (read-only)

**RW-checkpoints:** pre-computed checkpoints shared by multiple HPUs (read/write, fine-grain synchronization)

**Graph:**
- **Line rate**
- **Specialized**
- **RW Checkpoints**
- **HPU-Local**
- **RO Checkpoints**
- **Host Unpack**
Real Applications DDTs

Speedup over host-based unpack

- SW4LITE-X
  - vector
- SPEC-CM
  - Index block
- NAS-MG
  - vector
- SPEC-OC
  - Index block

- WRF-X
  - struct(subarray)
- WRF-Y
  - struct(subarray)
- FFT2D
  - contiguous(vector)
- NAS-LU
  - vector

- LAMMPS-F
  - Index block
- MILC
  - vector(vector)
- SW4LITE-Y
  - vector
- COMB
  - subarray

Color keys:
- Red: RW-CP
- Orange: Specialized
- Purple: Portals 4 (lovec)
Real Applications DDTs

Speedup over host-based unpack

- SW4LITE-X
- SPEC-CM
- NAS-MG
- SPEC-OC
- WRF-X
- WRF-Y
- FFT2D
- NAS-LU
- LAMMPS-F
- MILC
- SW4LITE-Y
- COMB

Speedup

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

- RW-CP
- Specialized
- Portals 4 (lovec)
Real Applications DDTs

Speedup over host-based unpack

sPIN - General handler

Speedup

SW4LITE-X
vector

SPEC-CM
index_block

NAS-MG
vector

SPEC-OCC
index_block

LAMMPS-F
index_block

MILC
vector(vector)

SW4LITE-Y
vector

COMB
subarray

SPEC-CM
index_block

NAS-MG
vector

SPEC-OCC
index_block

LAMMPS-F
index_block

MILC
vector(vector)

SW4LITE-Y
vector

COMB
subarray

SW4LITE-X
vector

SPEC-CM
index_block

NAS-MG
vector

SPEC-OCC
index_block

LAMMPS-F
index_block

MILC
vector(vector)

SW4LITE-Y
vector

COMB
subarray

RW-CP
Specialized
Portals 4 (lovec)
Real Applications DDTs

Speedup over host-based unpack

sPIN - Specialized handler
Real Applications DDTs

Speedup over host-based unpack

Portals 4 - IOVECs

- SW4LITE-X vector
- SPEC-CM index_block
- NAS-MG vector
- SPEC-OQ index_block
- WRF-Y struct(subarray)
- FFT2D contiguous(vector)
- NAS-LU vector
- LAMMPS-F index_block
- MILC vector(vector)
- SW4LITE-Y vector
- COMS subarray

Speedup

- RW-CP
- Specialized
- Portals 4 (lovec)
Real Applications DDTs

Speedup over host-based unpack

- SW4LITE-X
  - Vector
  - Speedup

- SPEC-CM
  - Index block
  - Speedup

- NAS-MG
  - Vector
  - Speedup

- SPEC-OC
  - Index block
  - Speedup

- WRF-X
  - Struct(subarray)
  - Speedup

- WRF-Y
  - Struct(subarray)
  - Speedup

- FFT2D
  - Contiguous(vector)
  - Speedup

- NAS-LU
  - Vector
  - Speedup

- LAMMPS-F
  - Index block
  - Speedup

- MILC
  - Vector(vector)
  - Speedup

- SW4LITE-Y
  - Vector
  - Speedup

- COMB
  - Subarray
  - Speedup

Red: RW-CP
Orange: Specialized
Purple: Portals 4 (lovec)
Real Applications DDTs

Speedup over host-based unpack

- SW4LITE-X
- SPEC-CM
- NAS-MG
- SPEC-OC

- WRF-X
- WRF-Y
- FFT2D
- NAS-LU

- LAMMPS-F
- MILC
- SW4LITE-Y
- COMB

Speedup

Indices:
- struct(subarray)
- index_block

Categories:
- RW-CP
- Specialized
- Portals 4 (lovec)
Real Applications DDTs

Checkpointing Overhead

75% of the analyzed DDTs amortized after 4 reuses

Data Movement

Up to 3.8x less moved data volume

Handler Analysis

Full app speedup (FFT2D)

75% of the analyzed DDTs amortized after 4 reuses

Data Movement

Up to 3.8x less moved data volume

Handler Analysis

Full app speedup (FFT2D)
Network-accelerated datatypes

Quantization

Erasure coding

Distributed File Systems

Zoo-sPINNER consensus on sPIN

Network Group Communication

Packet classification and pattern matching

In-network allreduce

Serverless sPIN
Fast, scalable, and reliable storage is a first-class requirement of both HPC systems and datacenters.
Fast, scalable, and reliable storage is a first-class requirement of both HPC systems and datacenters.

Up to 60% I/O overhead [1].

Fast, scalable, and reliable storage is a first-class requirement of both HPC systems and datacenters.

Up to 60% I/O overhead [1].

Up to 90% I/O overhead [2].


Fast, scalable, and reliable storage is a first-class requirement of both HPC systems and datacenters.

Up to 60% I/O overhead [1].

Efficiency of Distributed File Systems (DFS) is crucial in these systems

Up to 90% I/O overhead [2].


https://www2.cisl.ucar.edu/user-support/allocations/climate-simulation-laboratory-csl


Distributed File Systems
Distributed File Systems
Distributed File Systems

Metadata nodes

Management nodes

Storage node

Storage node

Storage node
Distributed File Systems

- Compute node
  - DFS abstraction

- Storage node

- Metadata nodes
- Management nodes

- Storage node
Distributed File Systems

- **Compute node**
- **DFS abstraction**
- **Storage node**

- **Metadata nodes**
- **Management nodes**
Distributed File Systems
Distributed File Systems with NVMM

Storage node

NVMM

Compute node

DFS abstraction

Storage node

NVMM

Compute node

DFS abstraction

Storage node

NVMM

Storage node

NVMM
Distributed File Systems with NVMM
Distributed File Systems with NVMM

Compute node

DNS abstraction

RDMA

Storage node

NVMM

Compute node

DNS abstraction

Storage node

NVMM

Storage node

NVMM
Distributed File Systems with NVMM

Client request validation

Data replication

Erasure coding
Distributed File Systems with NVMM and sPIN

sPIN adds compute capabilities on the data path
Offloaded data replication

Offloaded data replication

Offloaded data replication

Client → Storage node 0 → Client
Client → Storage node 1 → Client
Client → Storage node 2 → Client

Storage node 0 → Storage node 1 → Storage node 2
Storage node 0 → Storage node 1 → Storage node 2
Storage node 0 → Storage node 1 → Storage node 2

RDMA

sPIN
Offloaded data replication

Client

Storage node 0

Storage node 1

Storage node 2

Client

Storage node 0

Storage node 1

Storage node 2
Offloaded data replication
Offloaded data replication

Latency (us)

Write size (KiB)

CPU-chain
CPU-pbt
RDMA-flat

Client
Storage node 0
Storage node 1
Storage node 2
Client
Storage node 0
Storage node 1
Storage node 2

Data replication
Offloaded data replication

Up to 4x lower latency
(replication factor: 4)
Network-accelerated datatypes

Quantization

Erasure coding

Distributed File Systems

Zoo-sPINNER consensus on sPIN

Network Group Communication

Packet classification and pattern matching

In-network allreduce

Serverless sPIN
In-network allreduce

Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21
In-network allreduce

Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21
In-network allreduce

Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21
In-network allreduce

Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21
In-network allreduce

In-network allreduce

In-network allreduce

Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21
In-network allreduce

In-network allreduce
In-network allreduce

2x traffic reduction compared to host-based allreduce

Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21
In-network allreduce

2x traffic reduction compared to host-based allreduce

2x bandwidth improvement

Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21
Missing features
Missing features

Custom operators and datatypes
Missing features

Custom operators and datatypes

- int4
- float16
- int8
Missing features

Custom operators and datatypes
Missing features

Custom operators and datatypes

Support for sparse data
Missing features

Support for sparse data
Missing features

Support for sparse data
Missing features

Support for sparse data
Missing features

Custom operators and datatypes

Support for sparse data

Reproducibility
Missing features

Reproducibility
Missing features

Custom operators and datatypes

Support for sparse data

Reproducibility
Existing switches architectures
Existing switches architectures
Existing switches architectures
Existing switches architectures
PsPIN-equipped switches

Switch ports
PsPIN-equipped switches

Switch ports

Parser
PsPIN-equipped switches

Switch ports → Parser → Processing unit
PsPIN-equipped switches

Switch ports → Parser → Processing unit → Routing
PsPIN-equipped switches

Switch ports → Parser → Processing unit → Routing
PsPIN-equipped switches
PsPIN-equipped switches

Switch ports ➔ Parser ➔ Processing unit ➔ Routing ➔ Crossbar
PsPIN-equipped switches
PsPIN-equipped switches
Results – single switch

- Flare (Single Buffer)
- Flare (Multiple Buffers)
- Flare (Tree)

Bandwidth (Tbps)

Data Size

1KiB  4KiB  512KiB  1MiB

SHARP

SwitchML

Bandwidth (Elements per sec)

Data Type

int32int16int8 float

1e11
Results – 64 nodes, 2-level fat tree

**Communication time** of a ResNet50 iteration with **sparsified gradients** (0.2% density)
Wrapping up
Wrapping up

In this presentation

- Programming model (SC ’17)
- HW accelerator (ISCA ’21)
- Use cases (SC ‘19, SC’21)
Wrapping up

In this presentation

- Programming model (SC ’17)
- HW accelerator (ISCA ’21)
- Use cases (SC ‘19, SC’21)

Our research directions

- sPIN in the cloud
- Prototyping PsPIN
- More use cases!
Wrapping up

In this presentation

- Programming model (SC ’17)
- HW accelerator (ISCA ’21)
- Use cases (SC ‘19, SC’21)

Our research directions

- sPIN in the cloud
- Prototyping PsPIN
- More use cases!

https://github.com/spcl/pspin
RTL, runtime, examples
OSMOSIS
OSMOSIS
OSMOSIS

Isolation?

Multitenancy?

QoS?
Prototyping PsPIN

Use Case 1: Broadcast acceleration

Use Case 1: Broadcast acceleration

Use Case 1: Broadcast acceleration

Use Case 1: Broadcast acceleration

Use Case 1: Broadcast acceleration

Message size: 8 Bytes

Use Case 1: Broadcast acceleration

Portals 4

Message size: 8 Bytes

Latency (us)

Number of Processes

Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. *HOTI’11*
Use Case 1: Broadcast acceleration

Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. HOTI’11
Use Case 1: Broadcast acceleration

Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. *HOTI’11*
Use Case 1: Broadcast acceleration

Portals 4

Message size: 8 Bytes

Latency (us)

Number of Processes

Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. HOTI’11
Use Case 1: Broadcast acceleration

Portals 4

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

Use Case 1: Broadcast acceleration

Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. *HOTI’11*
Use Case 1: Broadcast acceleration

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

Liu, J., et al., High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming 2004*
Use Case 1: Broadcast acceleration

Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. *HOTI*’11
Use Case 1: Broadcast acceleration

![Image of network cards and memory modules]

Message size: 8 Bytes

Latency (us)

Number of Processes

Offloaded collectives (e.g., ConnectX-2, Portals 4)

RDMA

sPIN

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

Use Case 1: Broadcast acceleration

Message size: 8 Bytes

Handlers cost: 24 instructions + Log P Puts

Offloaded collectives (e.g., ConnectX-2, Portals 4)

Latency (us)

Number of Processes

RDMA

sPIN

Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. HOTI’11
Use Case 2: RAID acceleration

Server Node

Parity Node

Distributed Data Management

Shankar D. et al., High-performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads. ICDCS’17
Use Case 2: RAID acceleration

Shankar D. et al., High-performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads. ICDCS’17
Use Case 2: RAID acceleration

Parity Update

Server Node

Parity Node

Write

RDMA

Distributed Data Management

Shankar D. et al., High-performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads. ICDCS’17
Use Case 2: RAID acceleration

Shankar D. et al., High-performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads. ICDCS’17
Use Case 2: RAID acceleration

Server Node

Parity Node

Write

Parity Update

ACK

Parity ACK

RDMA

Shankar D. et al., High-performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads. ICDCS'17
Use Case 2: RAID acceleration

Shankar D. et al., High-performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads. ICDCS’17
Use Case 2: RAID acceleration

Shankar D. et al., High-performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads. ICDCS’17
Use Case 2: RAID acceleration

Server Node

Parity Node

Write

Parity Update

sPIN

Parity ACK

ACK

20% lower latency

176% higher BW

Completion Time (us)

Number of Transferred Bytes

Handlers cost:
Server: 58 instructions + 1 Put
Parity: 46 instructions + 1 Put

Shankar D. et al., High-performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads. ICDCS’17
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>
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>
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>
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

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

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

Discarded data: 80%

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>

41% lower latency


Use Case 5: Distributed KV Store


Use Case 6: Conditional Read

Discarded data: 80%
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</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</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>msg</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 360</td>
<td>360</td>
<td>28.1M</td>
<td>6.5%</td>
<td>2.8%</td>
<td>58%</td>
</tr>
<tr>
<td>Cloverleaf 72</td>
<td>2.7M</td>
<td>5.2%</td>
<td>2.4%</td>
<td>53%</td>
<td></td>
</tr>
<tr>
<td>Cloverleaf 360</td>
<td>15.3M</td>
<td>5.6%</td>
<td>3.2%</td>
<td>42%</td>
<td></td>
</tr>
</tbody>
</table>

41% lower latency


Use Case 5: Distributed KV Store

Network

Use Case 6: Conditional Read

Discarded data: 80%


Use Case 7: Distributed Transactions

Network

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

Use Case 8: FT Broadcast

Network

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

Use Case 8: FT Broadcast

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

Discarded data: 80%

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

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

Use Case 8: FT Broadcast

Network

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

Use Case 9: Distributed Consensus

Network

István, Z., et al., Consensus in a Box: Inexpensive Coordination in Hardware. NSDI’16
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

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
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

Use Case 6: Conditional Read

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

Use Case 9: Distributed Consensus

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

István, Z., et al., Consensus in a Box: Inexpensive Coordination in Hardware. NSDI’16
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**

The Next 700 sPIN use-cases

41% lower latency


**Use Case 6: Conditional Read**

Discarded data: 80%

**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
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%</td>
</tr>
<tr>
<td>POP</td>
<td>64</td>
<td>772M</td>
<td>3.1%</td>
<td>2.4%</td>
</tr>
<tr>
<td>coMD</td>
<td>72</td>
<td>5.3M</td>
<td>6.1%</td>
<td>2.4%</td>
</tr>
<tr>
<td>coMD</td>
<td>360</td>
<td>28.1M</td>
<td>6.5%</td>
<td>2.8%</td>
</tr>
<tr>
<td>Cloverleaf</td>
<td>72</td>
<td>2.7M</td>
<td>5.2%</td>
<td>2.4%</td>
</tr>
<tr>
<td>Cloverleaf</td>
<td>360</td>
<td>15.3M</td>
<td>5.6%</td>
<td>3.2%</td>
</tr>
</tbody>
</table>

Use Case 5: Distributed KV Store

The Next 700 sPIN use-cases

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

41% lower latency


Use Case 6: Conditional Read

Discarded data: 80%

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
400G Data Path

L2 packet memory

L2 handler memory

L2 program memory

Cluster 0
DMA L1

Cluster 1
DMA L1

Cluster 2
DMA L1

Cluster 3
DMA L1

Host slv (AXI2PCI)

IOMMU

Host mst (AXI2PCI)

NIC inbound

NIC outbound

DMA (off-cluster)

Host Direct
400G Data Path

- L2 packet memory
- L2 handler memory
- L2 program memory

DMA interconnect

PE interconnect

NHI interconnect

Cluster 0: DMA L1
Cluster 1: DMA L1
Cluster 2: DMA L1
Cluster 3: DMA L1

Host mst (AXI2PCI)

NIC inbound

NIC outbound

DMA (off-cluster)

Host slv (AXI2PCI)

IOMMU

Host Direct
400G Data Path

- **L2 packet memory**
- **L2 handler memory**
- **L2 program memory**

**DMA interconnect**

- **Cluster 0**
  - DMA
  - L1
- **Cluster 1**
  - DMA
  - L1
- **Cluster 2**
  - DMA
  - L1
- **Cluster 3**
  - DMA
  - L1

**PE interconnect**

**NHI interconnect**

- **Host slv (AXI2PCI)**
  - IOMMU
  - mux

**NIC inbound**

**NIC outbound**

**DMA (off-cluster)**

**Host Direct**

**Host mst (AXI2PCI)**

**NIC inbound**

**DMA interconnect**

- AXI4 512 bit
- AXI4 32 bit

**write**

**read**

**multiplexer (mux)**
400G Data Path

- L2 packet memory
- L2 handler memory
- L2 program memory
- DMA interconnect
- PE interconnect
- NH1 interconnect
- Cluster 0
  - DMA
  - L1
- Cluster 1
  - DMA
  - L1
- Cluster 2
  - DMA
  - L1
- Cluster 3
  - DMA
  - L1
- Host mst (AXI2PCI)
- NIC inbound
- NIC outbound
- DMA (off-cluster)
- Host slv (AXI2PCI)
- IOMMU
- Host Direct

AXI4 512 bit → AXI4 32 bit
400G Data Path

- **L2 packet memory**
- **L2 handler memory**
- **L2 program memory**

**Clusters**:
- **Cluster 0**: DMA, L1
- **Cluster 1**: DMA, L1
- **Cluster 2**: DMA, L1
- **Cluster 3**: DMA, L1

**Interconnects**:
- **DMA interconnect**
- **PE interconnect**
- **NHI interconnect**

**Connections**:
- **AXI4 512 bit**
- **AXI4 32 bit**

- **Host slv (AXI2PCI)**
- **IOMMU**
- **Host Direct**

**Other connecting points**:
- **NIC inbound**
- **NIC outbound**
- **DMA (off-cluster)**

**Additional notes**:
- AXI2PCI
- Host mst
- mux
400G Data Path

- **L2 packet memory**
- **L2 handler memory**
- **L2 program memory**

**DMA interconnect**

**PE interconnect**

**L2 packet**
- **Cluster 0**
  - DMA
  - L1
- **Cluster 1**
  - DMA
  - L1
- **Cluster 2**
  - DMA
  - L1
- **Cluster 3**
  - DMA
  - L1

**NIC outbound**

**Host slv (AXI2PCI)**

**IOMMU**

**Host Direct**

**DMA (off-cluster)**

**NHI interconnect**

AXI4 512 bit → AXI4 32 bit
400G Data Path

- DMA interconnect
- PE interconnect
- NHI interconnect

- Cluster 0: DMA → L1
- Cluster 1: DMA → L1
- Cluster 2: DMA → L1
- Cluster 3: DMA → L1

- Host mst (AXI2PCI)
- NIC inbound
- NIC outbound
- DMA (off-cluster)

- mux
- Direct

- 400G in
- 400G out

- AXI4 512 bit → AXI4 32 bit
NIC integration

- Outbound Engine
- Inbound Engine
- Command Unit
- PsPIN Unit
- Network Interface
- Host Interface
- Packet Scheduler
- DMA engine (off-cluster)
- Command unit
- Monitoring & control
- Cluster 0
  - L1 TCDM
  - DMA: H H H H
  - CSCHED: H H H H
- Cluster 1
  - L1 TCDM
  - DMA: H H H H
  - CSCHED: H H H H
- Cluster 2
  - L1 TCDM
  - DMA: H H H H
  - CSCHED: H H H H
- Cluster 3
  - L1 TCDM
  - DMA: H H H H
  - CSCHED: H H H H
- L2 packet buffer
- L2 program memory
- L2 handler memory
NIC integration

**Match-action tables**

- **Parser**
  - ETH
  - IPv4
  - IPv6
  - TCP

- **Match action**
  - **Match:** QUIC.flow_id == 1234
  - **Action:** process with EC X

**Network Interface**
- Outbound Engine
- Inbound Engine
- Command Unit
- PsPIN Unit

**Host Interface**

**PsPIN Unit**

**Cluster 1**
- DMA
- CSCHED
- L1 TCDM

**Cluster 2**
- DMA
- CSCHED
- L1 TCDM

**Cluster 3**
- DMA
- CSCHED
- L1 TCDM

**L2 program memory**

**L2 handler memory**
NIC integration

**Match-action tables**

- **Parser**
  - ETH
  - IPv4
  - IPv6
  - TCP

- **Match action**
  - Match: QUIC.flow_id == 1234
  - Action: process with EC X

**RDMA NICs**

- **Standard queue pairs**
- **Processing queue pairs**

**Network Interface**

- Outbound Engine
- Inbound Engine
- Command Unit
- PsPIN Unit

**Host Interface**

- PsPIN Unit

**Outbound Engine**

- Host Interface
- Command Unit
- PsPIN Unit

**Inbound Engine**

- Host Interface
- Command Unit
- PsPIN Unit

**RDMA NICs**

- Host Interface
- PsPIN Unit

**Command Unit**

- PsPIN Unit

**PsPIN Unit**

- Host Interface
- PsPIN Unit
Low latency, full throughput

- Highly parallel
- Fast scheduling
- Fast explicit memory access

Low latency, full throughput
Highly parallel

Fast scheduling

Fast explicit memory access

32 cores, higher core-count configurations are possible with more clusters

Tens of nanoseconds to get handlers started

Single-cycle L1 memory
### Highly Parallel
- Fast scheduling
- Fast explicit memory access

- **32 cores, higher core-count configurations are possible with more clusters**
- **Tens of nanoseconds to get handlers started**
- **Single-cycle L1 memory**

### Stateful computation support

- **Support for wide range of use cases**
- **Handlers isolation**
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low latency, full throughput</td>
<td>Highly parallel</td>
<td>32 cores, higher core-count configurations are possible with more clusters</td>
</tr>
<tr>
<td>Fast scheduling</td>
<td>Fast explicit memory access</td>
<td>Tens of nanoseconds to get handlers started</td>
</tr>
<tr>
<td>Support for wide range of use cases</td>
<td>Stateful computation support</td>
<td>Implicit in the sPIN programming model</td>
</tr>
<tr>
<td></td>
<td>Handlers isolation</td>
<td>HW-configured (1 cycle) RISC-V PMP</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Status</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Area and power efficiency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configurability</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy to integrate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low latency, full throughput</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Highly parallel</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Fast scheduling</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Fast explicit memory access</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Support for wide range of use cases</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stateful computation support</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Handlers isolation</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Implicit in the sPIN programming model</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32 cores, higher core-count configurations</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>are possible with more clusters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tens of nanoseconds to get handlers started</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single-cycle L1 memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware-configured (1 cycle) RISC-V PMP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>Low latency, full throughput</td>
<td>Highly parallel, Fast scheduling, Fast explicit memory access</td>
<td>32 cores, higher core-count configurations are possible with more clusters</td>
</tr>
<tr>
<td>Support for wide range of use cases</td>
<td>Stateful computation support, Handlers isolation</td>
<td>Implicit in the sPIN programming model</td>
</tr>
<tr>
<td>Easy to integrate</td>
<td>Area and power efficiency, Configurability</td>
<td>18.5 mm², 6.1 W</td>
</tr>
</tbody>
</table>

- **32 cores, higher core-count configurations are possible with more clusters**
- **Tens of nanoseconds to get handlers started**
- **Single-cycle L1 memory**
- **Implicit in the sPIN programming model**
- **HW-configured (1 cycle) RISC-V PMP**
- **Configurable number of clusters and cores/cluster**
Experimental results
PsPIN Throughput and utilization

**Handler complexity**

<table>
<thead>
<tr>
<th>Throughput (Gbit/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
</tr>
<tr>
<td>400</td>
</tr>
<tr>
<td>300</td>
</tr>
<tr>
<td>200</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

- theoretical
- misaligned (size+1B)
- 64 B
- 512 B
- 1024 B

**Utilization**

<table>
<thead>
<tr>
<th>Max # HPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

- 64 B
- 512 B
- 1024 B

**Outbound NIC flow**

- Throughput (Gbit/s)
- Packet size (B)
- data from L1
- data from L2

**Outbound host flow**

- Throughput (Gbit/s)
- Packet size (B)
- data from L1
- data from L2
Handlers Characterization
Handlers Characterization

Packet steering
filtering, strided datatypes

Data movement
key-value store

Full packet processing
aggregate, histogram, reduce
Handlers Characterization

Packet steering
filtering, strided datatypes

Data movement
key-value store

Full packet processing
aggregate, histogram, reduce

Packet Scheduler

DMA engine
(off-cluster)

Command unit

Monitoring & control

Cluster 0
DMA
CSCHED
L1 TCDM
H H H H

Cluster 1
DMA
CSCHED
L1 TCDM
H H H H

Cluster 2
DMA
CSCHED
L1 TCDM
H H H H

Cluster 3
DMA
CSCHED
L1 TCDM
H H H H

L2 packet buffer
L2 program memory
L2 handler memory

DMA
CSCHED
Handlers Characterization

Packet steering
filtering, strided datatypes

Data movement
key-value store

Full packet processing
aggregate, histogram, reduce

Packet Scheduler

DMA engine
(off-cluster)

Command unit

Monitoring & control

Cluster 0
L1 TCDM
DMA
CSCHED

Cluster 1
L1 TCDM
DMA
CSCHED

Cluster 2
L1 TCDM
DMA
CSCHED

Cluster 3
L1 TCDM
DMA
CSCHED

L2 packet buffer

L2 program memory

L2 handler memory

<table>
<thead>
<tr>
<th></th>
<th>aggregate</th>
<th>filtering</th>
<th>histogram</th>
<th>kvstore</th>
<th>reduce</th>
<th>strided_ddt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gb/s</td>
<td>200</td>
<td>300</td>
<td>400</td>
<td>500</td>
<td>600</td>
<td>700</td>
</tr>
<tr>
<td>512 B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1024 B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64 B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Handlers Characterization

- Packet steering: filtering, strided datatypes
- Data movement: key-value store
- Full packet processing: aggregate, histogram, reduce

![Graph showing Gb/s for different packet types and sizes](image)

- Aggregate: 200 Gbit/s
- Filtering: 400 Gbit/s
- Histogram: 200 Gbit/s
- Kvstore: 400 Gbit/s
- Reduce: 200 Gbit/s
- Strided_ddt: 400 Gbit/s

![Graph showing max handler time vs packet size](image)

- 1 ns = 1 cycle @1GHz
- 200 Gbit/s
- 400 Gbit/s

---

Cluster 0
- L1 TCDM
- DMA
- CSCHED

Cluster 1
- L1 TCDM
- DMA
- CSCHED

Cluster 2
- L1 TCDM
- DMA
- CSCHED

Cluster 3
- L1 TCDM
- DMA
- CSCHED

L2 packet buffer
L2 program memory
L2 handler memory
How about other architectures?
How about other architectures?

ault @ CSCS

Xeon Gold @ 3 GHz
(18-core, 4-way superscalar, OOO, 64-bit)

zynq

ARM Cortex-A53 @ 1.2 GHz
(4 cores, 2-way superscalar, 64-bit)

PsPIN

RISC-V (RISC-V) @ 1 GHz
(32 cores, single-issue, in-order, 32-bit)
How about other architectures?

ault @ CSCS

Xeon Gold @ 3 GHz
(18-core, 4-way superscalar, OOO, 64-bit)

zynq

ARM Cortex-A53 @ 1.2 GHz
(4 cores, 2-way superscalar, 64-bit)

PsPIN

RI5CY (RISC-V) @ 1 GHz
(32 cores, single-issue, in-order, 32-bit)

Per-core throughput

- ault
- zynq
- PsPIN

Gbit/s
How about other architectures?

ault @ CSCS

Xeon Gold @ 3 GHz
(18-core, 4-way superscalar, OOO, 64-bit)

zynq

ARM Cortex-A53 @ 1.2 GHz
(4 cores, 2-way superscalar, 64-bit)

PsPIN

RISCY (RISC-V) @ 1 GHz
(32 cores, single-issue, in-order, 32-bit)

<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>
How about other architectures?

ault @ CSCS

Xeon Gold @ 3 GHz
(18-core, 4-way superscalar, OOO, 64-bit)

zynq

ARM Cortex-A53 @ 1.2 GHz
(4 cores, 2-way superscalar, 64-bit)

PsPIN

RI5CY (RISC-V) @ 1 GHz
(32 cores, single-issue, in-order, 32-bit)
Comparison vs other architectures

ault @ CSCS
- Xeon Gold @ 3 GHz
  - (18-core, 4-way superscalar, OOO, 64-bit)

zynq
- ARM Cortex-A53 @ 1.2 GHz
  - (4 cores, 2-way superscalar, 64-bit)

PsPIN
- RI5CY (RISC-V) @ 1 GHz
  - (32 cores, single-issue, in-order, 32-bit)
Other SmartNICs

- **Netronome/P4-based NICs**
  - Different philosophy
  - Offloaded computation is not per-application (vs sPIN per-application packet handlers)
    - *No isolation (computation on the NIC sees all packets)*
  - Introduce limitation on the offloaded computation
    - *e.g., XDP is not Turing complete*
  - Not open source 😎

- **INCA/off-path SmartNICs**
  - Complementary to sPIN/PsPIN
How about other architectures?

- **Xeon Gold @ 3 GHz** (18-core, 4-way superscalar, OOO, 64-bit)
- **ARM Cortex-A53 @ 1.2 GHz** (4 cores, 2-way superscalar, 64-bit)
- **PsPIN**
  - **RI5CY (RISC-V) @ 1 GHz** (32 cores, single issue, in-order, 32-bit)

---

**A RISC-V in-network accelerator for flexible high-performance low-power packet processing**

Salvatore Di Girolamo*, Andrea Kirsch†, Alexandru Caltioae*, Thomas Benz†, Timo Schneider*, Jakub Beranek†, Luca Benini†, Torsten Hoefler*

*Dept. of Computer Science, ETH Zürich, Switzerland  
†Integrated System Laboratory, ETH Zürich, Switzerland  
‡IITInnovations, VSB - Technical University of Ostrava

Abstract—The capacity of offloading data and control tasks to the network is becoming increasingly important, especially if we consider the faster growth of network speed compared to CPU frequencies. In-network computing alleviates the host CPU load by running tasks directly in the network, enabling additional computation/communication overlap and potentially improving overall application performance. However, sustaining bandwidths provided by next-generation networks, e.g., 400 Gb/s, can become a challenge. PsPIN is a programming model for in-NIC compute, where users specify handler functions that are executed on the NIC, for each incoming packet belonging to a given message or flow. It enables a CUDA-like acceleration, where the NIC is equipped with lightweight processing elements that process network packets in parallel. We investigate the architectural specializations that a PsPIN NIC should provide to enable high-performance, low-power, and flexible packet processing. We introduce PsPIN, a first open-source PSIN implementation, based on a multi-cluster RISC-V architecture and designed according to the identified architectural specializations. We investigate the performance of PsPIN with cycle-accurate simulations, showing that it can process packets at 400 Gb/s for several use cases. Introducing minimal latencies (26 ns for 44 B packets) and occupying a total area of 18.5 mm² (22 nm FDSOI), PsPIN inserts into network packet processing specializations into user-space memory. Even though this greatly reduces packet processing overheads on the CPU, the incoming data must still be processed. A flurry of specialized technologies exists to move additional parts of this processing into network cards, e.g., FPGAs virtualization support [22], P4 simple rewriting rules [13], or triggered operations [9].

Streaming processing in the network (SPIN) [28] defines a unified programming model and architecture for network acceleration beyond simple RDMA. It provides a user-level interface, similar to CUDA for compute acceleration, considering the specialties and constraints of low-latency line-rate packet processing. It defines a flexible and programmable network instruction set architecture (NISA) that not only lowers the barrier of entry but also supports a large set of use-cases [28]. For example, Di Girolamo et al. demonstrate up to 10X speedups for serialization and deserialization (marshalling) of non-consecutive data [20].

While the NISA defined by SPIN can be implemented on existing SmartNICs [1], their microarchitecture (often standard ARM SocC) is not optimized for packet-processing tasks. In
RI5CY Core

- 4-stage pipeline, in-order, optimized for energy efficiency
- Area: 40 kGE
- Critical path: 30 logic levels of critical path
- CoreMark/MHZ 3.19
- Includes various extensions (X) to RISC-V (SIMD, Fixed point, bit manipulation, hw loops)

Options:
- FPU: IEEE 754 single precision (+ 40-70 kGE)
- U/M privileges

https://pulp-platform.org/docs/hipeac/pulp_intro_kgf.pdf
But why PULP/RISC-V?
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*
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
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
  - Open source!
  - 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)
  - Actively researched
Custom ISA Extensions

```c
for (i = 0; i < 100; i++)
    d[i] = a[i] + b[i];
```
Custom ISA Extensions

```c
for (i = 0; i < 100; i++)
    d[i] = a[i] + b[i];
```
Custom ISA Extensions

for (i = 0; i < 100; i++)
    d[i] = a[i] + b[i];

mv x5, 0
mv x4, 100
Lstart:
    lb x2, 0(x10)
    lb x3, 0(x11)
    addi x10, x10, 1
    addi x11, x11, 1
    add x2, x3, x2
    sb x2, 0(x12)
    addi x4, x4, -1
    addi x12, x12, 1
    bne x4, x5, Lstart
Custom ISA Extensions

for (i = 0; i < 100; i++)
  d[i] = a[i] + b[i];

mv x5, 0
mv x4, 100
Lstart:
  lb x2, 0(x10)
  lb x3, 0(x11)
  addi x10, x10, 1
  addi x11, x11, 1
  add x2, x3, x2
  sb x2, 0(x12)
  addi x4, x4, -1
  addi x12, x12, 1
  bne x4, x5, Lstart

11 cycles/output
Custom ISA Extensions

for (i = 0; i < 100; i++)
    d[i] = a[i] + b[i];

mv x5, 0
mv x4, 100
Lstart:
    lb x2, 0(x10)
    lb x3, 0(x11)
    addi x10, x10, 1
    addi x11, x11, 1
    add x2, x3, x2
    sb x2, 0(x12)
    addi x4, x4, -1
    addi x12, x12, 1
    bne x4, x5, Lstart

11 cycles/output
Custom ISA Extensions

for (i = 0; i < 100; i++)
    d[i] = a[i] + b[i];

11 cycles/output
Custom ISA Extensions

```assembly
for (i = 0; i < 100; i++)
    d[i] = a[i] + b[i];
```

```
mv x5, 0
mv x4, 100
Lstart:
    lb x2, 0(x10)
    lb x3, 0(x11)
    addi x10, x10, 1
    addi x11, x11, 1
    add x2, x3, x2
    sb x2, 0(x12)
    addi x4, x4, -1
    bne x4, x5, Lstart
```

Auto-incremental load/store

11 cycles/output  8 cycles/output
Custom ISA Extensions

for (i = 0; i < 100; i++)
    d[i] = a[i] + b[i];

mv x5, 0
mv x4, 100
Lstart:
    lb x2, 0(x10)
    lb x3, 0(x11)
    addi x10, x10, 1
    addi x11, x11, 1
    add x2, x3, x2
    sb x2, 0(x12)
    addi x4, x4, -1
    add x2, x3, x2
    sb x2, 0(x12!)
    bne x4, x5, Lstart

mv x5, 0
mv x4, 100
Lstart:
    lb x2, 0(x10!)
    lb x3, 0(x11!)
    addi x4, x4, -1
    add x2, x3, x2
    sb x2, 0(x12!)
    bne x4, x5, Lstart

lp.setupi 100, Lend
    lb x2, 0(x10!)
    lb x3, 0(x11!)
    add x2, x3, x2
Lend: sb x2, 0(x12!)

Auto-increment load/store

11 cycles/output  8 cycles/output

HW loop
Custom ISA Extensions

for (i = 0; i < 100; i++)
    d[i] = a[i] + b[i];

mv x5, 0
mv x4, 100
Lstart:
    lb x2, 0(x10)
    lb x3, 0(x11)
    addi x10, x10, 1
    addi x11, x11, 1
    add x2, x3, x2
    sb x2, 0(x12)
    addi x4, x4, -1
    add x2, x3, x2
    sb x2, 0(x12!)
    bne x4, x5, Lstart

mv x5, 0
mv x4, 100
Lstart:
    lb x2, 0(x10!)
    lb x3, 0(x11!)
    addi x4, x4, -1
    add x2, x3, x2
    sb x2, 0(x12!)
    bne x4, x5, Lstart

lp.setupsi 100, Lend
    lb x2, 0(x10!)
    lb x3, 0(x11!)
    add x2, x3, x2
Lend: sb x2, 0(x12!)

HW loop

Auto-incr load/store

11 cycles/output 8 cycles/output 5 cycles/output
Custom ISA Extensions

```plaintext
for (i = 0; i < 100; i++)
    d[i] = a[i] + b[i];
```

mv x5, 0
mv x4, 100
Lstart:
    lb x2, 0(x10)
    lb x3, 0(x11)
    addi x10, x10, 1
    addi x11, x11, 1
    add x2, x3, x2
    sb x2, 0(x12)
    addi x4, x4, -1
    add x2, x3, x2
    sb x2, 0(x12!)
    bne x4, x5, Lstart

mv x5, 0
mv x4, 100

Lstart:
    lb x2, 0(x10!)  
    lb x3, 0(x11!)
    addi x4, x4, -1
    add x2, x3, x2
    sb x2, 0(x12!)
    bne x4, x5, Lstart

```plaintext
lp.setupi 100, Lend
    lb x2, 0(x10!)
    lb x3, 0(x11!)
    add x2, x3, x2
Lend: sb x2, 0(x12!)
```

```plaintext
lp.setupi 25, Lend
    lw x2, 0(x10!)
    lw x3, 0(x11!)
    pv.add.b x2, x3, x2
Lend: sw x2, 0(x12!)
```

11 cycles/output 8 cycles/output 5 cycles/output

Auto-incr load/store

HW loop

Packed SIMD
Custom ISA Extensions

```
for (i = 0; i < 100; i++)
    d[i] = a[i] + b[i];
```

```
mv x5, 0
mv x4, 100
Lstart:
    lb x2, 0(x10)
    lb x3, 0(x11)
    addi x10, x10, 1
    addi x11, x11, 1
    add x2, x3, x2
    sb x2, 0(x12)
    bne x4, x5, Lstart
```

```
mv x5, 0
mv x4, 100
Lstart:
```

```
lp.setupi 100, Lend
    lb x2, 0(x10!)
    lb x3, 0(x11!)
    add x2, x3, x2
    sb x2, 0(x12!)
Lend:
    bne x4, x5, Lstart
```

```
lp.setupi 25, Lend
    lw x2, 0(x10!)
    lw x3, 0(x11!)
    pv.add.b x2, x3, x2
Lend:
    sw x2, 0(x12!)
```

**Auto-increment load/store**

<table>
<thead>
<tr>
<th>Method</th>
<th>Cycles/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto-incr load/store</td>
<td>11 cycles/output</td>
</tr>
<tr>
<td>HW loop</td>
<td>8 cycles/output</td>
</tr>
<tr>
<td>Packed SIMD</td>
<td>5 cycles/output</td>
</tr>
<tr>
<td></td>
<td>1.25 cycles/output</td>
</tr>
</tbody>
</table>
Scalability

Area:
95 MGE (18.5 mm², 70% layout density)

Power:
6.1 W (98% dynamic power, worst case)

\[ C = \frac{P \cdot N \cdot f}{B} \]

Heavier handlers
- Number of clusters
  - (number of cores/cluster is more challenging)
  - Simple cores (e.g., Snitch)
- Number of PsPIN units
- Frequency
- Accelerators (FPGAs?), specialized instructions

22nm -> 14 nm -> ~50% area reduction -> 20/30% frequency increase

Higher network bandwidth
- Need to rebalance
- Need to scale data path as well!
- Only packet buffer is affected!
A 4096-Core RISC-V Architecture

A sPINning ecosystem

**architecture**

- PsPIN (ISCA ‘21)
  - Power-efficient sPIN accelerator
- sPINIC
- Flare (SC ‘21)
  - Offloading all-reduce to sPIN switches

**use cases**

- sPIN (SC ’17, best paper fin.)
  - programming model
- sPIN-FS
- Zoo-sPINNER (MSc)
  - consensus on sPIN
- Quantization (MSc)
  - Allreduce and other collectives
- Erasure coding (MSc)
- Packet classification and pattern matching (BSc/MSc)
- Serverless sPIN (BSc)

**Simulations**

- Verilator support (Github)
  - Running PsPIN in an open-source cycle-accurate simulator
- SST support (in progress)
  - Large scale network simulations mixed with cycle accurate ones

**feedbacks**

- Network-accelerated datatypes (SC ‘19)
- Erasure coding (MSc)
Multitenancy
Network-accelerated non-contiguous memory transfers

Di Girolamo, Salvatore, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. “Network-accelerated non-contiguous memory transfers.” SC ’19
Network-accelerated non-contiguous memory transfers

Di Girolamo, Salvatore, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler.
“Network-accelerated non-contiguous memory transfers.” SC ’19
Network-accelerated non-contiguous memory transfers

Di Girolamo, Salvatore, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. "Network-accelerated non-contiguous memory transfers." SC ’19
Network-accelerated non-contiguous memory transfers

Di Girolamo, Salvatore, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. "Network-accelerated non-contiguous memory transfers." SC '19
Network-accelerated non-contiguous memory transfers

Di Girolamo, Salvatore, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. “Network-accelerated non-contiguous memory transfers.” SC ‘19
Network-accelerated non-contiguous memory transfers


Di Girolamo, Salvatore, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. “Network-accelerated non-contiguous memory transfers.” SC ’19
Network-accelerated non-contiguous memory transfers

Di Girolamo, Salvatore, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. “Network-accelerated non-contiguous memory transfers.” SC ’19

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

http://fourier.eng.hmc.edu/e161/lectures/fourier/node10.html
Network-accelerated non-contiguous memory transfers


Di Girolamo, Salvatore, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. "Network-accelerated non-contiguous memory transfers." SC ‘19
Network-accelerated non-contiguous memory transfers

Di Girolamo, Salvatore, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. “Network-accelerated non-contiguous memory transfers.” SC ‘19


http://fourier.eng.hmc.edu/e161/lectures/fourier/node10.html

MPI Derived Datatypes

https://specfem3d.readthedocs.io/en/latest/
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*
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
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
MPI Datatypes Processing

Gropp, W., et al., March. Improving the performance of MPI derived datatypes. *MPI’99*
MPI Datatypes Processing

Gropp, W., et al., March. Improving the performance of MPI derived datatypes. MPIDC’99
MPI Datatypes Processing

Gropp, W., et al., March. Improving the performance of MPI derived datatypes. MPIDC'99
MPI Datatypes Processing

Gropp, W., et al., March. Improving the performance of MPI derived datatypes. *MPIDC’99*
MPI Datatypes Processing

Gropp, W., et al., March. Improving the performance of MPI derived datatypes. *MPIDC’99*
Can we offload datatype processing?

Gropp, W., et al., March. Improving the performance of MPI derived datatypes. MPIDC’99
Specialized Handlers
Specialized Handlers

vector
Specialized Handlers

**spin_vec_t:**
- num_blocks: 3
- block_size: 2
- stride: 3
- base_type: int
Specialized Handlers

vector

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int
Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int
Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int
Specialized Handlers

NIC Memory

spin_vec_t:
- num_blocks: 3
- block_size: 2
- stride: 3
- base_type: int

Handler

vector
Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int
Specialized Handlers

NIC Memory

**spin_vec_t:**
- num_blocks: 3
- block_size: 2
- stride: 3
- base_type: int

```
vector

_handler 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 = (uint32_t)args->pkt_offset / ddt_descr->block_size;
    uint8_t *host_address = host_base_addr + 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;
}
```
Specialized Handlers

**NIC Memory**

- `spin_vec_t`:
  - num_blocks: 3
  - block_size: 2
  - stride: 3
  - base_type: int

**Handler**

Handler `vector_payload_handler` with handler arguments `args`:

```c
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->pkt_payload_ptr;
    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;
}
```

**Load DDT info**
Specialized Handlers

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int

_handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->descrm;
    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;
}
Specialized Handlers

```
vector_payload_handler(handler_args_t *args) {
  spin_vec_t *ddt_descr = (spin_vec_t *)args->ddt_descr;
  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;
  uint8_t *host_address = host_base_ptr + host_offset * stride;

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

  return SPIN_SUCCESS;
}
```

Load DDT info

Compute host memory destination address

DMA all contig. regions contained in the packet
Specialized Handlers

NIC Memory

**spin_vec_t:**
- num_blocks: 3
- block_size: 2
- stride: 3
- base_type: int

```
__handler vector_payload_handler(handler_args_t *args) {
  spin_vec_t *ddt_descr = (spin_vec_t *)args->ddt_descr;
  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;
}
```

Load DDT info
Compute host memory destination address
DMA all contig. regions contained in the packet
Specialized Handlers

Specialized

Load DDT info

Compute host memory destination address

DMA all contig. regions contained in the packet

return SPIN_SUCCESS;

_handler 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;
}
Specialized Handlers

```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;
}
```

Load DDT info

Compute host memory destination address

DMA all contig. regions contained in the packet

Specialized

Line rate

Throughput (Gbits)

Block Size (B)

Host Unpack

vector

indexed

struct

base_type: int
Specialized Handlers

```
void __handler(vector_payload_handler)(handler_args_t *args) {
    spin_vec_t *ddt_descriptor = (spin_vec_t *) args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descriptor->block_size;
    uint32_t stride = ddt_descriptor->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_descriptor->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;
}
```

Load DDT info

Compute host memory destination address

DMA all contig. regions contained in the packet

return SPIN_SUCCESS;
Specialized Handlers

```
vector

NIo Memory

spin_vec_t * ppkt_payload
num_blocks = packet_length / ddt_descr->block_size;
stride = ddt_descr->stride;
host_base_ptr = args->host_address;
host_offset = (args->pkt_offset / ddt_descr->block_size) * stride;
host_address = host_base_addr + 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;
```

Load DDT info
Compute host memory destination address
DMA all contig. regions contained in the packet

vector
indexed
struct
Specialized Handlers

vector
indexed
struct

Load DDT info
Compute host memory destination address
DMA all contig. regions contained in the packet

Need a different handlers for each possible derived datatype!
Specialized Handlers

vector

indexed

struct

Can we define a general handler to process any datatype?
MPI Types Library on sPIN

MPI Types Library on sPIN

```
Vector { #blocks: 3, blocklen: 2, stride: 3, basetype: [] }
```

Dataloops
MPI Types Library on sPIN

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

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: □ }

Dataloops

MPI Types Library on sPIN

MPI Types Library on sPIN

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

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: □ }

Dataloops

Index: Vector:

Segment

Host Memory

vector index

vector
MPI Types Library on sPIN

NIC Memory

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

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: □ }

Dataloops

Host Memory

Index: Vector:

Segment

vector

index

vector
MPI Types Library on sPIN

Dataloops

NIC Memory

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

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: } 

Segment

Host Memory

Index: Vector:

vector index vector

…

Index:

Vector:

Segment
MPI Types Library on sPIN

MPI Types Library on sPIN

NIC Memory

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

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

Dataloops

Segment

Index:

Vector:

Host Memory

index

vector

vector

index

vector

...
MPI Types Library on sPIN

MPI Types Library on sPIN

**NIC Memory**

- **Index**
  - #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: *

- **Vector**
  - #blocks: 3, blocklen: 2, stride: 3, basetype: }

**Host Memory**

- **Index**
- **Vector**
- **Segment**

---

MPI Types Library on sPIN

MPI Types Library on sPIN

**NIC Memory**

- **Index**
  - #blocks: 2, blocklen: 1,
  - offsets: {0, x}, basetype: *

- **Vector**
  - #blocks: 3, blocklen: 2,
  - stride: 3, basetype: }

**Host Memory**

- Index
- Vector

**Segment**

**Dataloops**

**Packet Scheduler**

- HPU 0
- HPU 1
- HPU 2
- HPU 3
MPI Types Library on sPIN

NIC Memory

Index
- #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: *

Vector
- #blocks: 3, blocklen: 2, stride: 3, basetype:  }

Dataloops

Host Memory

Index:

Vector:

Segment

Packet Scheduler

HPU 0

HPU 1

HPU 2

HPU 3
MPI Types Library on sPIN

**NIC Memory**
- **Index**
  - #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: *
- **Vector**
  - #blocks: 3, blocklen: 2, stride: 3, basetype: }

**Host Memory**
- Index
- Vector
- Segment

---

**Dataloops**

**Packet Scheduler**

- HPU 0
- HPU 1
- HPU 2
- HPU 3
MPI Types Library on sPIN

NIC Memory

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

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

Dataloops

Host Memory

Index:

Vector:

Segment

Packet Scheduler

HPU 0
HPU 1
HPU 2
HPU 3
MPI Types Library on sPIN

HPU-Local: each HPU has its own state

NIC Memory

Index: #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: *
Vector: #blocks: 3, blocklen: 2, stride: 3, basetype: 

Snapshot 0

Index: Vector:

Snapshot 1

Index: Vector:

Snapshot 2

Index: Vector:

Snapshot 3

Index: Vector:

Snapshot 4

Snapshot 5

Index: Vector:

Δt = 2

Packet Scheduler

HPU 0

HPU 1

HPU 2

HPU 3

Line rate

Specialized

HPU-Local

Host Unpack

Throughput (Gbps)

Block Size (B)
MPI Types Library on sPIN

HPU-Local: each HPU has its own state

RO-checkpoints: pre-computed checkpoints shared by multiple HPUs (read-only)
MPI Types Library on sPIN

NIC Memory

Index: [ ]
Vector: [ ]

Index: [ ]
Vector: [ ]

Index: [ ]
Vector: [ ]

Index: [ ]
Vector: [ ]

Δt = 2

HPU-Local: each HPU has its own state

RO-checkpoints: pre-computed checkpoints shared by multiple HPUs (read-only)

Packet Scheduler

HPU 0
HPU 1
HPU 2
HPU 3

Graph showing line rate, specialized, HPU-Local, RO Checkpoints, and Host Unpack with throughput in Gbps vs block size in bytes.
MPI Types Library on sPIN

**HPU-Local:** each HPU has its own state

**RO-checkpoints:** pre-computed checkpoints shared by multiple HPUs (read-only)

**RW-checkpoints:** pre-computed checkpoints shared by multiple HPUs (read/write, fine-grain synchronization)

---

**NIC Memory**

*Index:* 
*Vector:*

**Snapshot 0**

*Index:* 
*Vector:*

**Snapshot 1**

*Index:* 
*Vector:*

**Snapshot 2**

*Index:* 
*Vector:*

**Snapshot 3**

*Index:* 
*Vector:*

**Snapshot 4**

*Index:* 
*Vector:*

**Snapshot 5**

*Index:* 
*Vector:*

**Packet Scheduler**

**HPU 0**

**HPU 1**

**HPU 2**

**HPU 3**

---

\[ \Delta t = 2 \]
MPI Types Library on sPIN

**NIC Memory**

<table>
<thead>
<tr>
<th>Snapshot 0</th>
<th>Snapshot 1</th>
<th>Snapshot 2</th>
<th>Snapshot 3</th>
<th>Snapshot 4</th>
<th>Snapshot 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index:</td>
<td>Index:</td>
<td>Index:</td>
<td>Index:</td>
<td>Index:</td>
<td>Index:</td>
</tr>
<tr>
<td>Vector:</td>
<td>Vector:</td>
<td>Vector:</td>
<td>Vector:</td>
<td>Vector:</td>
<td>Vector:</td>
</tr>
</tbody>
</table>

**Δt = 2**

- **HPU-Local**: each HPU has its own state
- **RO-checkpoints**: pre-computed checkpoints shared by multiple HPUs (read-only)
- **RW-checkpoints**: pre-computed checkpoints shared by multiple HPUs (read/write, fine-grain synchronization)

**Packet Scheduler**

- HPU 0
- HPU 1
- HPU 2
- HPU 3

**Graph**

- Line rate
- Specialized
- RW Checkpoints
- HPU-Local
- RO Checkpoints
- Host Unpack
Real Applications DDTs

![Speedup Graph](SWALITE-X vector)
Real Applications DDTs

![Bar chart showing speedup for different applications](image)
Real Applications DDTs

![Speedup Chart]

<table>
<thead>
<tr>
<th>Speedup</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWAITE-X</td>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>vector</td>
<td>5</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

- **Red**: RW-CP
- **Orange**: Specialized
- **Purple**: Portals 4 (iovec)
Real Applications DDTs

![Speedup Graph]

- RW-CP
- Specialized
- Portals 4 (iowc)
Real Applications DDTs

![Speedup Diagram](chart.png)
Real Applications DDTs

![Speedup Graph](Image)

- SWALITE-X vector

- Speedup on applications:
  - a
  - b
  - c

- Categories:
  - RW-CP
  - Specialized
  - Portals 4 (iovec)
Real Applications DDTs

**Swallite-X Vectors**

- Speedup

<table>
<thead>
<tr>
<th>Speedup</th>
<th>RW-CP</th>
<th>Specialized</th>
<th>Portals 4 (lovec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Legend**
- RW-CP
- Specialized
- Portals 4 (lovec)
Real Applications DDTs

- SW4LITE-X
  - Vector
- SPEC-CM
  - Index-block
- NAS-MG
  - Vector
- SPEC-OC
  - Index-block

- Speedup
  - a, b, c
Real Applications DDTs

Speedup

- SW4LITE-X
  - vector
- SPEC-CM
  - Index Block
- NAS-MG
  - vector
- SPEC-OX
  - Index Block

Real Applications DDTs

- MILC
  - vector(vector)
- WRF-X
  - struct(subarray)
- FFT2D
  - contiguous(vector)
- NAS-LU
  - vector
- LAMMPS-F
  - index_block
- SW4LITE-Y
  - vector
- COMB
  - subarray
- WRF-Y
  - struct(subarray)
Real Applications DDTs

Checkpointing Overhead

75% of the analyzed DDTs amortized after 4 reuses

Data Movement

Up to 3.8x less moved data volume

Handler Analysis

Full app speedup (FFT2D)

75% of the analyzed DDTs amortized after 4 reuses
Support for Non-Contiguous Transfers
Support for Non-Contiguous Transfers

<table>
<thead>
<tr>
<th>ARMCI</th>
<th>CAF</th>
<th>Chapel</th>
<th>Portals 4</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHMEM</td>
<td>UPC</td>
<td>X10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Support for Non-Contiguous Transfers

<table>
<thead>
<tr>
<th>ARMCI</th>
<th>CAF</th>
<th>Chapel</th>
<th>Portals 4</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHMEM</td>
<td>UPC</td>
<td>X10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- I/O Vectors
- Strided transfers
- Compiler-Assisted Aggregation

Support for multiple strides (e.g., 3D faces)
Support for Non-Contiguous Transfers

<table>
<thead>
<tr>
<th>ARMCI</th>
<th>CAF</th>
<th>Chapel</th>
<th>Portals 4</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHMEM</td>
<td>UPC</td>
<td>X10</td>
<td></td>
<td>Derived Datatypes</td>
</tr>
<tr>
<td>I/O Vectors</td>
<td>Strided transfers</td>
<td>Compiler-Assisted Aggregation</td>
<td>Support for multiple strides (e.g., 3D faces)</td>
<td></td>
</tr>
</tbody>
</table>
Support for Non-Contiguous Transfers

- ARMCI
- CHMEM
- CAF
- UPC
- Chapel
- X10
- Portals 4

MPI

Derived Datatypes

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

ARMCI  CAF  Chapel  Portals 4  MPI
SHMEM  UPC  X10

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

Derived Datatypes
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
MPI Types Library on sPIN: HPU-Local

NIC Memory

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

Vector: #blocks: 3, blocklen: 2, stride: 3, basetype:

Index: Vector:
Segment 0

Index: Vector:
Segment 1

Index: Vector:
Segment 2

Index: Vector:
Segment 3

Packet Scheduler

HPU 0
HPU 1
HPU 2
HPU 3

Host Memory

index

vector

vector

...
MPI Types Library on sPIN: HPU-Local

NIC Memory

- Index: 
  - #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: *

- Vector: 
  - #blocks: 3, blocklen: 2, stride: x, basetype: }

Host Memory

- Index: 
  - Vector:

- Vector:

- Index: 
  - Vector:

- Vector:

Packet Scheduler

HPU 0

- HPU 1

- HPU 2

- HPU 3
MPI Types Library on sPIN: HPU-Local

NIC Memory

- Index: #blocks: 2, blocklen: 1, offsets: (0, x), basetype: *
- Vector: #blocks: 3, blocklen: 2, stride: 3, basetype: }

- Segment 0
- Segment 1
- Segment 2
- Segment 3

Packet Scheduler

- HPU 0
- HPU 1
- HPU 2
- HPU 3

Host Memory

- index
- vector

Segment 0
Segment 1
Segment 2
Segment 3
MPI Types Library on sPIN: HPU-Local

NIC Memory

- **Index**: #blocks: 2, blocklen: 1, offsets: (0, x), basetype: *

- **Vector**: #blocks: 3, blocklen: 2, stride: 3, basetype: 

Segment 0
- **Index**: Vector:
- **Vector**: 

Segment 1
- **Index**: Vector:
- **Vector**: 

Segment 2
- **Index**: Vector:
- **Vector**: 

Segment 3
- **Index**: Vector:
- **Vector**: 

Packet Scheduler

Host Memory

- **index** → 
  - **vector** → 
  - **vector** → 
  - **vector** → 
  - **...** → 
  - **vector** → 

HPU 0
- HPU 1
- HPU 2
- HPU 3
MPI Types Library on sPIN: HPU-Local

NIC Memory

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

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: [] }

Segment 0

Segment 1

Segment 2

Segment 3

Packet Scheduler

HPU 0

HPU 1

HPU 2

HPU 3

Host Memory

index

vector

vector

Segment 0

Segment 1

Segment 2

Segment 3
MPI Types Library on sPIN: HPU-Local

NIC Memory

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

Vector: #blocks: 3, blocklen: 2, stride: 3, basetype: # }

Segment 0

Segment 1

Segment 2

Segment 3

Packet Scheduler

HPU 0

HPU 1

HPU 2

HPU 3

Host Memory

index

vector

vector

...
MPI Types Library on sPIN: HPU-Local

NIC Memory

- Index: { #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }
- Vector: { #blocks: 3, blocklen: 2, stride: 3, basetype: }

Segment 0
- Index: Vector:

Segment 1
- Index: Vector:

Segment 2
- Index: Vector:

Segment 3
- Index: Vector:

Packet Scheduler

- HPU 0
- HPU 1
- HPU 2
- HPU 3

Host Memory

- Index
- Vector
- Vector
- Vector
- Vector
MPI Types Library on sPIN: HPU-Local

NIC Memory

Index: #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * 
Vector: #blocks: 3, blocklen: 2, stride: 3, basetype: 

Segment 0: Index: Vector:
Segment 1: Index: Vector:
Segment 2: Index: Vector:
Segment 3: Index: Vector:

Host Memory

Packet Scheduler

HPU 0
HPU 1
HPU 2
HPU 3

index
vector
vector
...
MPI Types Library on sPIN: HPU-Local

NIC Memory

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

Vector: #blocks: 3, blocklen: 2, stride: 3, basetype: }

Host Memory

Packet Scheduler

HPU 0  HPU 1  HPU 2  HPU 3

Index: Vector: Segment 0
Index: Vector: Segment 1
Index: Vector: Segment 2
Index: Vector: Segment 3

vector
index

vector
MPI Types Library on sPIN: HPU-Local

- NIC Memory
  - Index: 
    - #blocks: 2, blocklen: 1,
    - offsets: {0, x}, basetype: *
  - Vector: 
    - #blocks: 3, blocklen: 2,
    - stride: 3, basetype: }

- Host Memory
  - index
  - vector
  - Segment 0: Index
    - Vector:
  - Segment 1: Index
    - Vector:
  - Segment 2: Index
    - Vector:
  - Segment 3: Index
    - Vector:

Packet Scheduler

HPU 0    HPU 1    HPU 2    HPU 3
MPI Types Library on sPIN: HPU-Local

NIC Memory

Index: #blocks: 2, blocklen: 1, offsets: [0, x], basetype: *

Vector: #blocks: 3, blocklen: 2, stride: 3, basetype: δ

Segment 0
Segment 1
Segment 2
Segment 3

Packet Scheduler

Host Memory

Fast-forward overhead
MPI Types Library on sPIN: HPU-Local

NIC Memory

Index

Vector

Segment 0
Segment 1
Segment 2
Segment 3

Fast-forward overhead

Host Memory

Packet Scheduler

HPU 0
HPU 1
HPU 2
HPU 3

Line rate

Specialized

Host Unpack
MPI Types Library on sPIN: HPU-Local

NIC Memory

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

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

Vector { #blocks: 3, blocklen: 2, stride: 3, basetype: }

Segment 0
Segment 1
Segment 2
Segment 3

Fast-forward overhead

Host Memory

Packet Scheduler

HPU 0
HPU 1
HPU 2
HPU 3

Host Unpack

Line rate

Specialized

HPU-Local

Throughput (Gbps)

Block Size (B)
MPI Types Library on sPIN: Checkpoints

Snapshot the state on the host to bootstrap the handlers

Index:
Vector:
Segment
MPI Types Library on sPIN: Checkpoints

Snapshot the state on the host to bootstrap the handlers

$$\Delta t = 2$$
MPI Types Library on sPIN: Checkpoints

Snapshot the state on the host to bootstrap the handlers

\[ \Delta t = 2 \]
MPI Types Library on sPIN: Checkpoints

Snapshot the state on the host to bootstrap the handlers

\[ \Delta t = 2 \]
MPI Types Library on sPIN: Checkpoints

Snapshot the state on the host to bootstrap the handlers

\[ \Delta t = 2 \]
MPI Types Library on sPIN: Checkpoints

Snapshot the state on the host to bootstrap the handlers

\[ \Delta t = 2 \]
MPI Types Library on sPIN: Checkpoints

Snapshot the state on the host to bootstrap the handlers

\[ \Delta t = 2 \]
MPI Types Library on sPIN: Checkpoints

Snapshot the state on the host to bootstrap the handlers

Δt = 2
MPI Types Library on sPIN: Checkpoints

Snapshot the state on the host to bootstrap the handlers

\[ \Delta t = 2 \]
MPI Types Library on sPIN: Checkpoints

Snapshot the state on the host to bootstrap the handlers

\[ \Delta t = 2 \]
MPI Types Library on sPIN: Checkpoints

NIC Memory

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

Vector: #blocks: 3, blocklen: 2, stride: 3, basetype: }

Snapshot 0:

Index: Vector:

Snapshot 1:

Index: Vector:

Snapshot 2:

Index: Vector:

Snapshot 3:

Index: Vector:

Snapshot 4:

Snapshot 5:

Index: Vector:

Delta t = 2

Packet Scheduler

HPU 0
HPU 1
HPU 2
HPU 3

Host Memory

index

vector

vector

Snapshot 1
Snapshot 2
Snapshot 3
Snapshot 4
Snapshot 5

HPU 0 - HPU 3
Packet Scheduler

Specialized
HPU-Local

Host Unpack

Line rate

Throughput (Gbps)

Block Size (B)

0 4 16 32 64 128 256 512 1K 2K 4K 8K 16K

Specialized
HPU-Local

Host Unpack
MPI Types Library on sPIN: Checkpoints

NIC Memory

<table>
<thead>
<tr>
<th>Snapshot 0</th>
<th>Snapshot 1</th>
<th>Snapshot 2</th>
<th>Snapshot 3</th>
<th>Snapshot 4</th>
<th>Snapshot 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index:</td>
<td>Index:</td>
<td>Index:</td>
<td>Index:</td>
<td>Index:</td>
<td>Index:</td>
</tr>
<tr>
<td>Vector:</td>
<td>Vector:</td>
<td>Vector:</td>
<td>Vector:</td>
<td>Vector:</td>
<td>Vector:</td>
</tr>
</tbody>
</table>

\[ \Delta t = 2 \]

Host Memory

- Index
- Vector
- vector

Packet Scheduler

- HPU 0
- HPU 1
- HPU 2
- HPU 3

Graph:

- Line rate
- Throughput (Gbps)
- Specialized
- HPU-Local
- Host Unpack

Block Size (B)

Throughput (Gbps)
MPI Types Library on sPIN: Checkpoints

NIC Memory

Snapshot 0
Index: Vector:

Snapshot 1
Index: Vector:

Snapshot 2
Index: Vector:

Snapshot 3
Index: Vector:

Snapshot 4
Index: Vector:

Snapshot 5
Index: Vector:

\[ \Delta t = 2 \]

Host Memory

Index

vector

vector

vector

vector

Packet Scheduler

HPU 0

HPU 1

HPU 2

HPU 3

Specialized

HPU-Local

Host Unpack

Line rate

Throughput (Gbps)

Block Size (B)
MPI Types Library on sPIN: Checkpoints

NIC Memory

<table>
<thead>
<tr>
<th>Snapshot 0</th>
<th>Snapshot 1</th>
<th>Snapshot 2</th>
<th>Snapshot 3</th>
<th>Snapshot 4</th>
<th>Snapshot 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index:</td>
<td>Vector:</td>
<td>Index:</td>
<td>Vector:</td>
<td>Index:</td>
<td>Vector:</td>
</tr>
</tbody>
</table>

Δt = 2

Host Memory

Packet Scheduler

HPU 0  HPU 1  HPU 2  HPU 3

Specialized  HPU-Local  Host Unpack
MPI Types Library on sPIN: Checkpoints

NIC Memory

Index: Vector:

Snapshot 0

Index: Vector:

Snapshot 1

Index: Vector:

Snapshot 2

Index: Vector:

Snapshot 3

Index: Vector:

Snapshot 4

Index: Vector:

Snapshot 5

Δt = 2

Host Memory

Snapshot 0

Snapshot 1

Snapshot 2

Snapshot 3

Snapshot 4

Snapshot 5

Packet Scheduler

HPU 0

HPU 1

HPU 2

HPU 3

Line rate

Specialized

HPU-Local

Host Unpack

Throughput (Gbps)

Block Size (B)

4

16

32

64

128

256

512

1K

2K

4K

8K

16K
MPI Types Library on sPIN: Read-Only Checkpoints

**NIC Memory**

- **Index:**
  - Block: 2
  - Blocklen: 1
  - Offsets: {0, x}
  - Basetype: *

- **Vector:**
  - Block: 3
  - Blocklen: 2
  - Stride: 3
  - Basetype: 

**Host Memory**

- **Index:**
- **Vector:**

**Packet Scheduler**

- HPU 0
- HPU 1
- HPU 2
- HPU 3

\[ \Delta t = 2 \]
MPI Types Library on sPIN: Read-Only Checkpoints

NIC Memory

Snapshot 0

Index: 
Vector: 

Snapshot 1

Index: 
Vector: 

Snapshot 2

Index: 
Vector: 

Snapshot 3

Index: 
Vector: 

Snapshot 4

Index: 
Vector: 

Snapshot 5

Index: 
Vector: 

Δt = 2

Host Memory

Packet Scheduler

HPU 0  HPU 1  HPU 2  HPU 3

Specialized

HPU-Local

Host Unpack

Line rate

Throughput (Gbps)

Block Size (B)
MPI Types Library on sPIN: Read-Only Checkpoints

NIC Memory

Snapshot 0
Index: Vector:

Snapshot 1
Index: Vector:

Snapshot 2
Index: Vector:

Snapshot 3
Index: Vector:

Snapshot 4
Index: Vector:

Snapshot 5
Index: Vector:

Δt = 2

Vector

Host Memory

Index

vector

vector

Packet Scheduler

HPU 0
Index: Vector:

HPU 1
Index: Vector:

HPU 2
Index: Vector:

HPU 3
Index: Vector:

Throughput (Gbps)

Block Size (B)

Line rate

Specialized

HPU-Local

Host Unpack
MPI Types Library on sPIN: Read-Only Checkpoints

NIC Memory

Index: Vector:
Snapshot 0

Index: Vector:
Snapshot 1

Index: Vector:
Snapshot 2

Index: Vector:
Snapshot 3

Index: Vector:
Snapshot 4

Snapshot 5

Vector: #blocks: 3, blocklen: 2, stride: 3, basetype: 

Δt = 2

Host Memory

index

t

vector

vector

snapshot

Packet Scheduler

HPU 0

Index: Vector:

HPU 1

Index: Vector:

HPU 2

Index: Vector:

HPU 3

Index: Vector:

Specialized

HPU-Local

RO Checkpoints

Host Unpack

Throughput (Gbit/s)

Block Size (B)

Line rate
MPI Types Library on sPIN: Read-Only Checkpoints

NIC Memory

Snapshot 0
- Index:
- Vector:

Δt = 2

Snapshot 1
- Index:
- Vector:

Snapshot 2
- Index:
- Vector:

Snapshot 3
- Index:
- Vector:

Snapshot 4
- Index:
- Vector:

Snapshot 5
- Index:
- Vector:

Packet Scheduler

HPU 0
- Index:
- Vector:

HPU 1
- Index:
- Vector:

HPU 2
- Index:
- Vector:

HPU 3
- Index:
- Vector:

Host Memory

Index → vector → vector

Snapshot 0
Snapshot 1
Snapshot 2
Snapshot 3
Snapshot 4
Snapshot 5

Line rate

Specialized
HPU-Local
RO Checkpoints
Host Unpack

Δt = 2

Checkpoint copying overhead
MPI Types Library on sPIN: Read-Write Checkpoints

NIC Memory

\[ \Delta t = 2 \]

Index:
Vector:

Vector:
Index:

Index:
Vector:

Index:
Vector:

Host Memory

Packet Scheduler

HPU 0    HPU 1    HPU 2    HPU 3

Line rate

Throughput (Gbps)

Block Size (B)

Specialized

HPU-Local

RO Checkpoints

Host Unpack
MPI Types Library on sPIN: Read-Write Checkpoints

NIC Memory

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:

\[ \Delta t = 2 \]

Host Memory

Packet Scheduler

HPU 0

HPU 1

HPU 2

HPU 3

Specialized

HPU-Local

RO Checkpoints

Host Unpack

throughput (Gbps)

Line rate

Block Size (B)

0 50 100 150 200

4 16 32 64 128 256 512 1k 2k 4k 8k 16k
MPI Types Library on sPIN: Read-Write Checkpoints

**NIC Memory**

- **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**

**Host Memory**

- **Index**
  - vector
- **vector**

**Packet Scheduler**

- HPU 0
- HPU 1
- HPU 2
- HPU 3

**Line rate**

- Specialized
- HPU-Local
- RO Checkpoints
- Host Unpack

**Throughput (Gbits)**

- 0
- 50
- 100
- 150
- 200

**Block Size (B)**

- 4
- 16
- 32
- 64
- 128
- 256
- 512
- 1K
- 2K
- 4K
- 8K
- 16K
MPI Types Library on sPIN: Read-Write Checkpoints

NIC Memory

Index: 
Vector: 

V-HPU 1
Index: 
Vector: 

V-HPU 2
Index: 
Vector: 

V-HPU 3
Index: 
Vector: 

V-HPU 4
Index: 
Vector: 

Δt = 2

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: } 

V-HPU 5
Index: 
Vector: 

Host Memory

index

vector

Vector{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * } 

Packet Scheduler

HPU 0

V-HPU 0

HPU 1

HPU 2

HPU 3

Index:
Vector:

Host Unpack

RO Checkpoints

Specialized

HPU-Local

Throughput (Gbps)

Line rate

Block Size (B)

0 50 100 150 200

4 16 32 64 128 256 512 1K 2K 4K 8K 16K

Host Unpack
MPI Types Library on sPIN: Read-Write Checkpoints

NIC Memory

\[ \Delta t = 2 \]

Host Memory

Packet Scheduler

HPU 0
HPU 1
HPU 2
HPU 3
MPI Types Library on sPIN: Read-Write Checkpoints

NIC Memory

\[ \Delta t = 2 \]

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: 

Host Memory

Packet Scheduler

HPU 0
V-HPU 0

HPU 1

HPU 2

HPU 3

Vector

Index

Vector

Index

Vector

Index

Vector

INDEX

\#blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * 

\#blocks: 3, blocklen: 2, stride: 3, basetype: 

\#blocks: 3, blocklen: 2, stride: 3, basetype: 

\#blocks: 3, blocklen: 2, stride: 3, basetype: 

Line rate

Specialized

HPU-Local

RO Checkpoints

Host Unpack

Throughput (Gbps)

Block Size (B)
MPI Types Library on sPIN: Read-Write Checkpoints

NIC Memory

\[ \Delta t = 2 \]

Host Memory

Packet Scheduler

HPU 0

HPU 1

HPU 2

HPU 3
MPI Types Library on sPIN: Read-Write Checkpoints

NIC Memory

\[ \Delta t = 2 \]

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

Vector: #blocks: 3, blocklen: 2, stride: 3, basetype: 

V-HPU 2

Index: Vector:

V-HPU 3

Index: Vector:

V-HPU 4

Index: Vector:

V-HPU 5

Index: Vector:

Packet Scheduler

HPU 0

V-HPU 0

HPU 1

V-HPU 1

HPU 2

HPU 3

Host Memory

index

vector

vector

Packet Scheduler

HPU 0

V-HPU 0

HPU 1

V-HPU 1

HPU 2

HPU 3

Line rate

Specialized

RW Checkpoints

HPU-Local

RO Checkpoints

Host Unpack

Throughput (Gbps)

Block Size (B)
Checkpoint Interval Selection

Network

HPU 0
HPU 1
HPU 2

time
Checkpoint Interval Selection

Network

HPU 0
HPU 1
HPU 2

time
Checkpoint Interval Selection

Network

HPU 0
HPU 1
HPU 2

time
Checkpoint Interval Selection

Network

HPU 0

HPU 1

HPU 2

time
Checkpoint Interval Selection

Network

HPU 0

HPU 1

HPU 2

Buffering

time
Checkpoint Interval Selection

Network

HPU 0

HPU 1

HPU 2

Buffering

time
Checkpoint Interval Selection

Network

HPU 0

HPU 1

HPU 2

Buffering

time
Checkpoint Interval Selection

Network

HPU 0

HPU 1

HPU 2

Buffering

(time)
Checkpoint Interval Selection

Network

Buffering

HPU 0

HPU 1

HPU 2

time
Checkpoint Interval Selection

Network

HPU 0

HPU 1

HPU 2

Buffering

time
Checkpoint Interval Selection

Network

HPU 0

HPU 1

HPU 2

Buffering

---

time
Checkpoint Interval Selection

Network

HPU 0

HPU 1

HPU 2

Buffering

time
Checkpoint Interval Selection

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

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

1. Limit the impact of the scheduling overhead
Checkpoint Interval Selection

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

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

Network

HPU 0

HPU 1

HPU 2

Buffering

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

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

- **SW4LTE-X**
  - Speedup:
    - Vector: 43.8, 43.3, 43.7
    - Struct (subarray): 13.28, 7.04, 3.83
    - Contiguous regions/packet: 581.2, 305.2, 167.1

- **WRF-X**
  - Speedup:
    - Struct (subarray): 171.3, 0.99, 0.59, 0.36
    - Contiguous regions/packet: 169.6, 100.2, 61.7

- **NAS-MG**
  - Speedup:
    - Vector: 8.5, 2.5, 1.5, 1.5
    - Contiguous regions/packet: 0.47, 25.6, 171.56, 684.45

- **SPEC-OC**
  - Speedup:
    - Index: 512, 512, 512, 512
    - Block: 12.6, 7.4, 3.4, 1.9

**Baseline (ms) and Message size (KiB):**
- Contiguous regions/packet
- Portals 4 (lovec)
Real Applications DDTs

<table>
<thead>
<tr>
<th>SW4LITE-X</th>
<th>WRF-X</th>
<th>NAS-MG</th>
<th>SPEC-OC</th>
</tr>
</thead>
<tbody>
<tr>
<td>vector</td>
<td>struct(subarray)</td>
<td>vector</td>
<td>index_block</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>43.8</td>
<td>43.3</td>
<td>43.7</td>
<td>171.3</td>
</tr>
<tr>
<td>13.28</td>
<td>7.04</td>
<td>3.83</td>
<td>169.6</td>
</tr>
<tr>
<td>581.2</td>
<td>305.2</td>
<td>167.1</td>
<td>4</td>
</tr>
</tbody>
</table>

Contiguous regions/packet
Baseline (ms)
Message size (KiB)

- RW–CP
- Specialized
- Portals 4 (lovec)
Real Applications DDTs

<table>
<thead>
<tr>
<th>Speedup</th>
<th>Contiguous regions/packet</th>
<th>Baseline (ms)</th>
<th>Message size (KiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW4LITE-X</td>
<td>vector</td>
<td>43.8, 43.3, 43.7</td>
<td>43.28, 7.04, 3.83</td>
</tr>
<tr>
<td>WRF-X</td>
<td>struct(subarray)</td>
<td>171.3, 0.99, 0.59</td>
<td>171.7, 169.6, 100.2, 61.7</td>
</tr>
<tr>
<td>NAS-MG</td>
<td>vector</td>
<td>8.5, 2.5, 1.5, 1.5</td>
<td>4, 64, 256, 1024</td>
</tr>
<tr>
<td>SPEC-OC</td>
<td>index_block</td>
<td>512, 512, 512, 512</td>
<td>0.02, 0.01, 0.01, 0.01</td>
</tr>
</tbody>
</table>

- RW-CP
- Specialized
- Portals 4 (lovec)

Contiguous regions/packet
Real Applications DDTs

<table>
<thead>
<tr>
<th></th>
<th>SW4LITE-X</th>
<th>WRF-X</th>
<th>NAS-MG</th>
<th>SPEC-OC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup</td>
<td>vector</td>
<td>struct(subarray)</td>
<td>vector</td>
<td>index_block</td>
</tr>
<tr>
<td>a</td>
<td>43.8</td>
<td>171.3</td>
<td>8.5</td>
<td>512</td>
</tr>
<tr>
<td>b</td>
<td>43.3</td>
<td>0.99</td>
<td>2.5</td>
<td>512</td>
</tr>
<tr>
<td>c</td>
<td>43.7</td>
<td>0.59</td>
<td>1.5</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td>13.28</td>
<td>0.36</td>
<td>1.5</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td>305.2</td>
<td>169.6</td>
<td>0.47</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td>167.1</td>
<td>100.2</td>
<td>0.02</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>25.6</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>171.56</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>684.45</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Contiguous regions/packet
Baseline (ms)
Message size (KiB)
Real Applications DDTs

- SW4LITE-X: 43.8, 43.3, 43.7
  - Vector
  - Speedup

- WRF-X: 171.3, 171.3, 171.7
  - Struct(subarray)

- NAS-MG: 8.5, 2.5, 1.5, 1.5
  - Vector

- SPEC-OC: 512, 512, 512, 512
  - Index_block

Contiguous regions/packet
Baseline (ms)
Message size (KiB)

- RW–CP
- Specialized
- Portals 4 (lovec)
Real Applications DDTs

- **SW4LITE-X**
  - Speedup: 43.8, 43.3, 43.7
  - Contiguous regions/packet: 13.28, 7.04, 3.83
  - Baseline (ms): 581.2, 305.2, 167.1

- **WRF-X**
  - Speedup: 171.3, 171.3, 171.7
  - Contiguous regions/packet: 0.99, 0.59, 0.36
  - Baseline (ms): 169.6, 100.2, 61.7

- **NAS-MG**
  - Speedup: 8.5, 2.5, 1.5
  - Contiguous regions/packet: 0.47, 25.6, 171.56
  - Baseline (ms): 4, 64, 256, 1024

- **SPEC-OC**
  - Speedup: 512, 512, 512, 512
  - Contiguous regions/packet: 0.02, 0.01, 0.01
  - Baseline (ms): 512, 512, 512, 512

Message size (KiB)
Real Applications DDTs

<table>
<thead>
<tr>
<th>SW4LTE-X</th>
<th>WRF-X</th>
<th>NAS-MG</th>
<th>SPEC-OC</th>
</tr>
</thead>
<tbody>
<tr>
<td>vector</td>
<td>struct(subarray)</td>
<td>vector</td>
<td>index_block</td>
</tr>
<tr>
<td>Speedup</td>
<td>Speedup</td>
<td>Speedup</td>
<td>Speedup</td>
</tr>
<tr>
<td>43.8</td>
<td>171.3</td>
<td>8.5</td>
<td>512</td>
</tr>
<tr>
<td>43.3</td>
<td>171.3</td>
<td>2.5</td>
<td>512</td>
</tr>
<tr>
<td>43.7</td>
<td>171.7</td>
<td>1.5</td>
<td>512</td>
</tr>
<tr>
<td>13.28</td>
<td>0.99</td>
<td>171.56</td>
<td>0.02</td>
</tr>
<tr>
<td>7.04</td>
<td>0.59</td>
<td>684.45</td>
<td>0.01</td>
</tr>
<tr>
<td>3.83</td>
<td>0.36</td>
<td>4</td>
<td>0.01</td>
</tr>
<tr>
<td>581.2</td>
<td>169.6</td>
<td>512</td>
<td>12.6</td>
</tr>
<tr>
<td>305.2</td>
<td>100.2</td>
<td>512</td>
<td>7.4</td>
</tr>
<tr>
<td>167.1</td>
<td>61.7</td>
<td>1024</td>
<td>3.4</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>Contiguous regions/packet</td>
<td>Baseline (ms)</td>
<td>Message size (KiB)</td>
<td></td>
</tr>
<tr>
<td>RW–CP</td>
<td>Specialized</td>
<td>Portals 4 (lovec)</td>
<td></td>
</tr>
</tbody>
</table>
Real Applications DDTs

Contiguous regions/packet
Baseline (ms)
Message size (KiB)

<table>
<thead>
<tr>
<th>Contiguous regions/packet</th>
<th>Baseline (ms)</th>
<th>Message size (KiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW-CP</td>
<td>512</td>
<td>12.6</td>
</tr>
<tr>
<td>Specialized</td>
<td>512</td>
<td>7.4</td>
</tr>
<tr>
<td>Portals (ivec)</td>
<td>512</td>
<td>3.4</td>
</tr>
<tr>
<td>Sorting</td>
<td>512</td>
<td>1.9</td>
</tr>
</tbody>
</table>

### SW4LITE-X
- **vector**
  - a: 43.8, 13.28, 581.2
  - b: 43.3, 7.04, 305.2
  - c: 43.7, 3.83, 167.1

### WRF-X
- **struct(subarray)**
  - a: 171.3, 0.99, 169.6
  - b: 171.3, 0.59, 102.2
  - c: 171.7, 0.36, 61.7

### NAS-MG
- **vector**
  - d: 8.5, 0.47, 4
  - c: 2.5, 25.6
  - b: 1.5, 171.56
  - a: 1.5, 648.45

### SPEC-OC
- **index block**
  - d: 512, 0.02
  - c: 512, 0.01
  - b: 512, 0.01
  - a: 512, 0

### SPEC-CM
- **index block**
  - d: 171.6, 1.01, 171.6
  - c: 171.8, 0.6, 171.6
  - b: 171.6, 0.31
  - a: 171.9, 0.15

### MILC
- **vector**
  - c: 173.5
  - b: 103.8
  - a: 53.9

### SW4LITE-Y
- **vector**
  - d: 122.9, 1.6
  - c: 87.73, 1.6
  - b: 65.79, 1.6
  - a: 191.4, 1.6

### NAS-LU
- **vector**
  - c: 1.6
  - b: 256, 1.6
  - a: 114.08, 1.6

### WRF-Y
- **struct(subarray)**
  - d: 1.22
  - c: 0.01
  - b: 114.08
  - a: 1.22

### FFT2D
- **contiguous(vector)**
  - c: 18.09, 32
  - b: 32, 72
  - a: 2316, 4096

### LAMMPS-F
- **index block**
  - c: 140.2
  - b: 103.9

### COMB
- **subarray**
  - c: 256
  - b: 256
  - a: 61.7

### Real Applications DDTs
- Speedup
  - a: 43.8, 13.28, 581.2
  - b: 43.3, 7.04, 305.2
  - c: 43.7, 3.83, 167.1
Cray Slingshot Simulator
Cray Slingshot Simulator
Cray Slingshot Simulator

32 Cortex A15 @800 MHz, single-cycle access memory
Network-accelerated DDT processing strategies

Diagram showing the process of Pack, Unpack, Streaming Puts, sPIN, sPIN-Out, and sPIN operations involving CPU and NICs.
DFS handlers

```c
void header_handler(spin_task_t* task, pkt_t* pkt) {
    dfs_state_t* state = (dfs_state_t*) task->mem;
    bool accept_next_pkts = DFS_request_init(state, pkt);
    int req_idx = task->flow_id;
    state->req_table[req_idx].greq_id = pkt->dfs.greq_id;
    state->req_table[req_idx].accept = accept_next_pkts;
}

void payload_handler(spin_task_t* task, pkt_t* pkt) {
    dfs_state_t* state = (dfs_state_t*) task->mem;
    int req_idx = task->flow_id;
    if (state->req_table[req_idx].accept)
        DFS_request_process_pkt(state, pkt);
}

void tail_handler(spin_task_t* task, pkt_t* pkt) {
    dfs_state_t* state = (dfs_state_t*) task->mem;
    int req_idx = task->flow_id;
    if (state->req_table[req_idx].accept)
        DFS_request_fini(state, pkt);
}
```
Request validation
Erasure coding
Simple write latency

Figure 8: Write latencies with different protocols and write sizes. RDMA writes are reported as baseline and do not implement request validation.
Writes with erasure coding

Figure 12: Encoding latency throughput.

Figure 13: Left: handler running times for RS(3,2) and RS(6,3). Right: HPUs needed to sustain 400 Gbit/s and 200 Gbit/s (with 1 KiB packets) vs average handler duration.
Writes with replication

Figure 10: Left: 512 KiB write latency for different replication factors ($k$). Right: goodput of sPIN-accelerated writes.

Figure 11: Left: 512 KiB write latency for different replication factors factor. Right: replicated writes goodput.