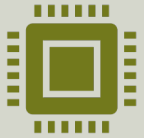
TIMO SCHNEIDER, PENGCHENG XU, TORSTEN HOEFLER

FPsPIN: An FPGA-based Open-Hardware Research Platform for Processing in the Network

Offloading light-weight data processing for HPC



High data rates require offloading data processing to NICs

CPU needs ~10 ns to read a 64 B cache line

At 400 Gbps: 64 B = 1.28 ns

Existing NIC offloading solutions are vendor and hardware dependent

Limits research and production adoption

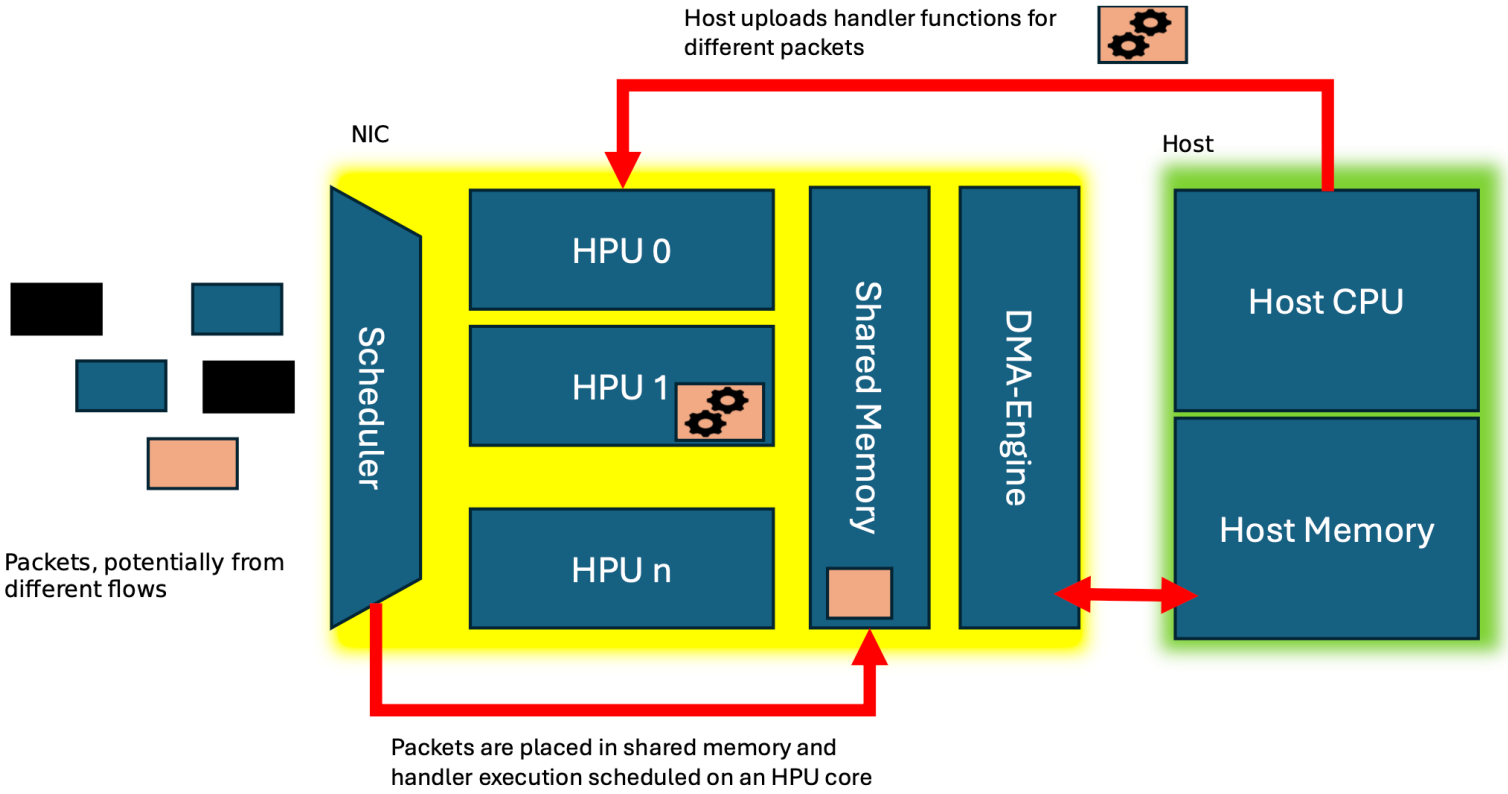
sPIN: Streaming Processing In the Network

Hardware-agnostic abstract machine model for in-network processing

sPIN: streaming Processing-In-the-Network

sPIN: High-performance streaming Processing In the Network, T. Hoefler et. al., SC'17.

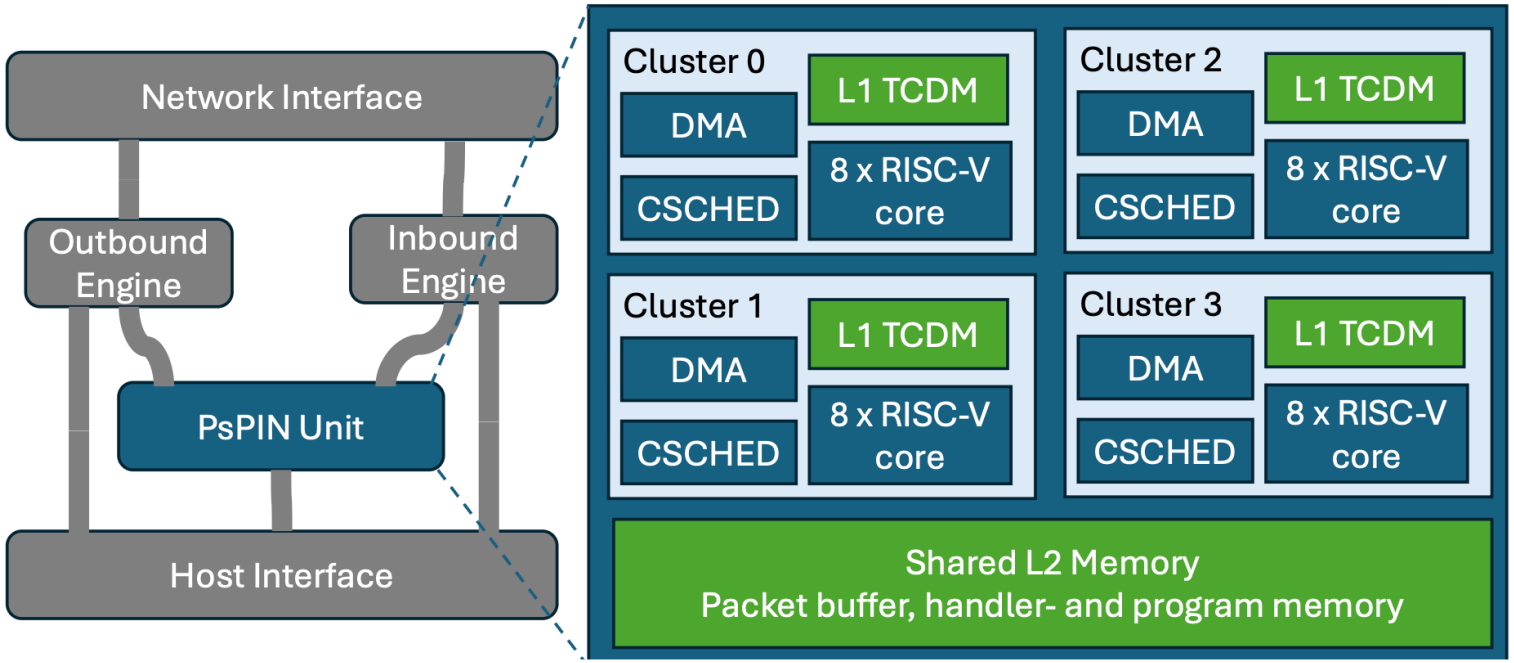
- On-path smart NICs
  - *Handler Processing Units (HPU)*
- Defines programming model of packet handlers
  - *Header, packet, tail handlers*
  - *Packet data in fast local mem.*
  - *Host mem. through DMA*
- Abstract machine model
  - *Doesn't define HPU ISA*
  - *Doesn't define network interface*



PsPIN: implement sPIN with PULP RISC-V cores

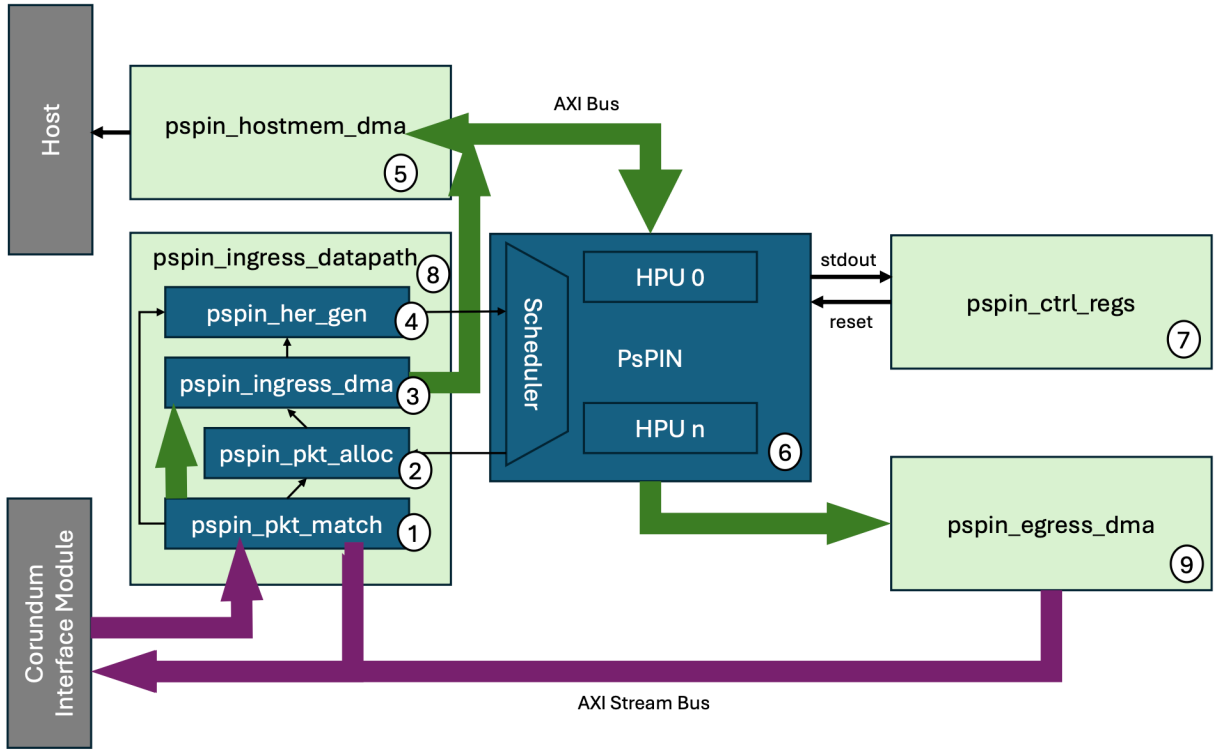
A RISC-V in-network accelerator for flexible high-performance low-power packet processing, S. Di Girolamo, et. al., ISCA'21.

- RTL implementation of sPIN
  - *HPU implemented as RISC-V cores*
  - *Handler Execution Requests (HER)*
  - *Global and local schedulers*
  - *Hierarchical memory*
- Key components unimplemented (simulation only)
  - *Network interface*
  - *Host interface*



This work: FPsPIN, an end-to-end sPIN system on FPGAs

- Integrate PsPIN with a NIC and host
  - *Corundum [1]: FPGA-based Ethernet NIC*
  - *NIC offers host DMA access*
- Implement HW adaptor modules:
  - *Match incoming packets*
  - *Generate HER for PsPIN*
  - *Stream packet data with AXI-Stream*
  - *Bridge host DMA port to PsPIN*
  - *Redirect debug I/O through registers*



[1]: Corundum: An Open-Source 100-Gbps NIC, A. Forencich, et.al., FCCM'20.

FPsPIN handler example: UDP ping-pong server

- Packet handlers are C functions
  - *Handler gets pointer to packet as argument*
  - *UDP echo is stateless: no need for header/tail handlers*
- C structs for parsing protocol headers
- `spin_*` API functions to send response

```c
#include <pspin.h>
#include <handler.h>
#include <packets.h>
#include <spin_dma.h>

__handler__ void ping_ph(handler_args_t *args) {
  task_t *task = args->task;
  pkt_hdr_t *hdrs = (pkt_hdr_t *)(task->pkt_mem);
  uint16_t pkt_len = args->task->pkt_mem_size;

  // swap ETH src and dst MAC address
  mac_addr_t src_mac = hdrs->eth_hdr.src;
  hdrs->eth_hdr.src = hdrs->eth_hdr.dest;
  hdrs->eth_hdr.dest = src_mac;

  // swap src and dst address in IP header
  uint32_t src_id = hdrs->ip_hdr.source_id;
  hdrs->ip_hdr.source_id = hdrs->ip_hdr.dest_id;
  hdrs->ip_hdr.dest_id = src_id;

  // swap src and dst port in UDP header
  uint16_t src_port = hdrs->udp_hdr.src_port;
  hdrs->udp_hdr.src_port = hdrs->udp_hdr.dst_port;
  hdrs->udp_hdr.dst_port = src_port;

  spin_cmd_t put; //sPIN commands are non-blocking
  spin_send_packet(task->pkt_mem, pkt_len, &put);
  spin_cmd_wait(put); // wait for send to complete
}
```

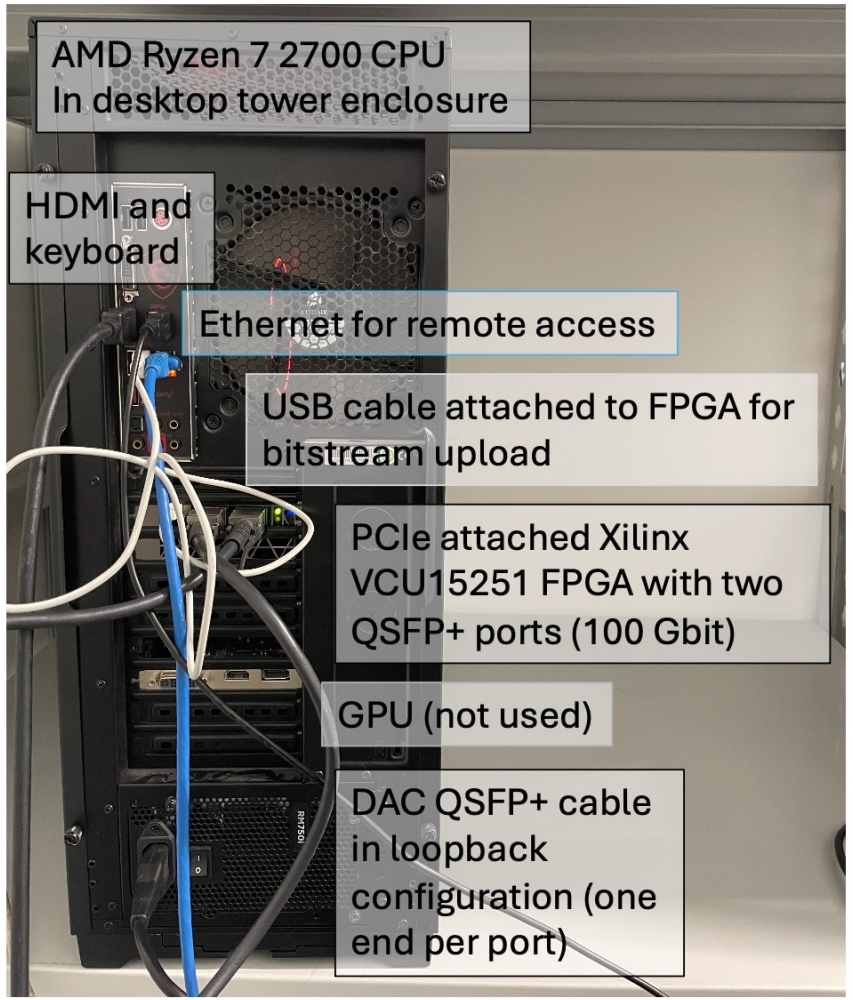
Loading a handler onto the FPsPIN NIC

- Host-side API functions to install handler
  - *Program matching engine for this handler*
  - *Write handler code to HPU memory*
- Set up host DMA page mappings
  - *Allows sPIN handlers to interact with the rest of the networked host application*

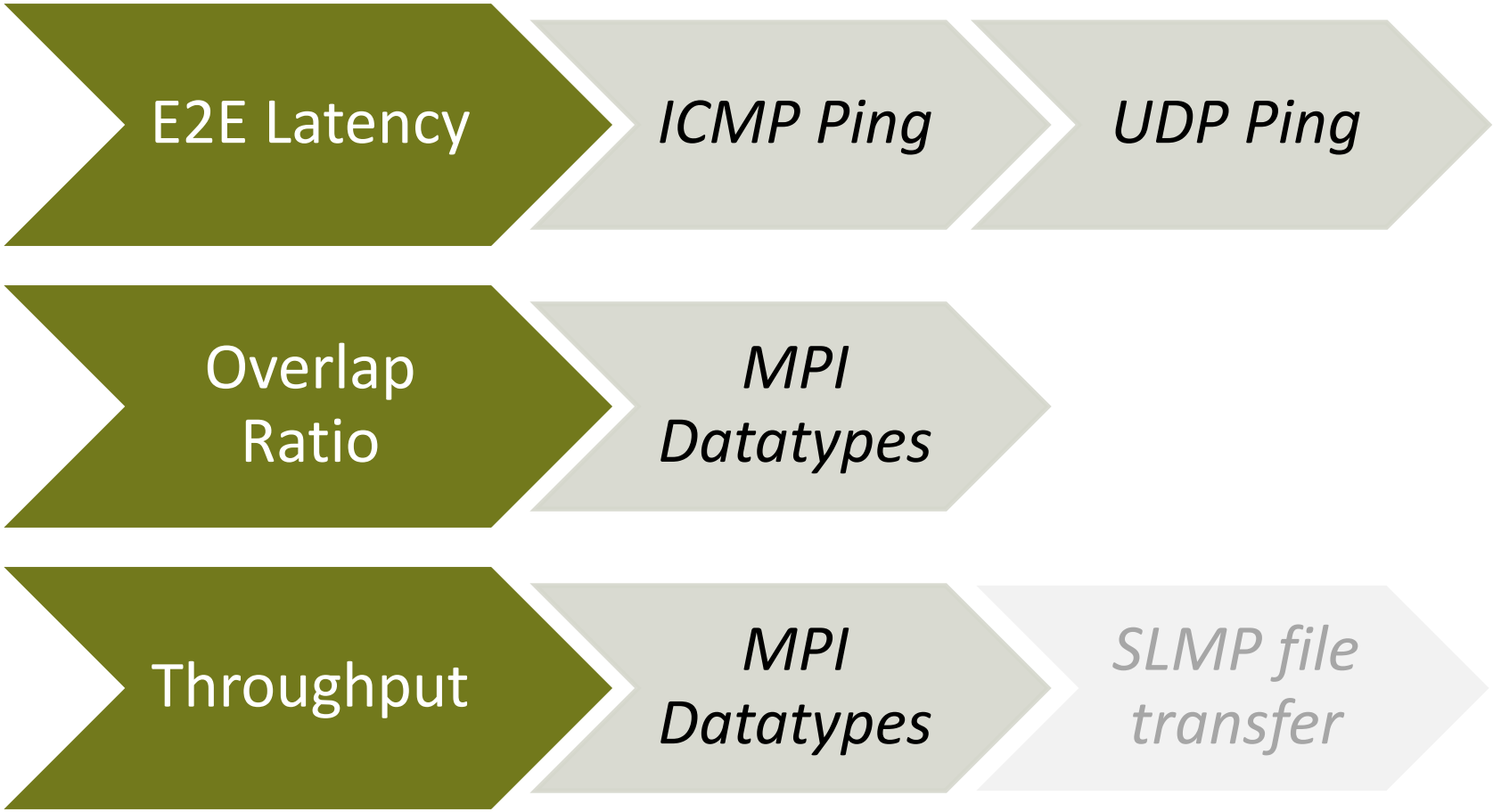
```c
#include "fpspin/fpspin.h"
#include <unistd.h>

void ruleset_icmp_echo(fpspin_ruleset_t *rs) {
  *rs = (fpspin_ruleset_t){
    .mode = FPSPIN_MODE_AND,
    .r = {
       FPSPIN_RULE_IP,
       FPSPIN_RULE_IP_PROTO(1), // ICMP
       ((struct fpspin_rule){
         .idx = 8,
         .mask = 0xff00,
         .start = 0x0800,
         .end = 0x0800}), // ICMP Echo-Request
       FPSPIN_RULE_FALSE,  // no EOM
    }, };
}

int main(int argc, char *argv[]) {
  fpspin_ctx_t ctx;
  fpspin_ruleset_t rs;
  ruleset_icmp_echo(&rs);
  fpspin_init(&ctx, "/dev/pspin0", "build/pingpong",
    0/*dst ctxt*/, &rs, 1/*dst ruleset*/,
    FPSPIN_HOSTDMA_PAGES_DEFAULT);
  while (1) {sleep(10);}
  fpspin_exit(&ctx);
}
```

Evaluation platform

- AMD Ryzen 7 2700 CPU
- Xilinx VCU1525 board
  - *VU9P FPGA*
  - *Corundum @250 MHz, PsPIN @ 40 MHz*
- External Ethernet loopback via DAC cable

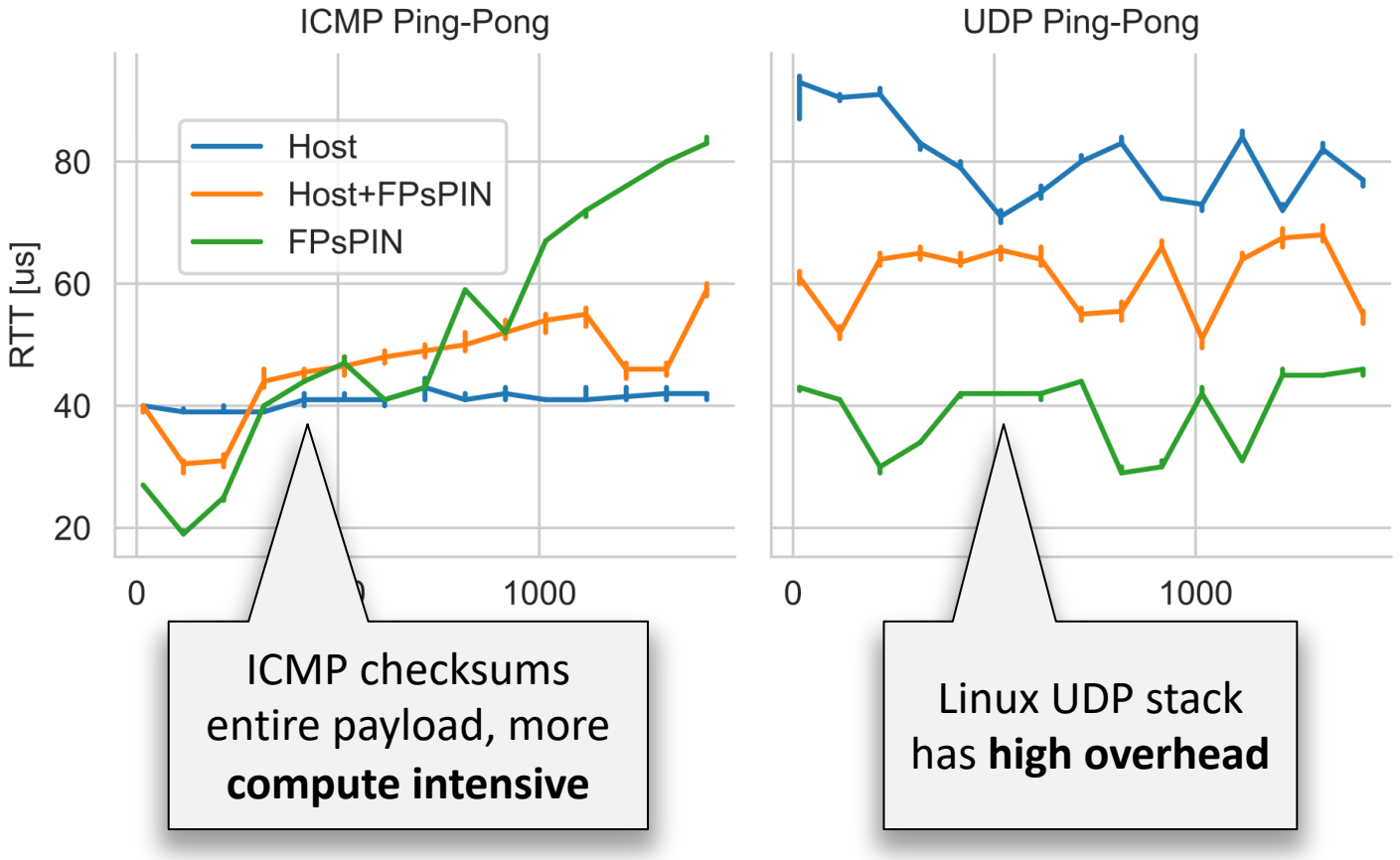


Evaluation


E2E Latency
ICMP Ping
UDP Ping
Overlap Ratio
MPI Datatypes
Throughput
MPI Datatypes
SLMP file transfer

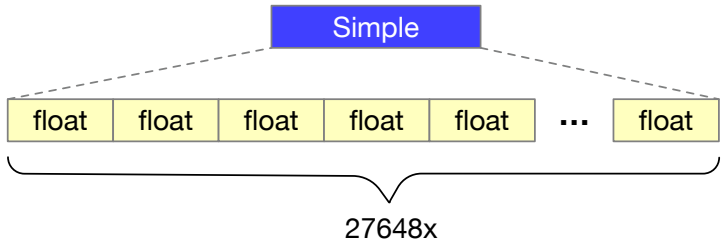
Evaluation: ICMP & UDP echo latency

- Three operation modes
  - *Host: bypass FPsPIN*
  - *Host+FPsPIN: FPsPIN forwards received packet to host, host updates checksum*
  - *FPsPIN: host CPU not involved*



Evaluation: MPI datatypes

Simple := ctg(27648)[float]



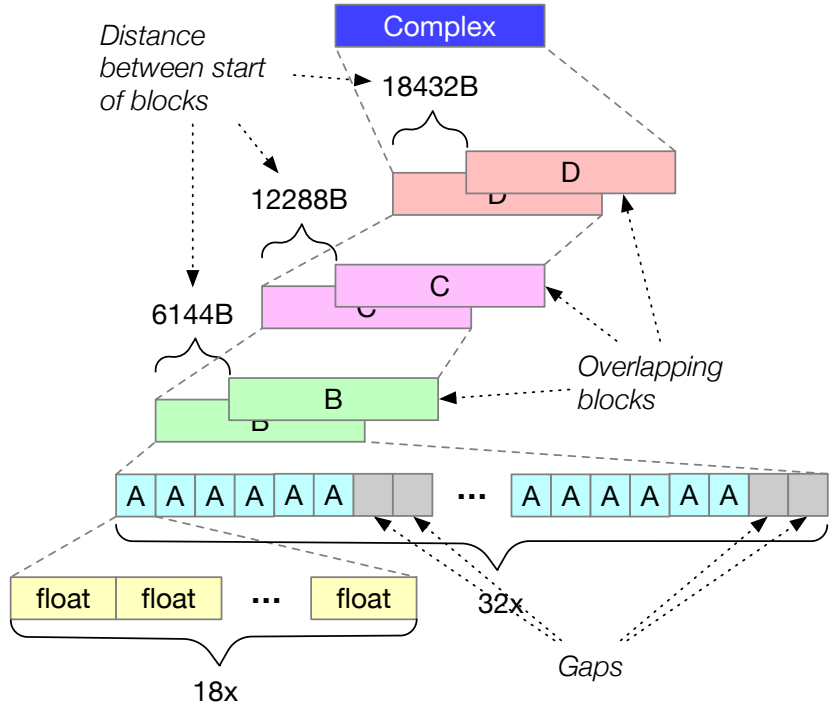
Complex := hvec(2 1 18432)[D]

D := hvec(2 1 12288)[C]

C := hvec(2 1 6144)[B]

B := vec(32 6 8)[A]

A := ctg(18)[float]



Two implementations:

- MPICH Dataloop on CPU
  - *Reference implementation*
  - *"Dataloop" binary generated via compiler, parsed at runtime*
- Ported dataloop to FPsPIN HPU
  - *Same code ported to RISC-V*

Evaluation: overlap ratio

- Overlap matrix multiplication on CPU with MPI Datatypes on FPsPIN
  - *Benchmark for **offloading capabilities** of FPsPIN*

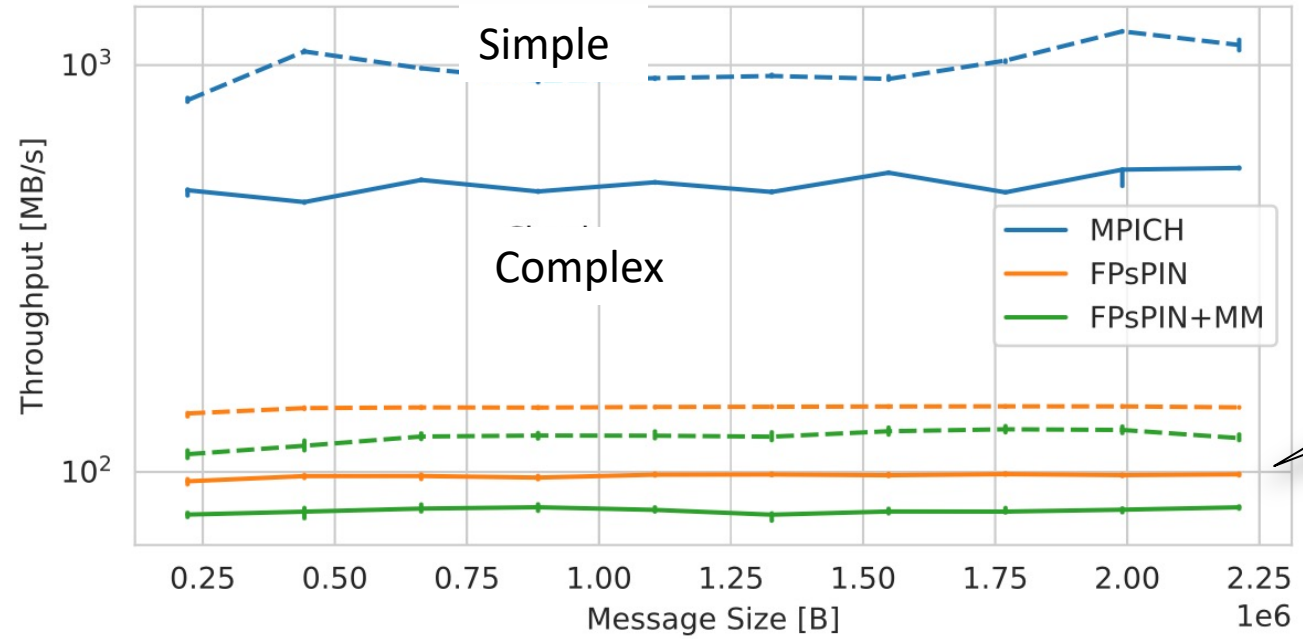
$$r_{\text{Overlap}} = \frac{T_{\text{GEMM}}}{T_{\text{GEMM}} + T_{\text{Poll}}}$$

Computation

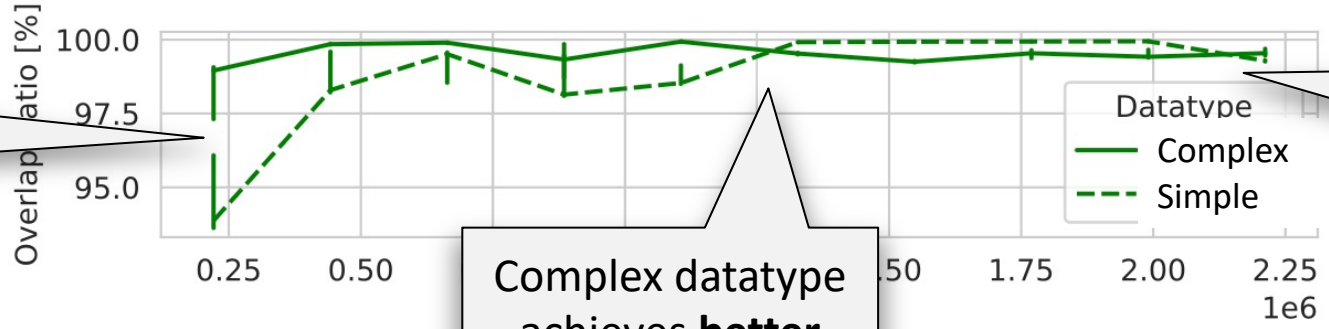
Computation + Communication

- Target: perfect overlap (100% ratio)
  - *"Free" communication during computation*
- Evaluation was **not possible** with PsPIN
  - *Due to lack of host-side interface*

MPI Datatypes throughput & overlap ratio



FPsPIN has **lower throughput** compared to CPU



Shorter messages increase polling overhead

Near-perfect overlap for long messages (>98%)

Complex datatype achieves **better overlapping**

Conclusion

- FPsPIN: smart NIC research platform implementing the sPIN model
- Enables full-system evaluation of NIC offloading for HPC applications
- Hardware and software available open-source: https://github.com/spcl/FPsPIN