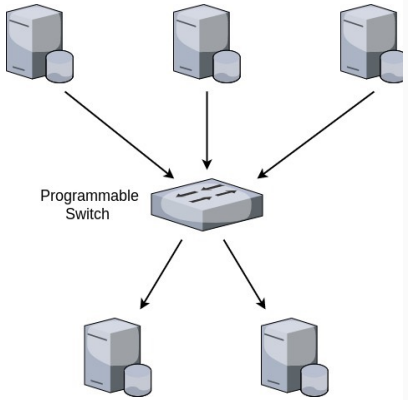
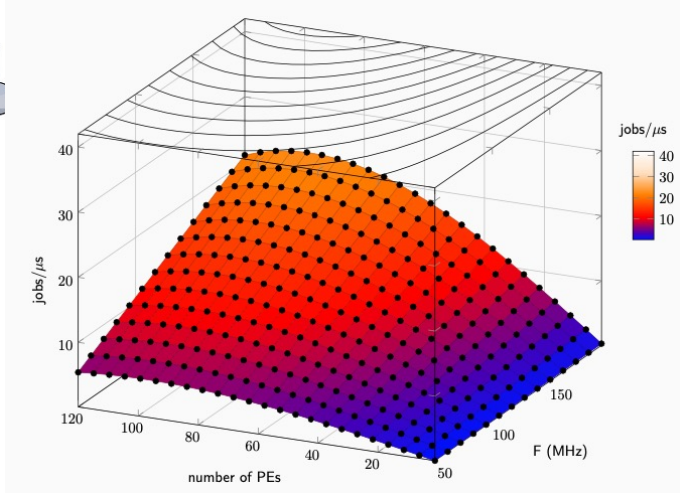
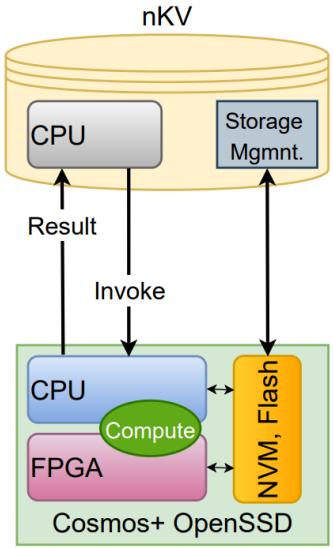
Applications, Tools and Outlook for Reconfigurable Computing: Selected Musings

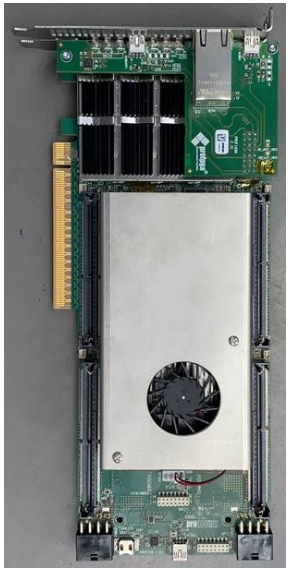
Andreas Koch, TU Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT









Contents

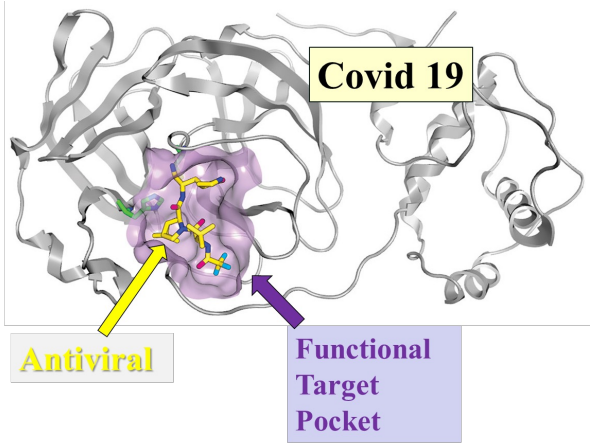
- Reconfigurable computing opportunities in HPC
- Application focus: in-network processing
- Tools and technologies as enablers
- Current practical challenges and upcoming opportunities

RECONFIGURABLE COMPUTING OPPORTUNITIES IN HPC

Reconfigurable Computing Successes in HPC/SC

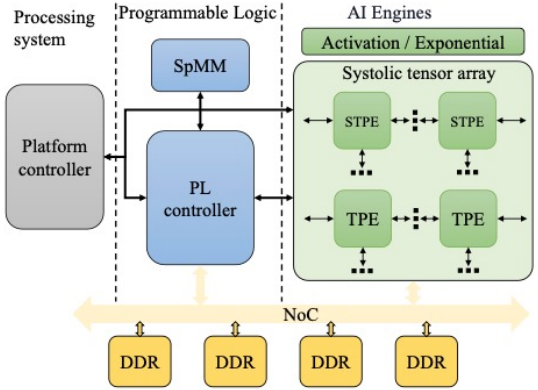
TECHNISCHE UNIVERSITÄT DARMSTADT

Molecular Docking (Martin Herbordt)



- Speedup of **2x** over GPUs at same node [FPL2022], [FCCM21] [SC19]

Graph Neural Networks (Tony Geng)

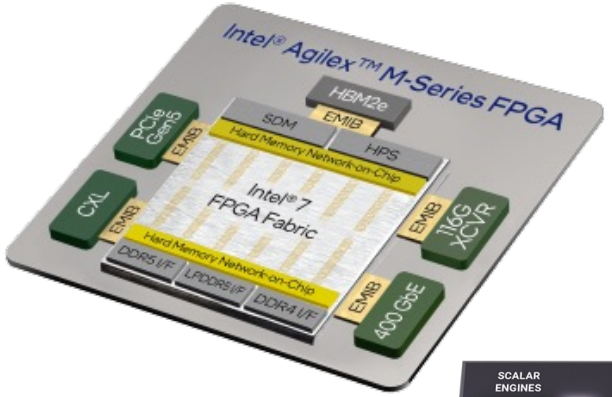


- Speedups of **~20x** on Versal over GPU [FPL 2022], [MICRO20+21]

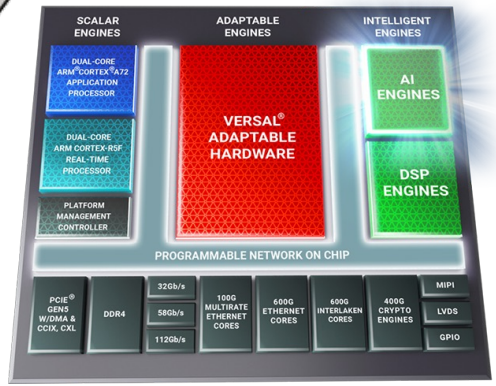
Embedded Systems & Applications

In-Network Processing using FPGAs

- Modern FPGAs & boards are tailor-made for this application
- Lots of networking bandwidth
  - 700 – 800 Gb/s in current Boards
- Heterogeneous parallel memories
  - BRAM, URAM, HBM, DDR4/5, Optane, Flash
- Low-jitter operation in hardware



Intel Agilex [intel.com]



AMD Versal [amd.com]

SAMPLE APPLICATIONS: IN-NETWORK PROCESSING

Network-Attached Reconfigurable Computing

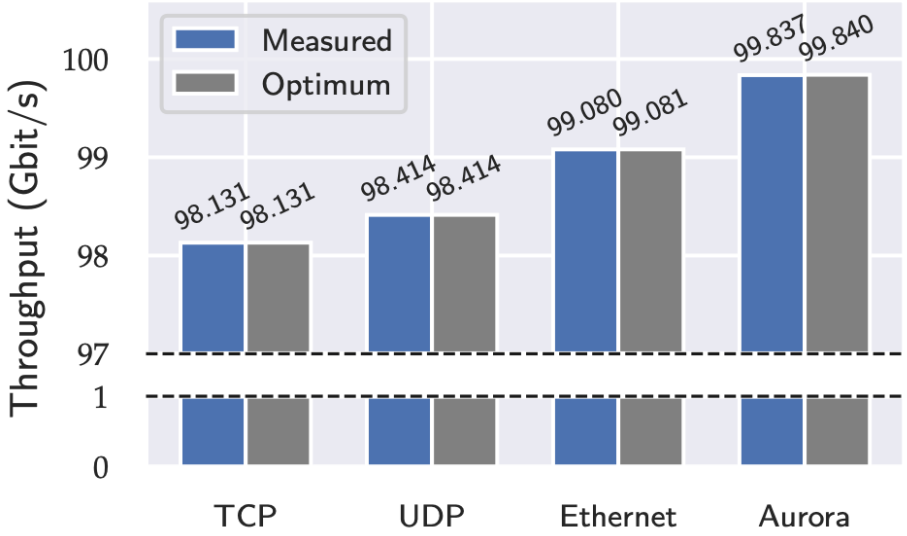
TECHNISCHE UNIVERSITÄT DARMSTADT

- Attach reconfigurable compute accelerators using high-speed network links
  - Instead of PCIe
- Advantages
  - Flexible trade-off of protocol features vs. throughput / latency requirements
  - Potentially higher throughput / lower latency than PCIe
- Good IP block support for network connectivity & protocols in FPGAs

Embedded Systems & Applications

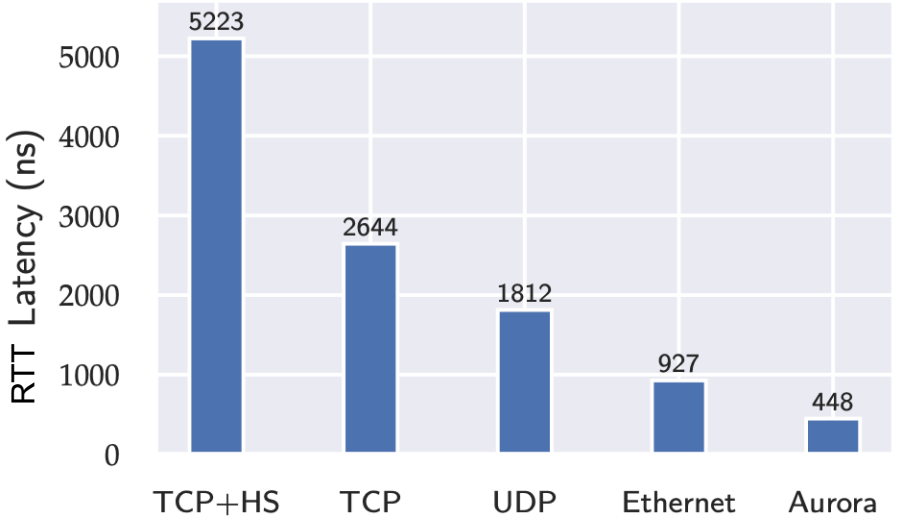
Customizing network interfaces: Throughput

- Inversely proportional to capabilities of network protocol
  - E.g., reliability, routability, switchability, point-to-point
- 100G protocols perform close to the theoretical throughputs on FPGA
  - No software overheads



Customizing network interfaces: Latency

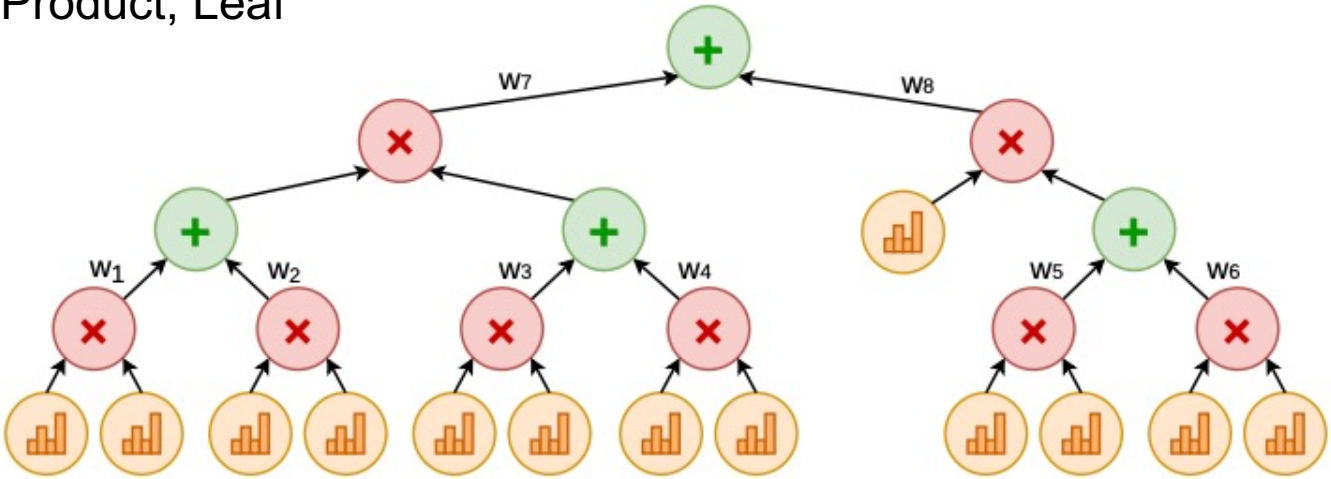
- Also inversely proportional to capabilities of network protocol
- Very low jitter on FPGA at 100G
  - No software overheads



A Network-Attached Appliance for ML Inference in Sum-Product-Networks

Inference in Sum-Product-Networks

- Probabilistic graphical model
  - Captures joint probability over set of random variables
  - Node types: Sum, Product, Leaf



SPN Inference Accelerator Architecture

TECHNISCHE UNIVERSITÄT DARMSTADT

- SPNs can be compiled to fully-spatial FPGA architectures [ICCD 2018]
- Many optimizations
  - Including custom-floating point and logarithmic number systems [FCCM 2020]
- Performance limited by Gen3x8 PCIe throughput of ~7.2 GB/s

→Alternative: Realize an *inference appliance* usable from the entire network

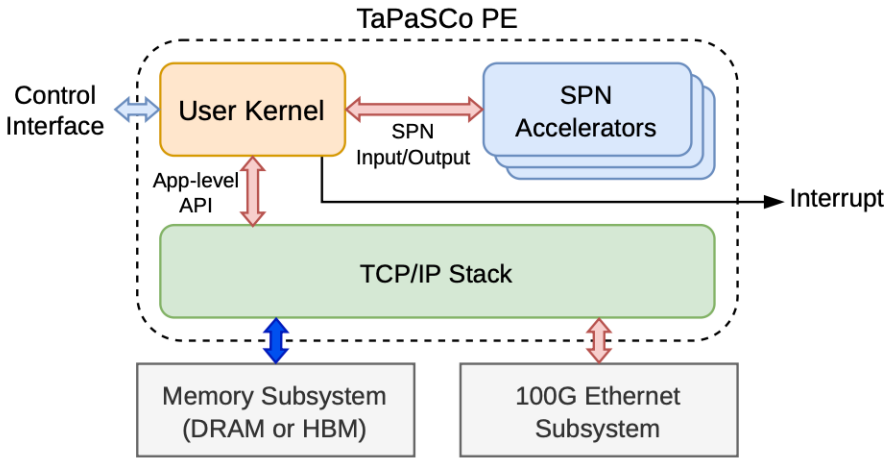
  - Use TCP, UDP or Ethernet; Aurora unsuitable (only P2P)

Low-hanging Fruit

Embedded Systems & Applications

SPN Inference Network Appliance

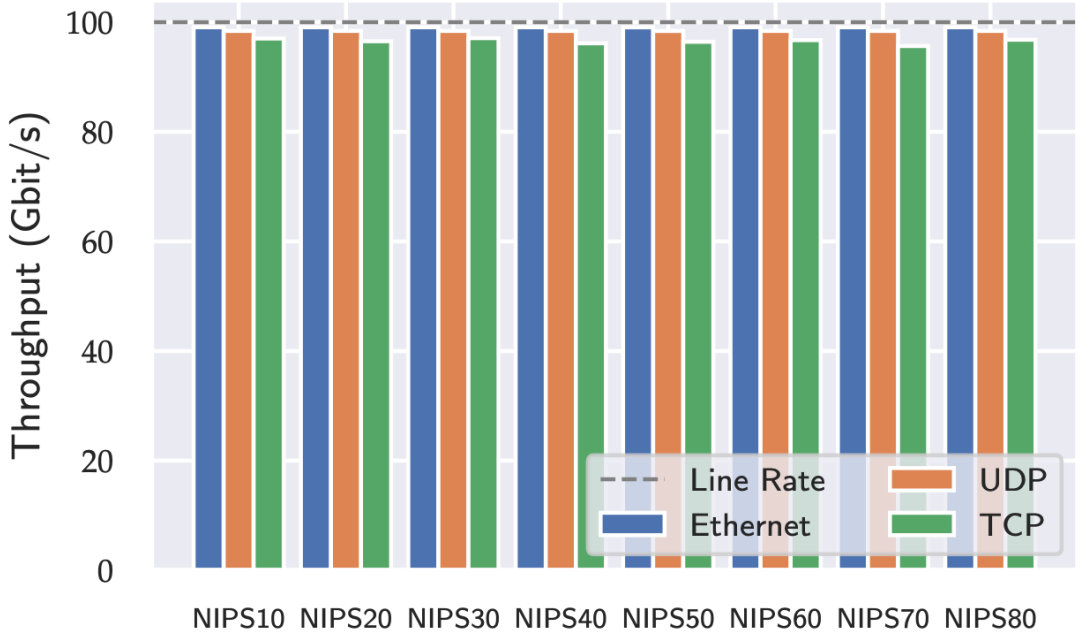
- Aim: Run at full network throughput
  - 100G → 12.5 GB/s
  - More than PCIe Gen3x8
- Xilinx/AMD UltraScale+ FPGA
- Process 50 Bytes / cycle at 250 MHz
  - No problem for larger SPNs
  - Replicate accelerator for smaller SPNs



TaPaSCo PE Architecture

SPN Inference Appliance Performance

- Performance close to 100G
  - For all protocols examined
- Easily scalable to more ports
- Moderate realization effort
  - Uses modified ETHZ 100G TCP/UDP stack



[H2RC 2020]

An In-Network Accelerator for Distributed Database Join Operations

Extending the idea

TECHNISCHE UNIVERSITÄT DARMSTADT

- For the SPN inference appliance
  - A **single** 100G port was used
  - Scaling to multiple ports possible by replicating SPN inference units
- Let's be more ambitious
  - Solve problems which actually require **multiple** network ports
    - Fully exploiting the bandwidth across multiple ports

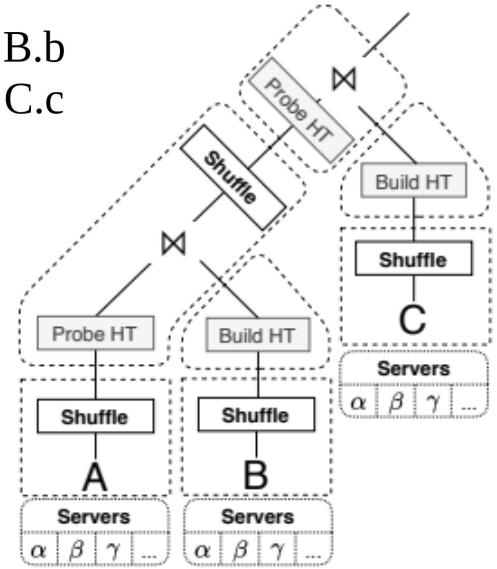
Embedded Systems & Applications

SQL Join Operation

- Common database operation
- Merges two tables/relations
  - Combine entries with matching values
- Implementation: hash join algorithm
- Problematic in distributed DBMS
  - Each Table is distributed *across* servers

  → Shuffling step required

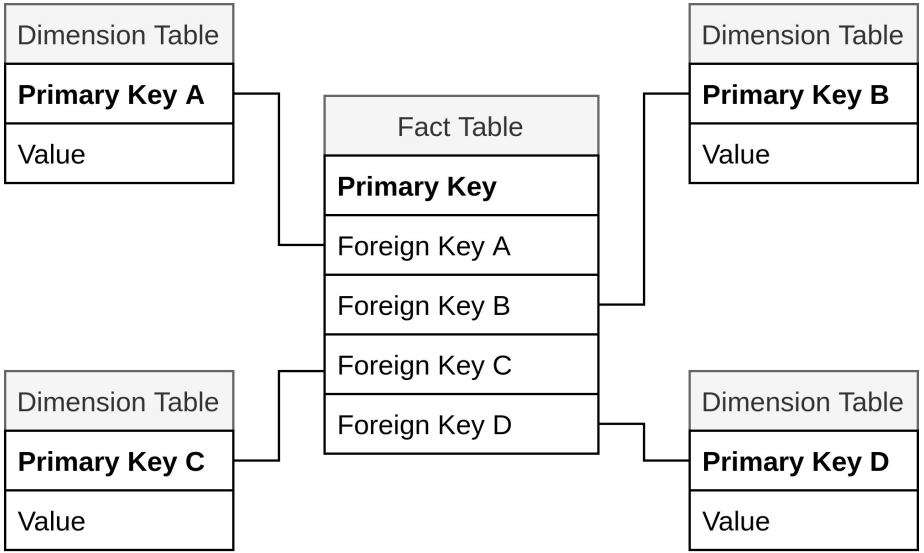
```
SELECT A.a,B.d,C.e
FROM A,B,C
WHERE A.b = B.b
    AND A.c = C.c
```



Common DB Organization: Star Schema

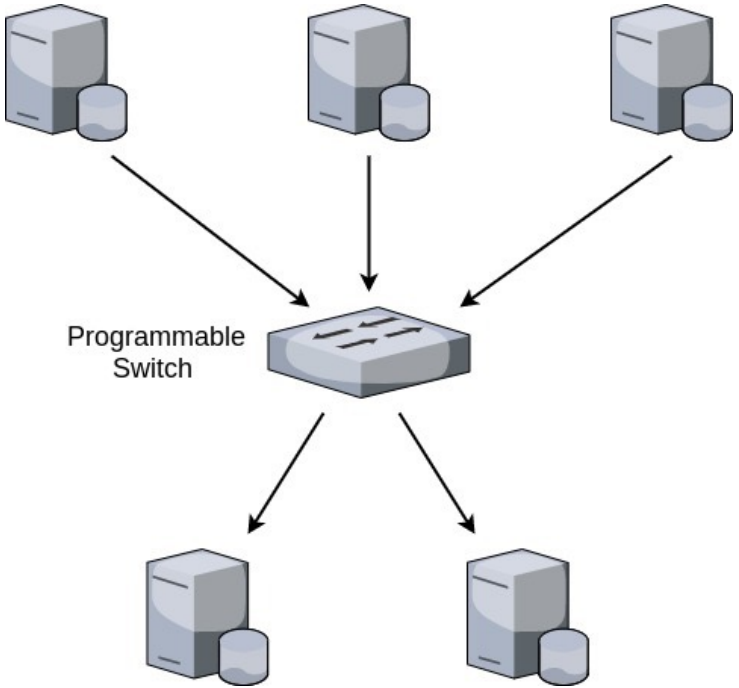
- Large fact table
- Smaller dimension tables

- Foreign key in fact table points to entry in corresponding dimension table
- SQL Join to combine them



In-Network Processing (INP)

- Perform processing on in-flight data
  → Reduce network traffic
- Programmable switches
  - Limited flexibility (e.g., P4)
  - Only small memory
  - Mostly stateless operations
- Alternative: FPGA-based INP
  - Keep state in heterogeneous memory



Idea for INP-based JOIN

TECHNISCHE UNIVERSITÄT DARMSTADT

- Stream smaller **dimension tables** into FPGA memory
  - Building hash tables (key/value)
- Then stream **fact table** across hash tables
  - Retrieving stored data from hash tables
- Exploit **heterogeneous** memory hierarchy on FPGA board
  - Begin with fast HBM (8 GB) on UltraScale+ device
  - Then spill to slower DDR4 DRAM (256 GB) on Bittware XUP-VVH board
  - In the future: Multi-bank high-parallelism Flash (30 TB per board)

Embedded Systems & Applications

Phase 1: Hashing of Dimension Tables

Hashing: bucket(x) = x % 2
Collision Handling: Use next free Slot in Bucket

Table B

b	e	*Server*
1	e1	α
2	e2	β
3	e3	γ

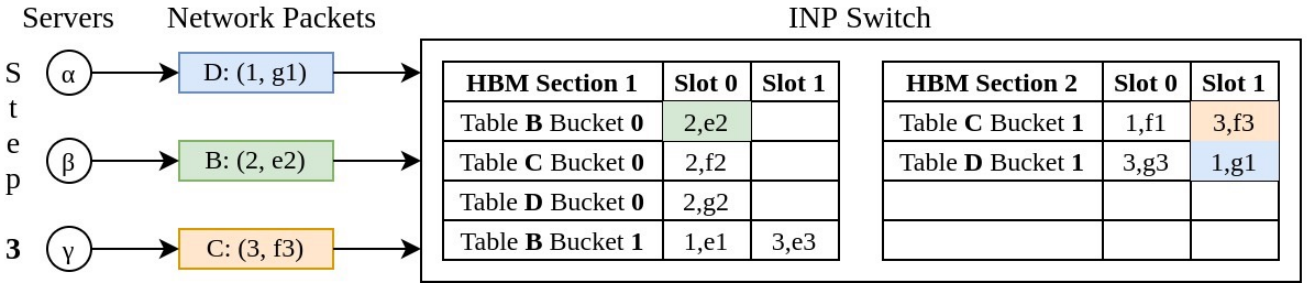
Table C

c	f	*Server*
1	f1	α
2	f2	β
3	f3	γ

Table D

d	g	*Server*
1	g1	α
2	g2	β
3	g3	γ

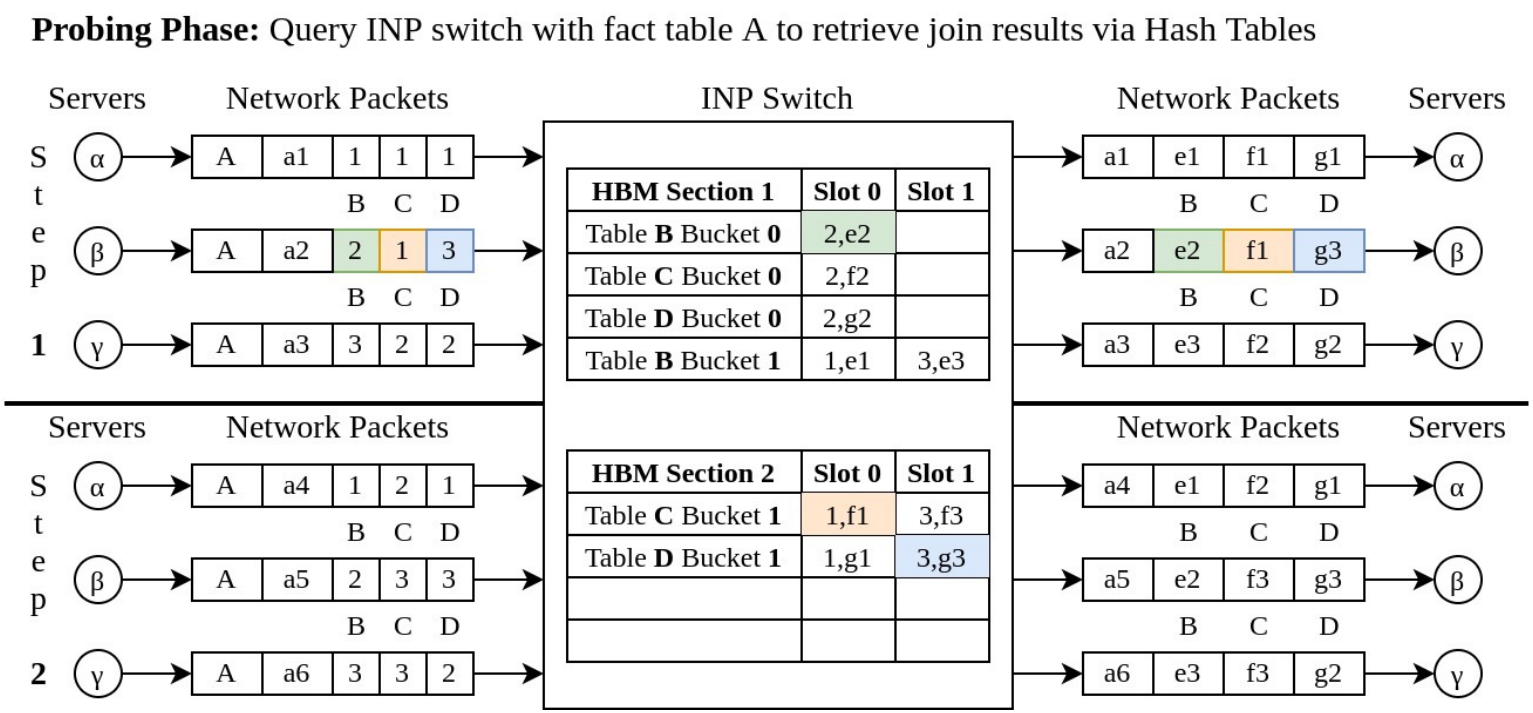
Hashing Phase: Transfer dimension tables B,C,D to INP switch and store in Hash Tables



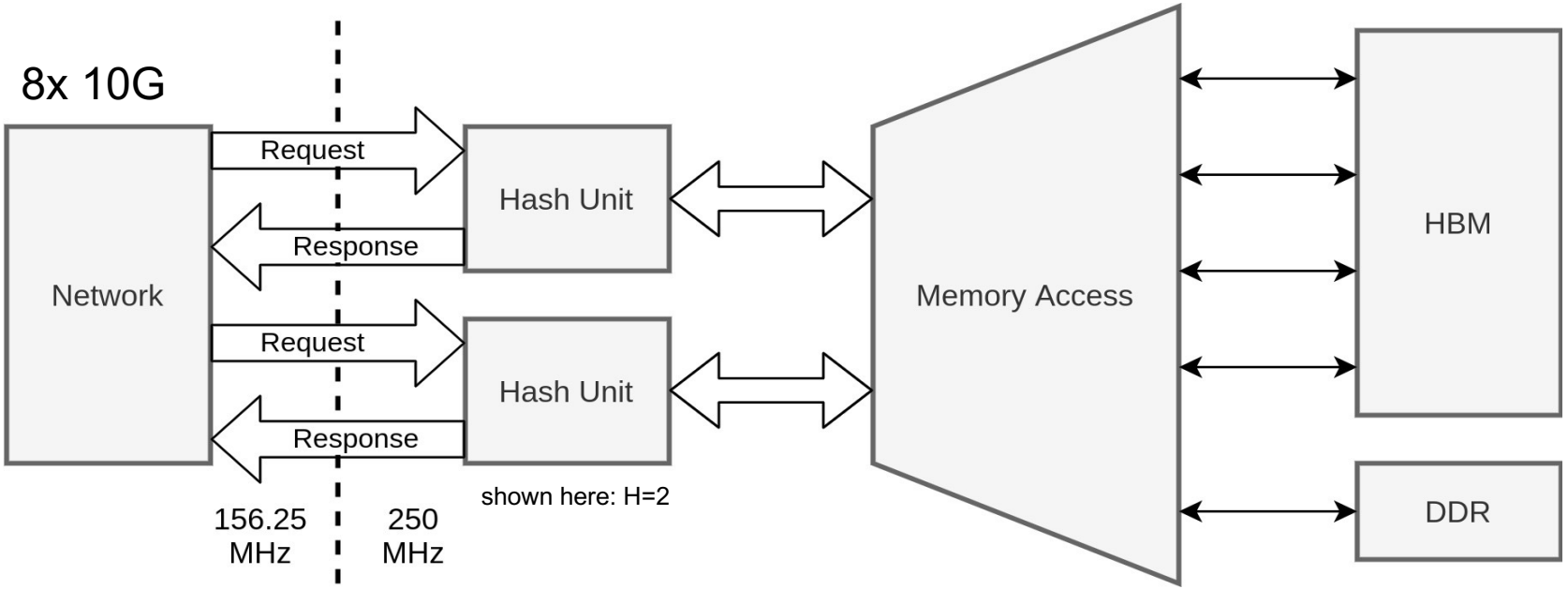
HBM Section 1	Slot 0	Slot 1
Table **B** Bucket **0**	2,e2	
Table **C** Bucket **0**	2,f2	
Table **D** Bucket **0**	2,g2	
Table **B** Bucket **1**	1,e1	3,e3

HBM Section 2	Slot 0	Slot 1
Table **C** Bucket **1**	1,f1	3,f3
Table **D** Bucket **1**	3,g3	1,g1

Phase 2: Probing using Fact Table



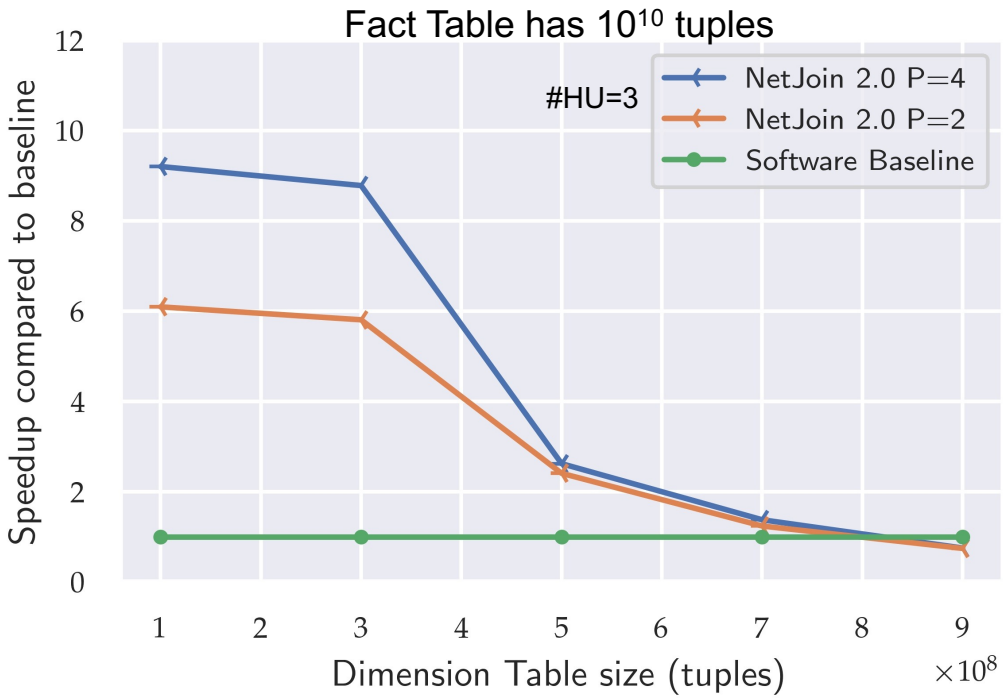
Architecture Overview



Simplified: Data hazard handling and out-of-order operations not shown [FPT 2021]

Performance Comparison

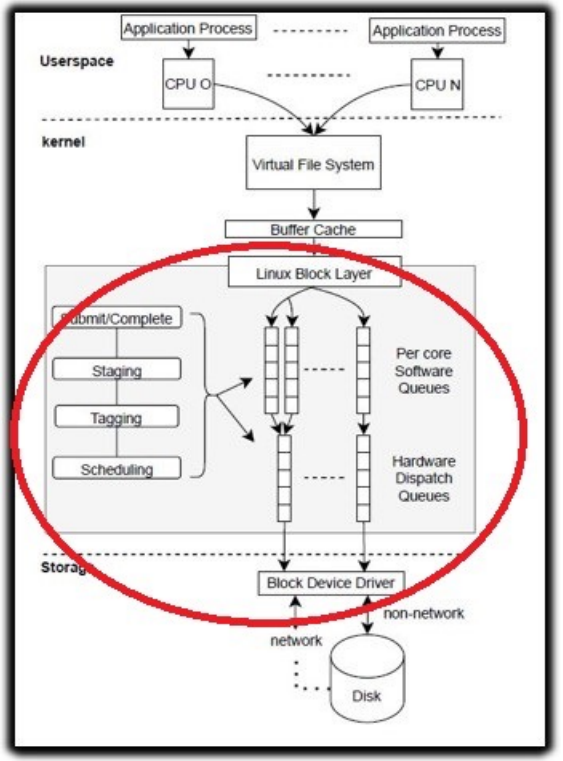
- Software baseline on 8 servers
- Good FPGA speedups
  - … when tables in HBM
- Graceful degradation of FPGA performance
  - Faster DRAM in servers
- FPGA electrical power draw
  - Just 66 W when active



Speeding-up accesses to distributed block storage

Observation & Motivation

- Many HPC applications also have a high I/O effort
- Each I/O operation is expensive
  - 18K-20K instructions per 4 KB block in Linux
- Solutions for different storage architectures
  - Local storage: Smart/computational storage
  - Distributed storage: Network acceleration



Use-Case: Ceph Distributed Storage System

- State-of-the-Art distributed storage
  - Block, File, Object
- Uses decentralized approach to locate replicated data in cluster
- CRUSH Algorithm [Weil SC2004]
  - Compute intensive (45% hashing)

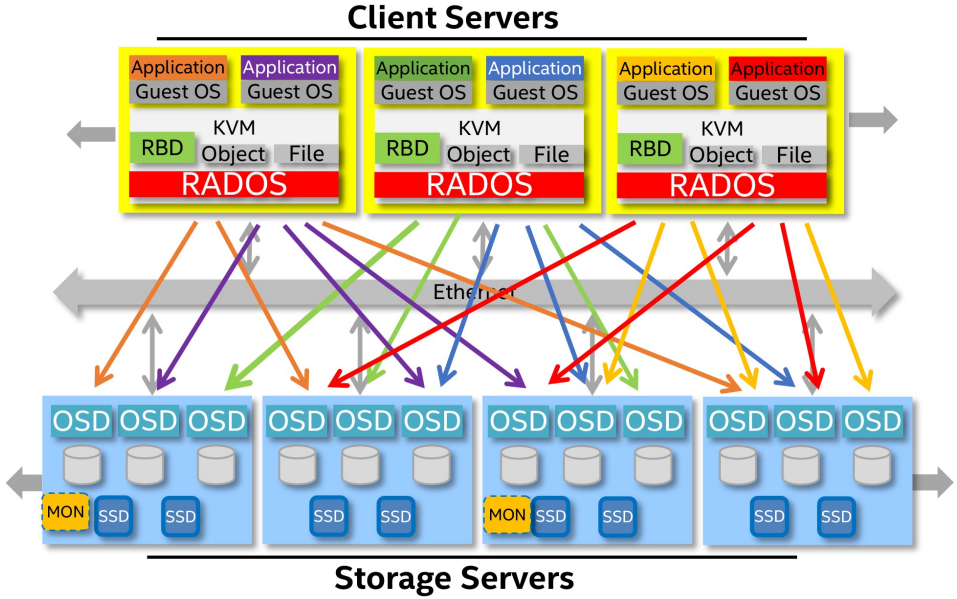


Image: Thomas Krenn Wiki

Speeding up CRUSH on FPGAs using HLS

TECHNISCHE UNIVERSITÄT DARMSTADT

kernel	Software Execution Time	Hardware kernel Execution
Straw Bucket (pure HLS code)	85 µs	0.675 µs
Straw Bucket (using Vitis ln x function IP)	85 µs	0.885 µs
List Bucket	65 µs	0.280 µs
Uniform Bucket	20 µs	2.240 µs
Tree Bucket	45 µs	0.810 µs

- Kernel-Level Speed-Up **~120x**
- Now “just” integrate this into a complete Ceph stack …

Embedded Systems & Applications

… “Just”

TECHNISCHE UNIVERSITÄT DARMSTADT

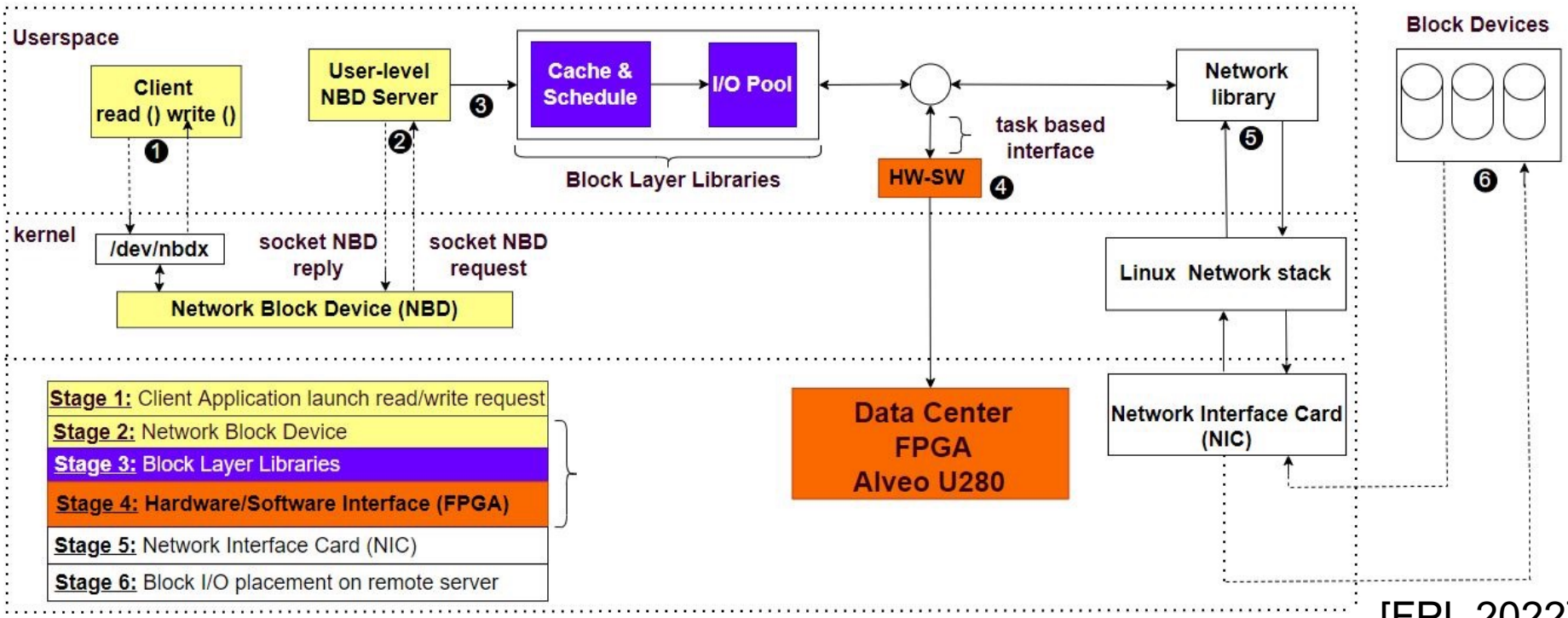
- Many moving parts
  - Linux kernel integration (challenging programming environment)
  - FPGA System-on-Chip
  - Ceph custom “Messenger” network protocol
  - TCP-based network connections
  - Interfaces: SW/HW, HW/SW, HW/HW, user/kernel spaces

→ Too many moving parts to get right “all at once” (at least for us)

Open-Source DeLiBA Framework

Development of Linux Block I/O Accelerators → **in User Space**

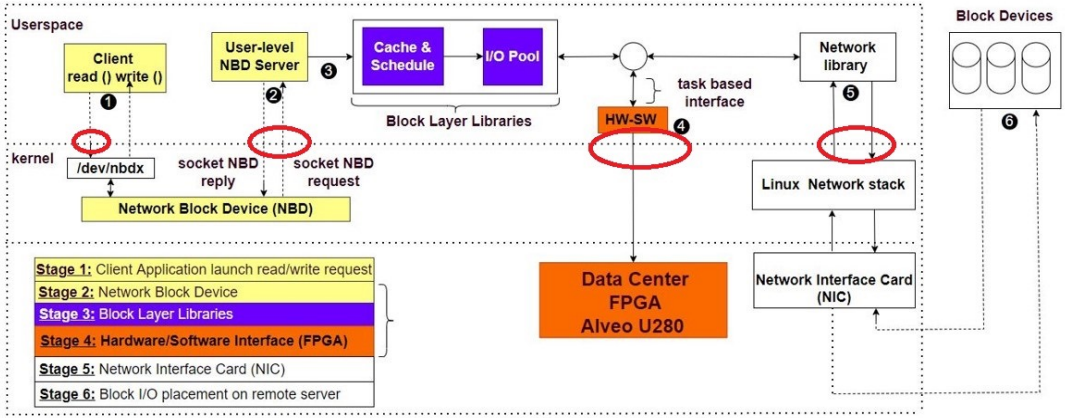
TECHNISCHE UNIVERSITÄT DARMSTADT



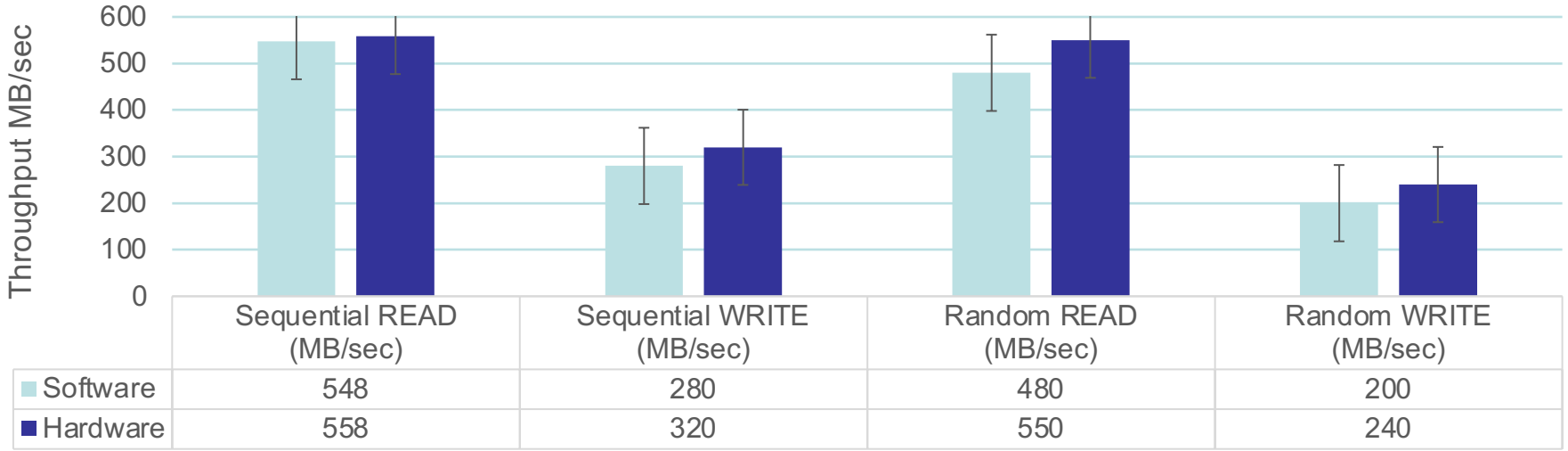
[FPL 2022]

DeLiBA Benefits and Caveats

- Has enabled us to actually bring-up a *complete* hardware-in-the-loop Ceph client protocol stack
- Good base for further research in block I/O acceleration
- But actual performance suffers from numerous kernel/user & HW/SW context switches



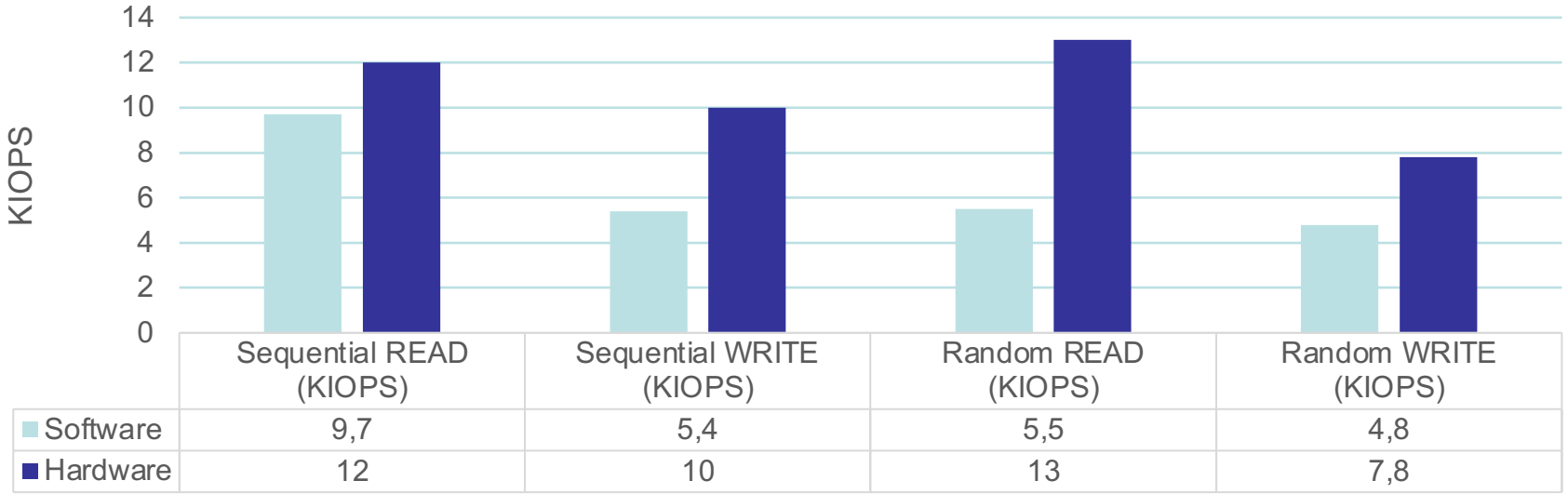
Hardware CRUSH – Throughput 128 KB requests



	Sequential READ (MB/sec)	Sequential WRITE (MB/sec)	Random READ (MB/sec)	Random WRITE (MB/sec)
Software	548	280	480	200
Hardware	558	320	550	240

- Software Baseline: NBD-based Ceph Stack on AMD EPYC 7302P
- With FPGA-in-the-Loop up to **1.2x** (RWr 128 KB) and **1.9x** (SWr 4 KB)

Hardware CRUSH – IOPS 4 KB requests

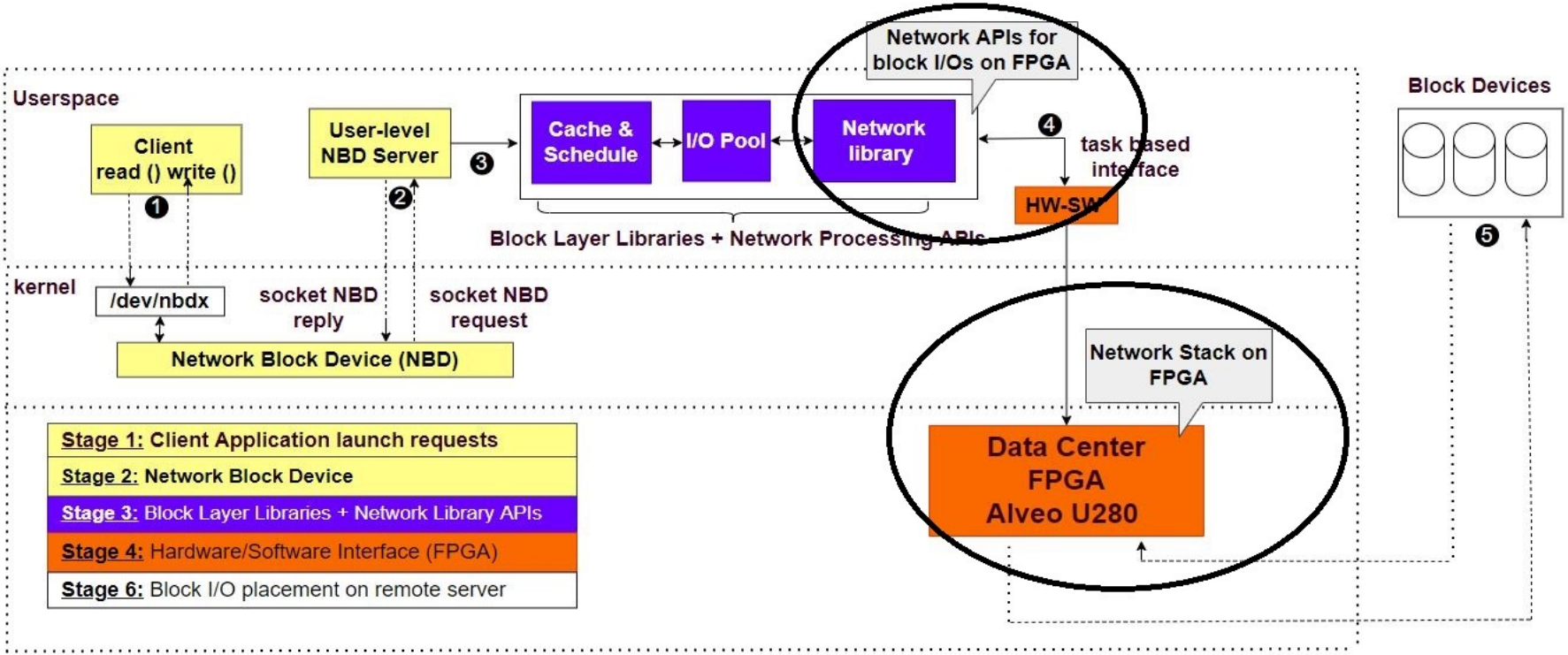


	Sequential READ (KIOPS)	Sequential WRITE (KIOPS)	Random READ (KIOPS)	Random WRITE (KIOPS)
Software	9,7	5,4	5,5	4,8
Hardware	12	10	13	7,8

- Software Baseline: NBD-based Ceph Stack on AMD EPYC 7302P
- Improvement with FPGA-in-the-Loop of up to **2.36x** (RRd 4 KB)

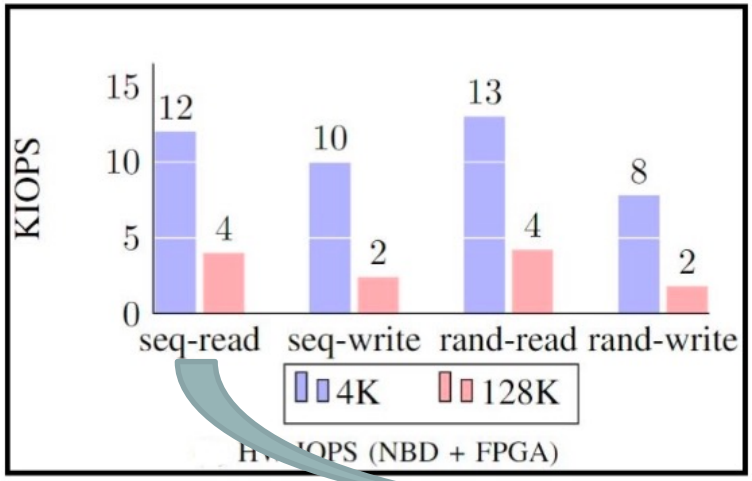
Optimization in DeLiBA2

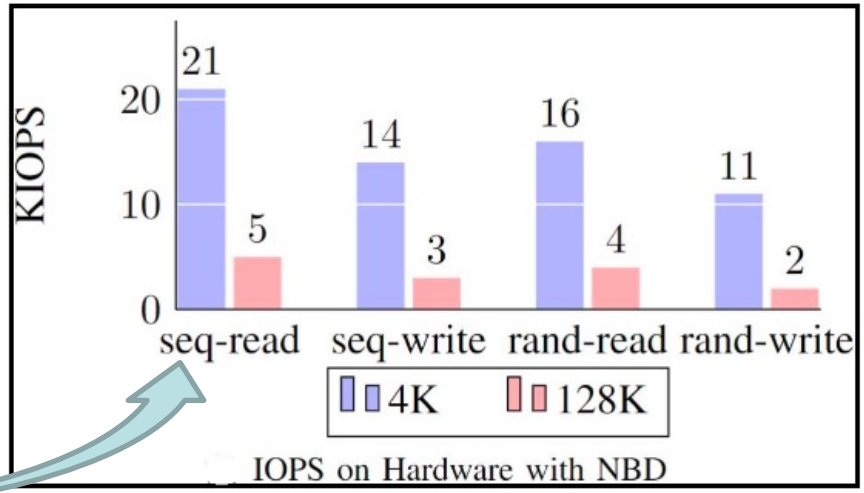
Reduce number of context switches → Also move networking to FPGA



Improved Performance in DeLiBA2

Most visible in IOPS, less in throughput





1.75x

DeLiBA1 [FPL 2022]

DeLiBA2 [preliminary]

More acceleration potential for Ceph

TECHNISCHE UNIVERSITÄT DARMSTADT

- Remove *all* unnecessary context switches
- Move to 100G networking
- Also consider server-side acceleration

→ all ongoing

- Ideas most likely also applicable to other distributed storage systems
- Especially beneficial to achieve low latency / low jitter

  → typical FPGA qualities

Embedded Systems & Applications

TOOLS AND TECHNOLOGIES AS ENABLERS

Progress in Tools / Support Libraries

TECHNISCHE UNIVERSITÄT DARMSTADT

- Traditional FPGA mantra heard at many a conference: “The tools suck …”

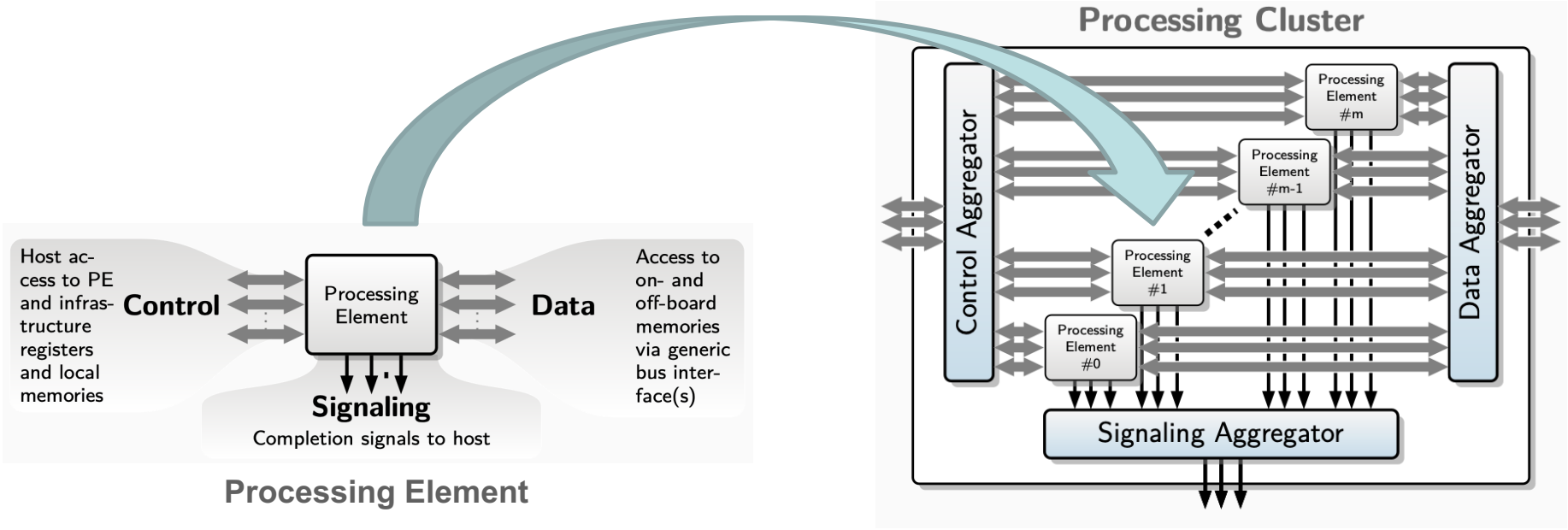
- Base for (almost) all of our research since 2015
  - Task Parallel System Composer (TaPaSCo)
  - https://github.com/esa-tu-darmstadt/tapasco

- Continuously improved, latest features:
  - Inter-PE-Communication, On-Chip-Dynamic Dispatch, Shared Virtual Memory with Physical Page Migration

Embedded Systems & Applications

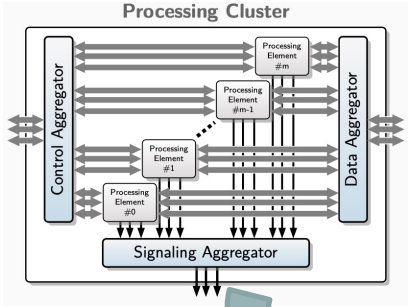
TaPaSCo Design Hierarchy

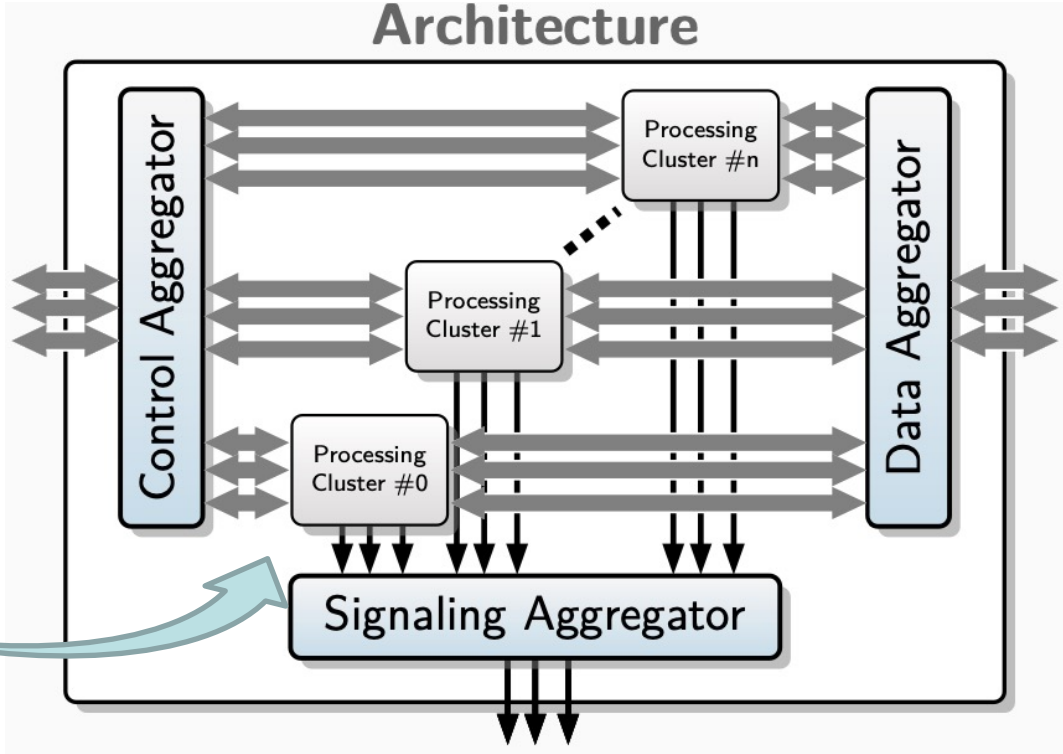
Processing Element and Processing Cluster



TaPaSCo Design Hierarchy

Processing Clusters and Architecture



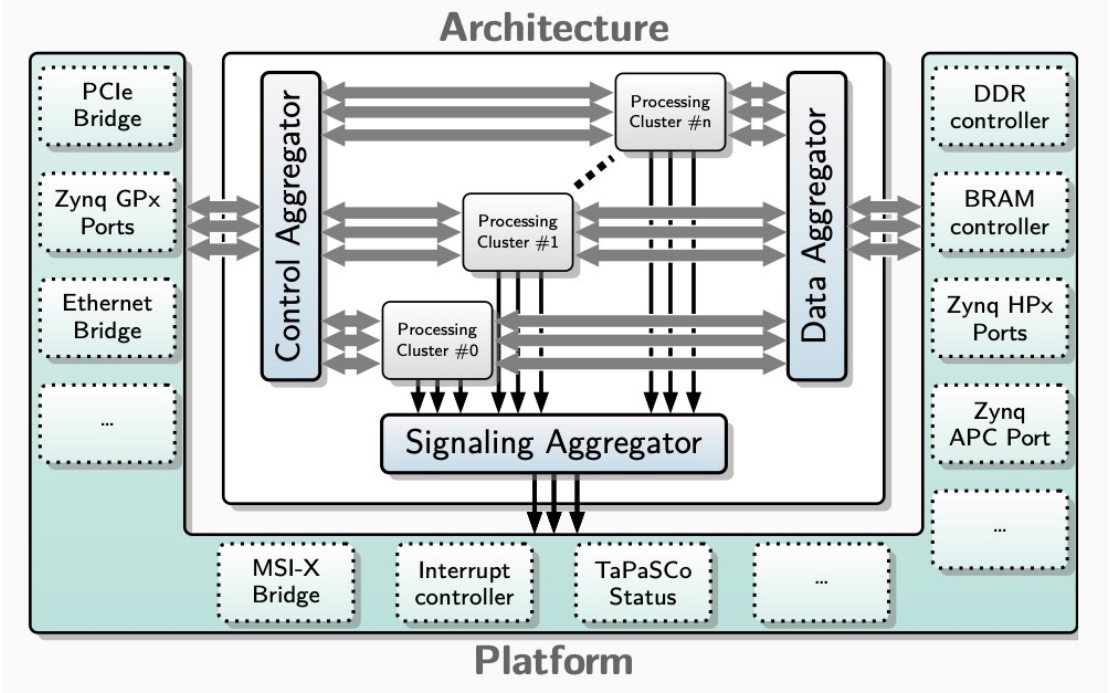


TaPaSCo Design Hierarchy

Architecture and Platform

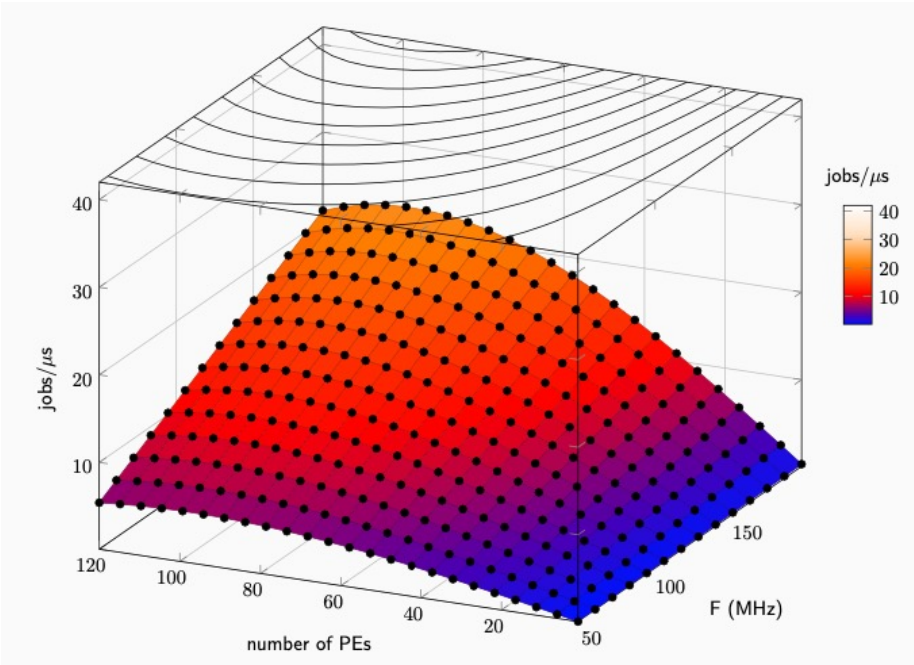
TECHNISCHE UNIVERSITÄT DARMSTADT

- Wraps all hardware specifics
- Allows highly portable designs
  - Boards
    - From Pynq to data-center PCIe
  - Devices
    - Zynq 7000 and MPSoC
    - Virtex 7, Ultrascale+
    - Versal



TaPaSCo Compilation Flow

- PEs are imported into the TaPaSCo flow as IP-XACT blocks
  - Can be described in arbitrary hardware design style (HDL, HCL, HLS, …)
- Automatic Design Space Exploration
  - Aiming for highest task throughput



TaPaSCo Programming Interface

Shown here: C++ API

TECHNISCHE UNIVERSITÄT DARMSTADT

Abstractions for

- Data transfers
- Synchronization
- FPGA specifics

```
int arraysum(int *in, size_t size)
{
    Tapasco tapasco;
    auto in_buf = makeInOnly(makeWrappedPointer(in, size));
    int sum = -1;
    RetVal<int> ret_val(&sum);

    // launch job and wait for PE
    auto job = tapasco.launch(PE_ID, ret_val, in_buf);
    job();
    return sum;
}
```


Embedded Systems & Applications

New TaPaSCo Features in 2022

TECHNISCHE UNIVERSITÄT DARMSTADT

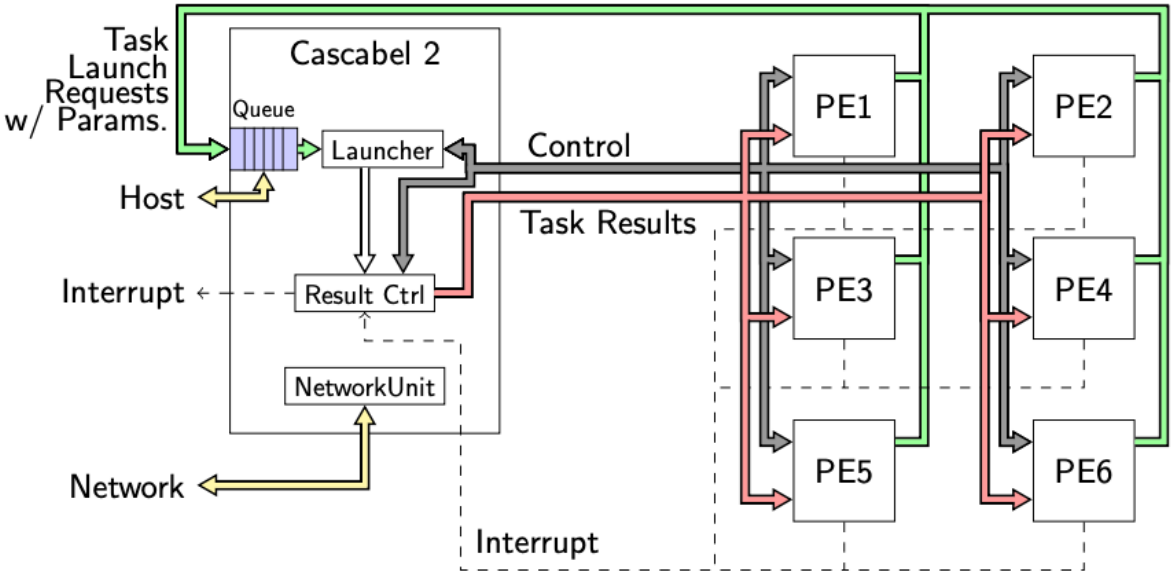
- On-chip dynamic task dispatch: Cascabel 2
- Inter-Processing Element Communication: IPEC
- Shared Virtual Memory with Physical Page Migration

Embedded Systems & Applications

Low-Latency Task Creation/Dispatch On-Device

Cascabel 2

- Similar to CUDA Dynamic Parallelism
- PEs can launch tasks
  - With parameter and result passing
- On-chip dispatch to idle PE of matching type
- Can even launch across network



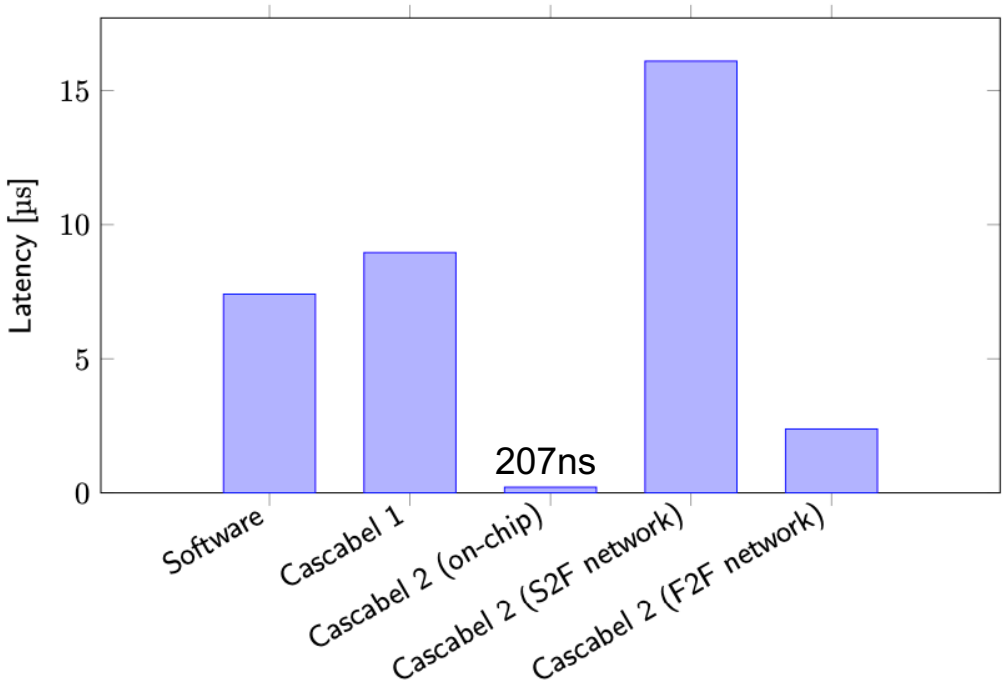
[JSPS 2022]

Cascabel 2 Hardware API and Performance

TECHNISCHE UNIVERSITÄT DARMSTADT

Bluespec Example

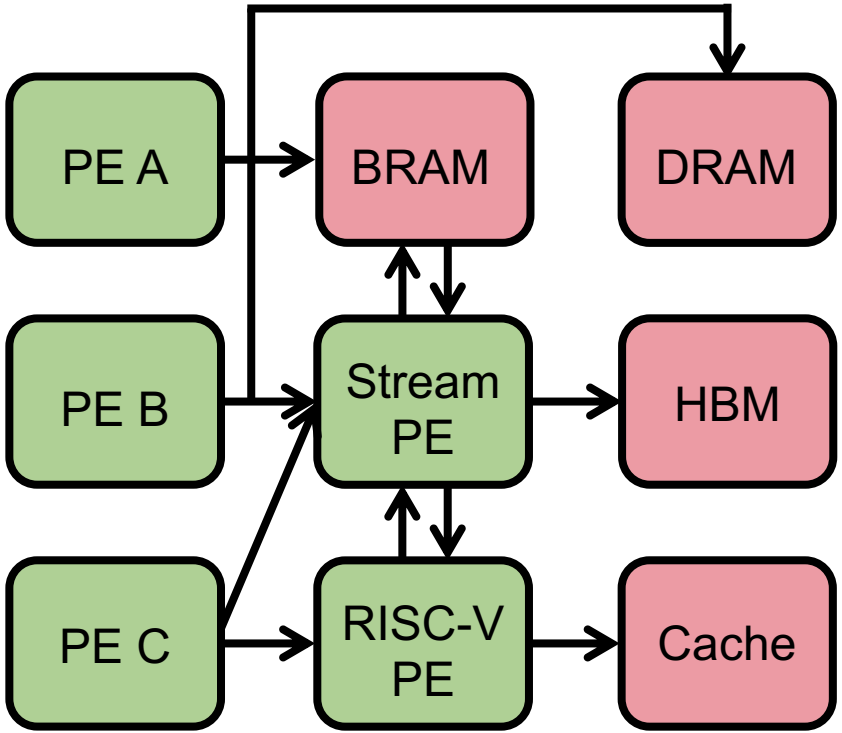
```
let tapasco <- mkTaPaSCo();

rule launch;
    let data <- alu.get();
    tapasco.launch1(PE_ID, data);
endrule

rule result;
    let r <- tapasco.result();
    outFifo.enq(r);
endrule
```



Embedded Systems & Applications

Inter-Processing Element Communication (IPEC)

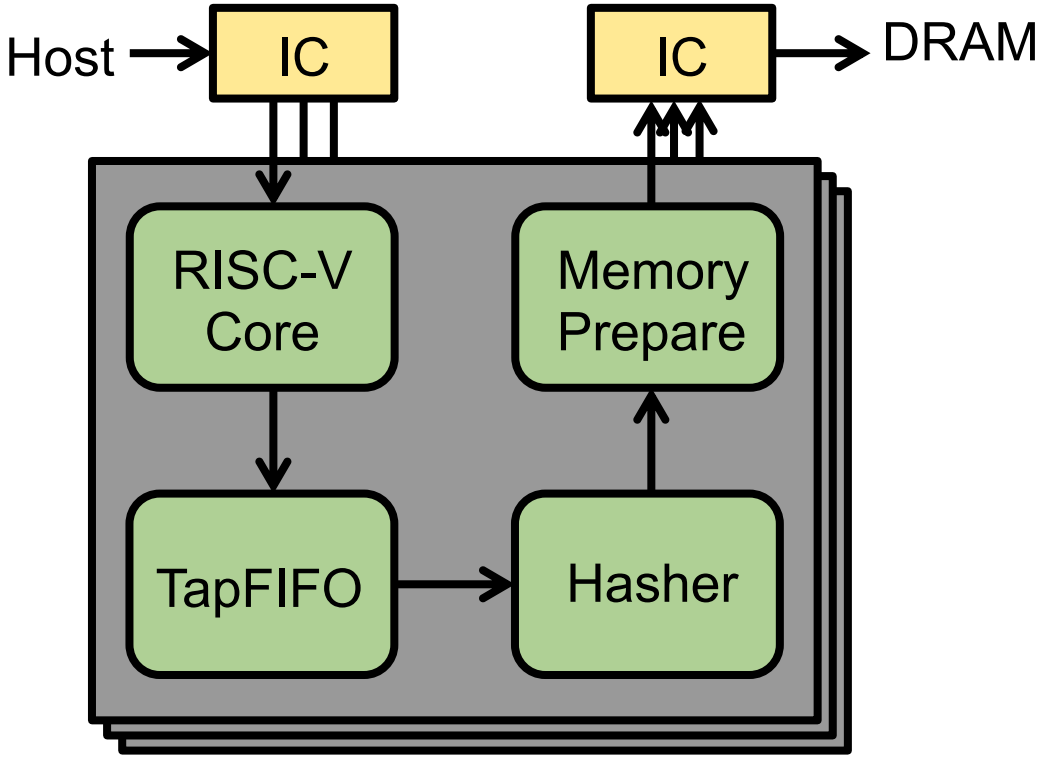
- Allow communication *between* PEs
- Across multiple interfaces
  - AXI4, AXI4Lite, AXI4Stream
- Access to heterogeneous memories
  - BRAM, DRAM, HBM
- RMA, locks and atomic ops for sync
- PEs can start other PEs
  - Directly, no dispatch required



Communication Patterns described in Python

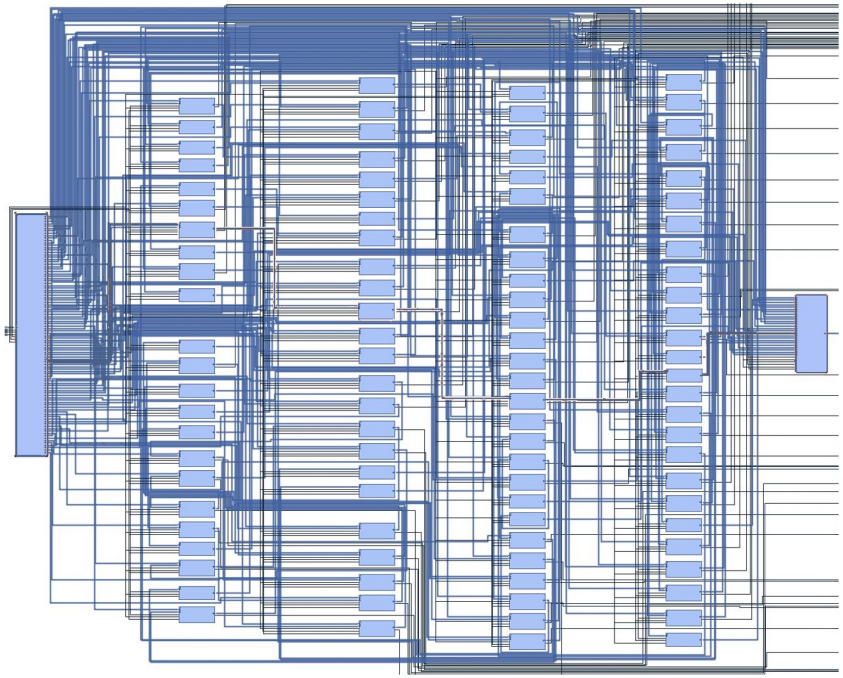
Example: HW-Assisted Control Flow Tracer for Software Fuzzing

```
dram = DRAM()
for i in range(25):
        pea = RISCV(id=i*4+0)
        peb = TapFIFO(id=i*4+1)
        pec = Hasher(id=i*4+2)
        ped = Memory_Prep(id=i*4+3)
        Connect(pea.dout, peb.din)
        Connect(peb.maxis, pec.saxis)
        Connect(pec.maxis, ped.saxis)
        Connect(ped.maxi, dram.saxi)
```



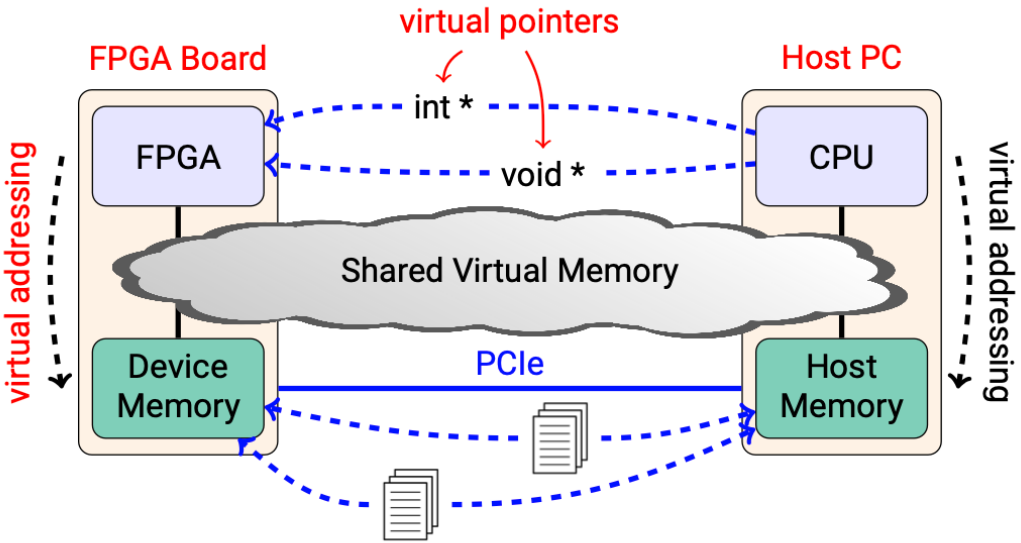
Productivity Gains

- 10 LoC IPEC Python vs ~2000 LoC Tcl
- Automatic optimization
  - insertion of protocol converters
  - merging of ports into channels
- Allows rapid design space explor.
- Example: 25 clusters had best performance



Shared Virtual Memory w/ Phys. Page Migration

- Freely exchange virtually addressed pointers between host and FPGA(s)
- Migrate memory pages to local memory of active compute unit
  - Virtual addresses preserved
- On-demand and user-managed migrations

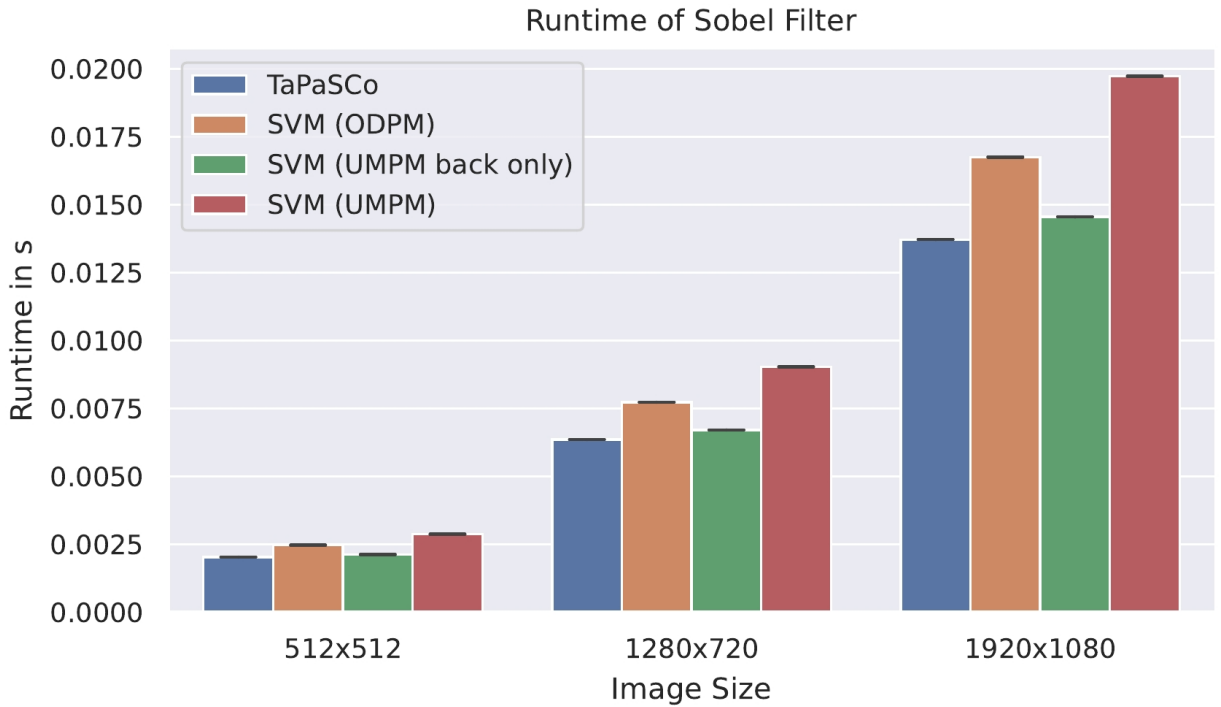


Also between FPGA boards over Ethernet

[FPL 2022]

SVM/PPM Performance

- Performance competitive with explicit data transfers
  - SVM/PPM vs. TaPaSCo copies
- But simplifies programming even further



OUTLOOK

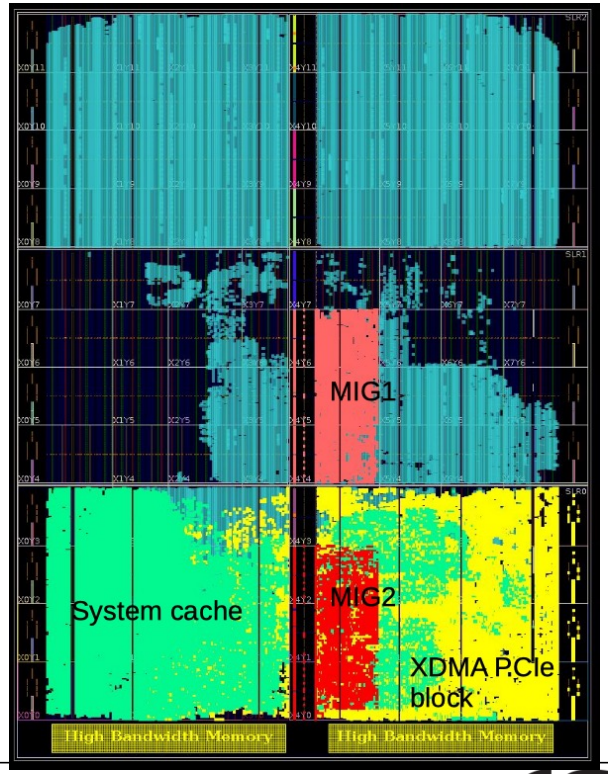
A Practical Challenge

Large FPGAs use Multi-Tile (SLR) Implementation

- Only limited routing available between tiles
  - Difficult to achieve good timing closure
- Aggravated by unfortunate placement of I/O
  - Severe congestion in lower-right part of chip
  - HBM not even used yet

→ better device / board designs required

* early tool support available [UCLA Autobridge]



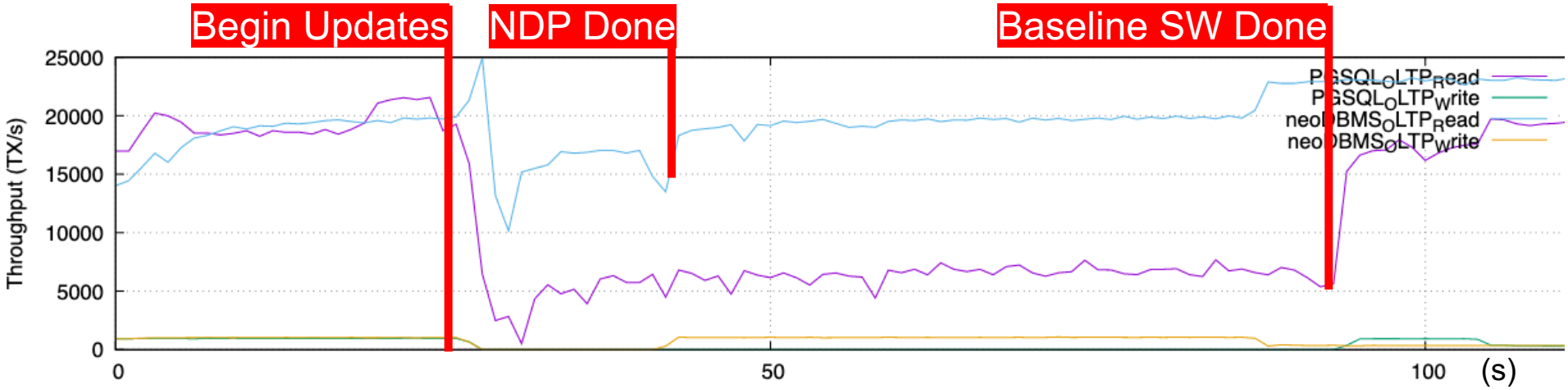
New Opportunity

Cache-Coherency in Standard Interfaces

TECHNISCHE UNIVERSITÄT DARMSTADT

- CCIX (legacy), CXL (upcoming)
- Example: NDP Database acceleration
- **Fine-graine**d HW/SW co-processing
- PostgreSQL + CCIX-attached FPGA

HTAP Scenario: Long-running data analytics disturbed by database updates



[new work, extends FCCM 2022]

Embedded Systems & Applications

Conclusion

TECHNISCHE UNIVERSITÄT DARMSTADT

- RC opportunities **beyond** traditional computing
  - Network processing (INP)
  - Smart storage (NDP)
- Tools for reconfigurable **computing** are improving
  - TaPaSCo from Darmstadt, but also other similar efforts, e.g., UCLA Tapa
- Outlook
  - Interactions of device architecture and board designs become more **challenging**
  - Promising new technologies (e.g., CXL) open **new application domains**

Embedded Systems & Applications