PADSYS

UNIVERSITY OF CALIFORNIA MERCED

High-Performance Parallel and Distributed Computing (HPDC) 2025

DPU-KV: On the Benefits of DPU Offloading for In-Memory Key-Value Stores at the Edge

Arjun Kashyap, Yuke Li, Xiaoyi Lu

{akashyap5,yli304,xiaoyi.lu}@ucmerced.edu

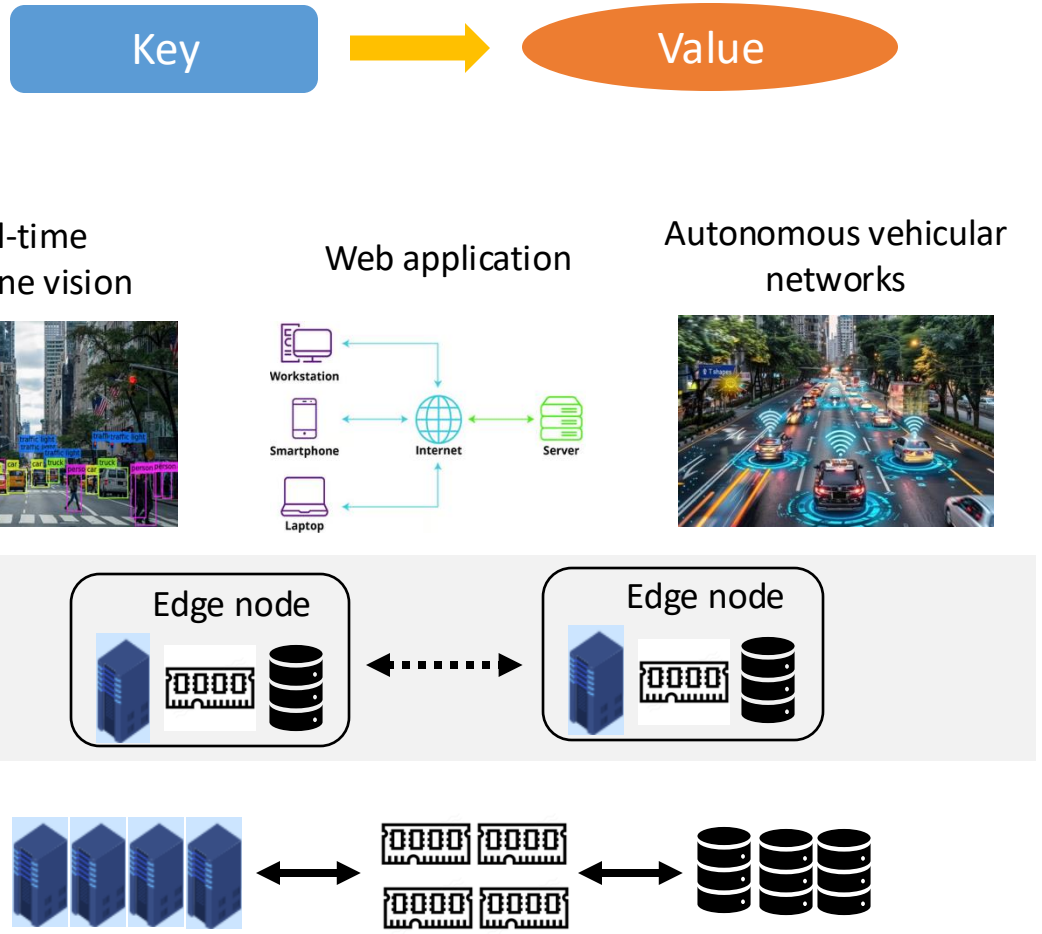
Department of Computer Science and Engineering
University of California, Merced

Outline

- Introduction and Background
  - Edge Computing
  - DPUs
- Related Work and Motivation
- Fine-grained KVS Offloading to DPU
  - Challenges, design, and benefits
- Evaluation
- Conclusion

Edge Computing

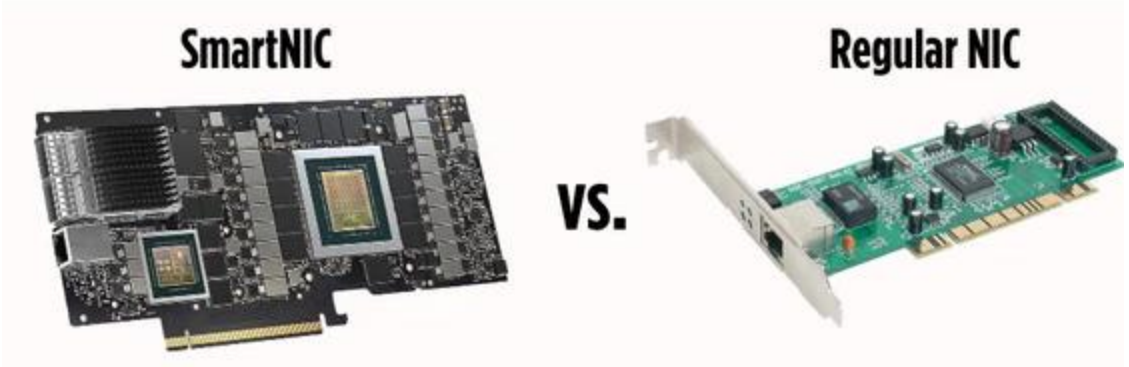
- Key-value store (KVS) critical component for edge storage
- Edge storage requires
  - Low latency
  - High throughput
  - Low server resource utilization
- Edge servers have -
  - Limited compute
  - Require low power consumption



https://medium.com/@mou.abdelhamid/learning-computer-vision-machine-learning-c1521ee6ed08
https://www.liquidweb.com/blog/client-server-architecture/
https://fpgainsights.com/wireless-networking/role-of-wireless-networking-in-autonomous-vehicles/

What is a Data Processing Unit (DPU)?

- SmartNIC or DPU
- Network Interface Card (NIC) facilitates communication between nodes
  - Cannot perform any additional operations
- DPU adds computing power to a regular NIC
- Components
  - Computing elements
  - On-chip memory
  - Hardware accelerators
  - Network adapter



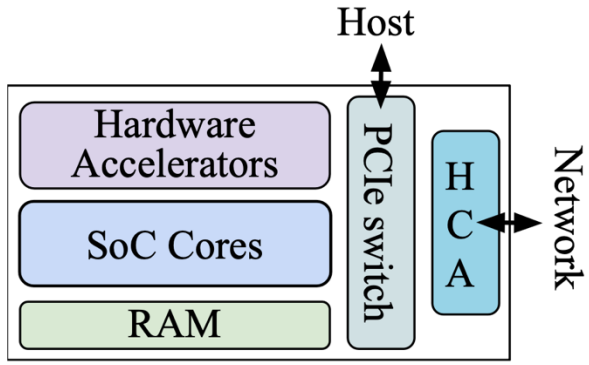
source: https://premioinc.com/blogs/blog/smartnic-vs-regular-nic-differences-explained

NVIDIA BlueField DPUs

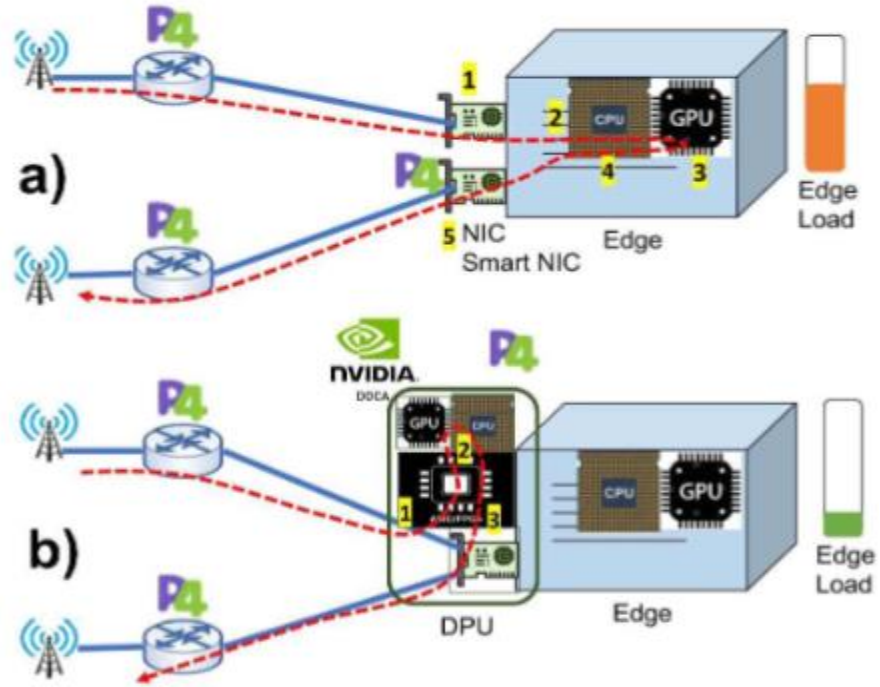
	BlueField-1	BlueField-2	BlueField-3
Network	ConnectX-5 Up to 100 Gbps	ConnectX-6 Up to 200 Gbps	ConnectX-7 Up to 400 Gbps
Compute	Up to 16 ARMv8 A72	Up to 8 ARMv8 A72	Up to 16 ARMv8.2 A78
Memory	Up to 16GB DDR4 DRAM	Up to 32GB DDR4 DRAM	32 GB DDR5 DRAM
Hardware accelerators		DMA engine Compression/Decompression	

DPUs Rise in Edge

- Low power devices
- Compute units and hardware accelerators help edge applications offload tasks
  - *Better utilization* of limited host computing resources



BlueField (BF) DPU architecture



Partial (a) and full (b) in-network function offload at edge

source - https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9868927

Related Work

Existing KVS Designs
Target edge deployments

No co-design with DPUs

DPUs at Edge
Telemetry, network security, Load balancing

Yet unexplored for Edge KVS

DPU Offloading
Used for MPI, compression

Fine-grained KVS offloading untapped

We are the first work to explore the performance benefits of **fine-grained KVS offloading to DPUs at edge**

DPU-KV achieves **68%** lower latency and **36%** higher throughput over CPU-based KVS and naïve KVS offloading

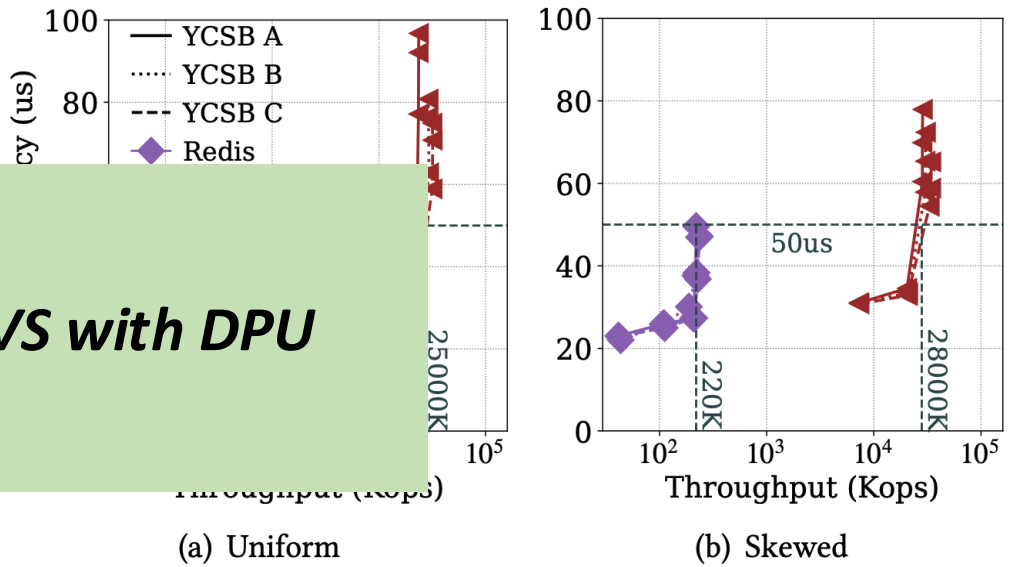
Testbed and Workload

- Two edge host machines (server and client)
  - 12-core 3.3 GHz Intel Xeon E-2136 CPU and 32GB DRAM
  - Use lower-performance hosts to mimic low-power edge systems
- Workloads
  - YCSB A (50% GET), YCSB B (95% GET), YCSB C (100% GET)
- Key-value request generator (MICA client)
  - ~200 million key-value pairs
  - 8B key, 8B value
  - Uniform and Skewed (Zipfian 0.99) key distributions
  - Uses all cores (12)

Conventional Edge KVS Fall Short on Performance

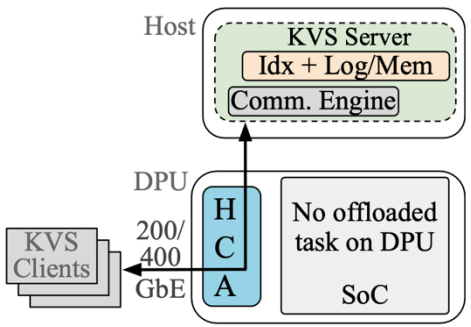
- Redis KVS popular in edge and cloud
- MICA KVS o
- Redis achiev
  - *127x* lowe
    50us laten

Choose to Co-design MICA KVS with DPU

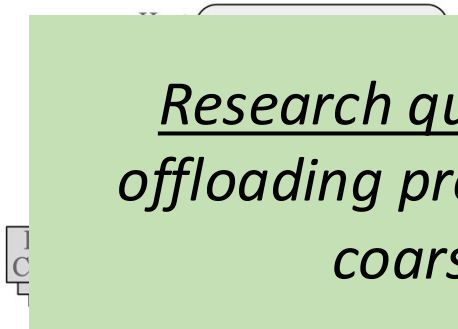


Performance comparison of in-memory Redis and MICA KVS on host

Coarse-Grained DPU Offloading Limits Performance



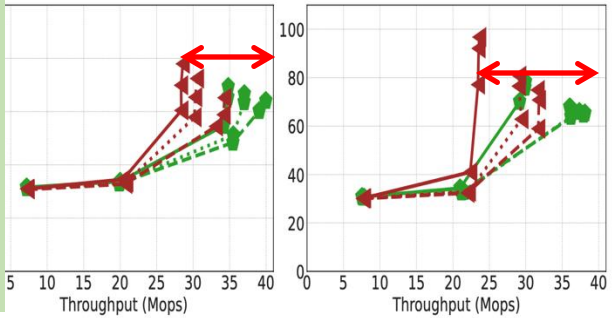
CPU-only
(no offloading)



DPU-only
(full offloading)

Performance comparison of *CPU-only* and *DPU-only* KVS for YCSB workloads

Research question: Can fine-grained KVS offloading provide better performance than coarse-grained offloading?

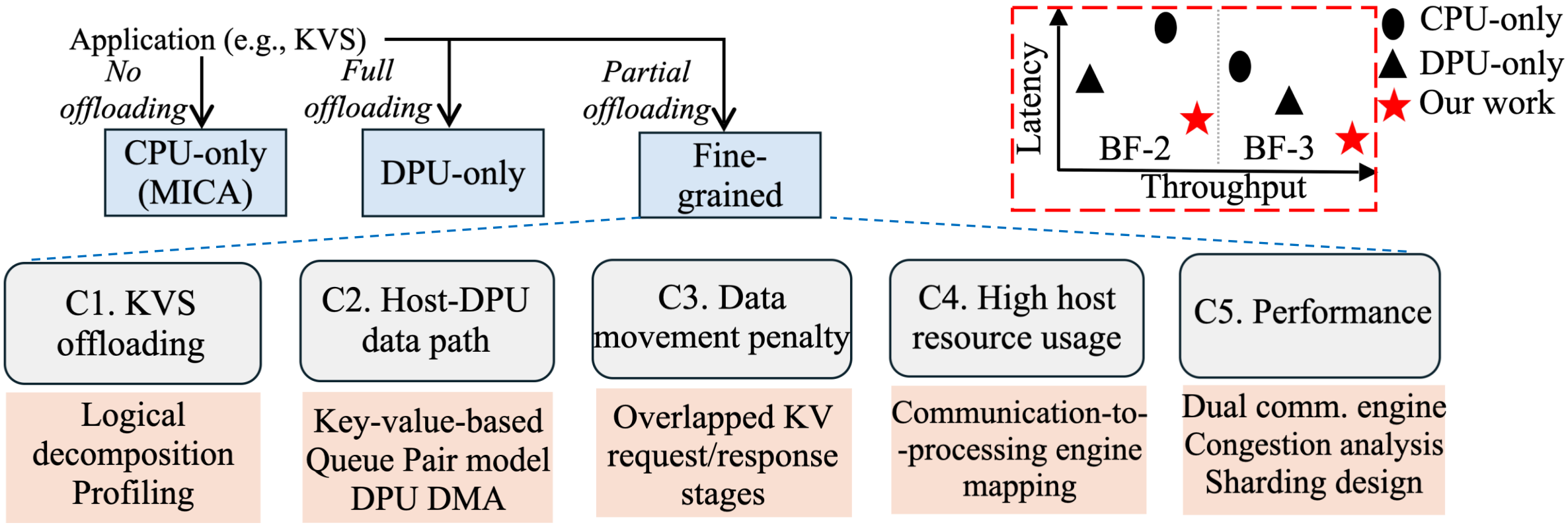


(a) Skewed (BF-2) (b) Uniform (BF-2) (c) Skewed (BF-3) (d) Uniform (BF-3)

Coarse-grained offloading (*DPU-only*)

- With BF-2 exhibits lower performance than *CPU-only* with BF-2
- With BF-3 exhibits superior performance than *CPU-only* with BF-3

Offloading Benefits of KVS to DPUs at Edge



- *Fine-grained offloading*
  - Improve performance (latency and throughput)
  - Enable resource sharing among other edge services/applications by freeing up resources consumed by KVS tasks

C1 – KVS Components & Profiling

- Processing engine
- Communication engine

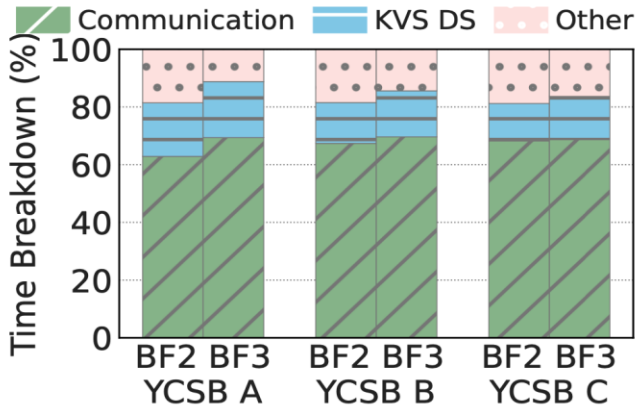
- Communication engine is more CPU-intensive
  - CPU utilization
  - Time consumed

CPU usage of KVS components

Component (engine)	Host Function	CPU usage (%) YCSB A BF2	YCSB A BF3	YCSB B BF2	YCSB B BF3	YCSB C BF2	YCSB C BF3
Packet Processing (comm.)	receive	18.2	25.4	16.4	23.7	14.4	17.2
	parse	17.1	20.7	21.9	20.6	23.9	23.4
	response	27.5	23.2	28.9	25.3	29.8	28.1
Key-value Processing	Get/Set	19.5	13.4	15.9	10.2	13.1	9.5
	others	17.7	17.2	16.6	20.1	18.5	21.7

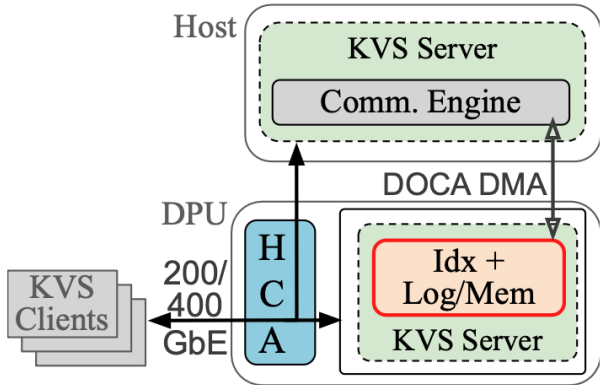
70%

KVS time breakdown for KVS

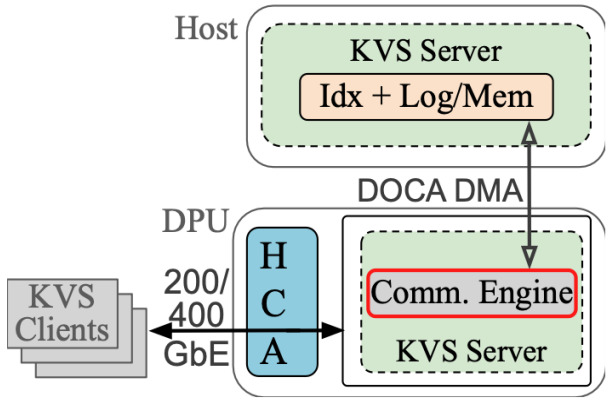


Only 10%-19% time spent in processing engine

C1 – Logical Decomposition & KVS Offloading



Processing engine offload



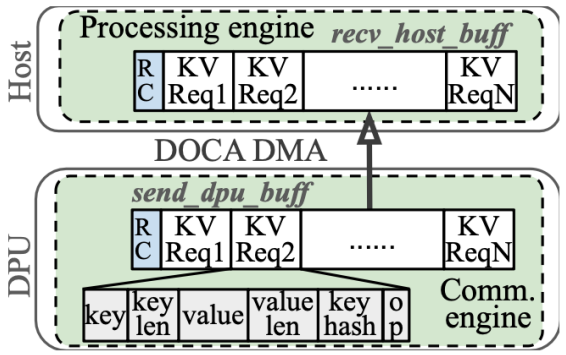
Communication engine offload

Why communication engine offloading?

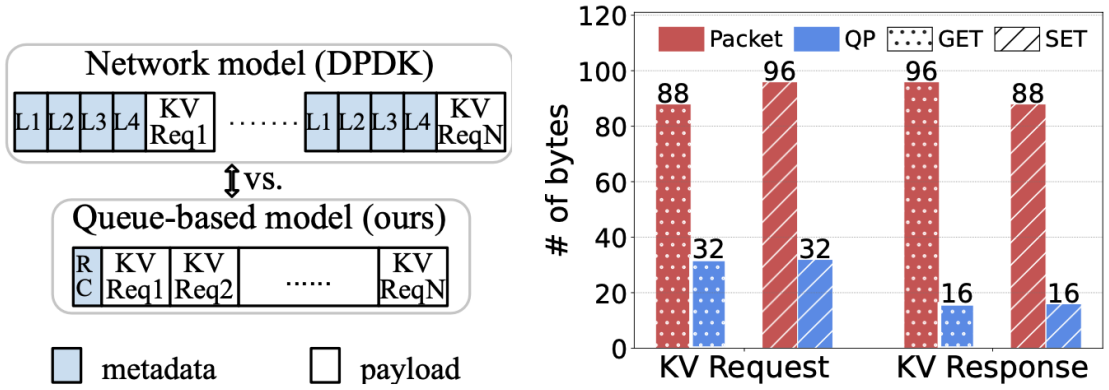
- Communication engine consumes most CPU cycles (~70%)
  - Offloading PE will only help save up to 20% host CPU cycles
- KV item set size of in-memory KVS >> DPU memory
- Communication engine memory requirement ~ 1-4GiB << DPU memory

C2 – Efficient Host-DPU Data Path

- Key-value-based queue-pair (QP) model
  - Per-core send and receive QPs on both host and DPU
  - Minimal metadata
    - ❖ Reduces bytes transferred across PCIe
    - ❖ 63.67%/83.3% and 66.67%/81.81% bytes per network packet for requests/responses in GET & SET
  - Enabling efficient data transfer over PCIe when offloading KVS component to DPUs

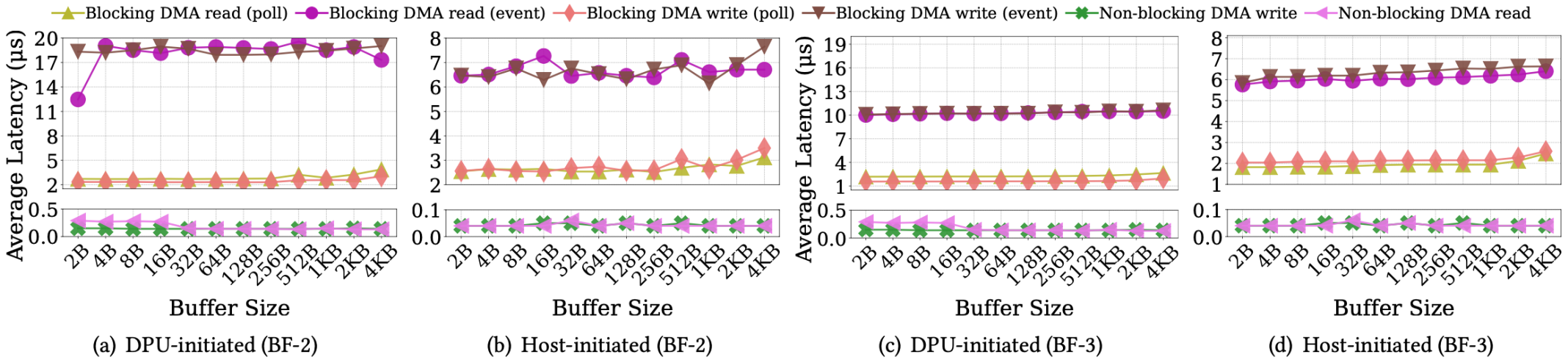


Queue Pair



KV QP reduces data/metadata transfer for KV requests and responses

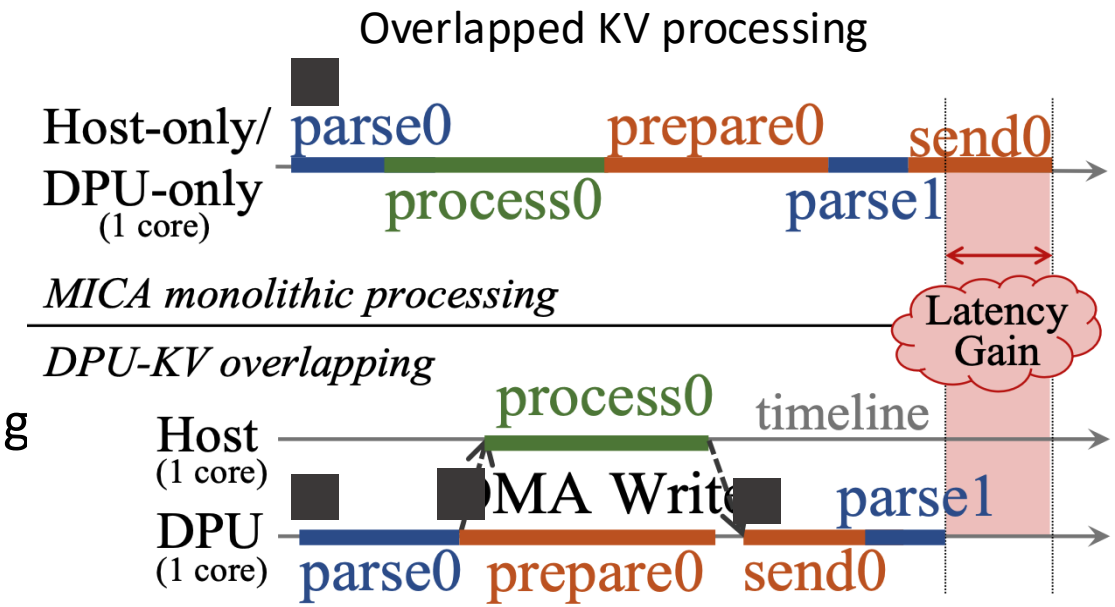
C2 – Efficient Host-DPU Data Path



- DMA APIs enable both DPU SoC and host to use DPU's DMA engine for buffer transfers over PCIe
  - Blocking (Polling or Event-based) vs. non-blocking
  - Initiator of DMA request (host or DPU)
- *Non-blocking* DPU DMA gives low latency
  - Avoids checking completion for each KV request

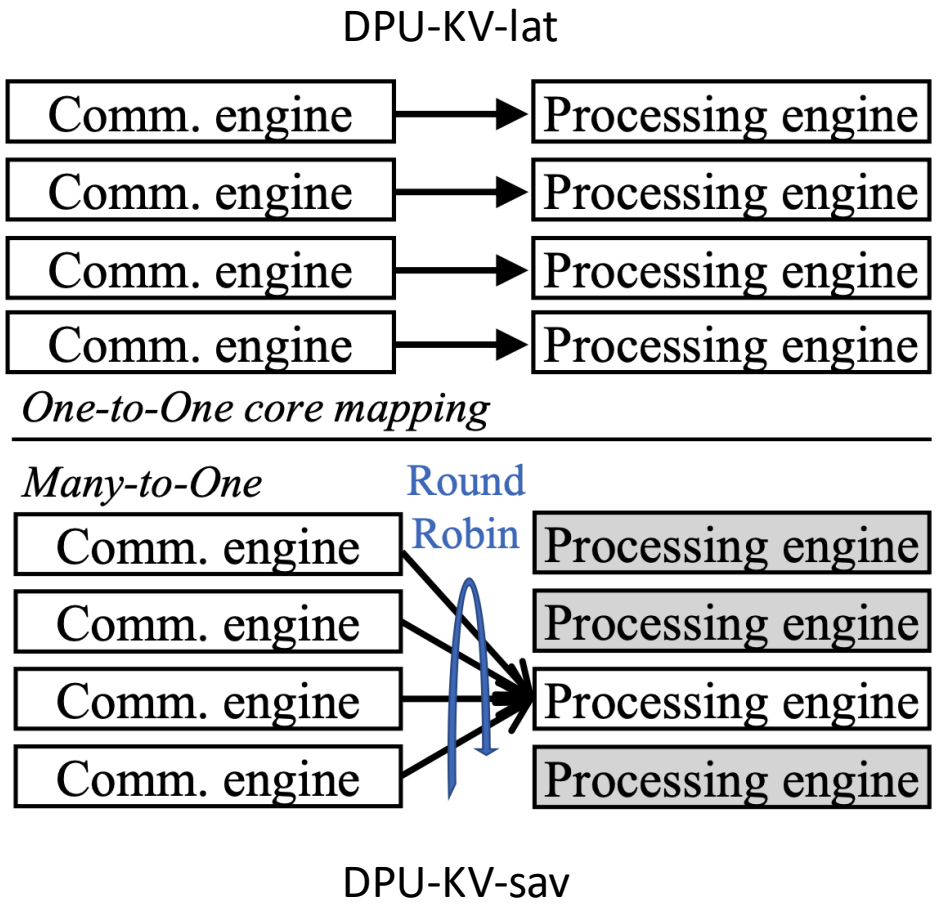
C3 – Hiding Data Movement Penalty

- KVS request processing stages
  - Parse
  - Process
  - Respond
- Optimizing fine-grained KVS offloading (*DPU-KV-lat*)
  - Overlapped KV request/response processing
  - Reducing DMA operations per batch
  - Response processing optimization
- Hides data movement penalty and reduces latency



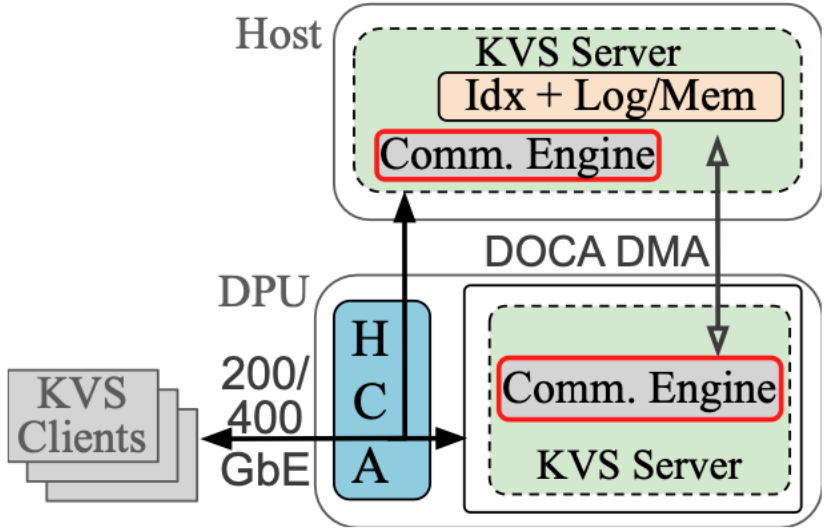
C4 – Reducing Host Resources

- *DPU-KV-lat*
  - One-to-one mapping between host KVS cores and DPU communication cores
    - ❖ Parallel access to individual KV partitions
  - Leads to high host core utilization
- Insight: *Offloading communication engine frees up host CPU cycles*
  - Enables host to handle *more* KV requests with *fewer* CPU cores (*DPU-KV-sav*)
    - ❖ Host processes KV requests in *round-robin*



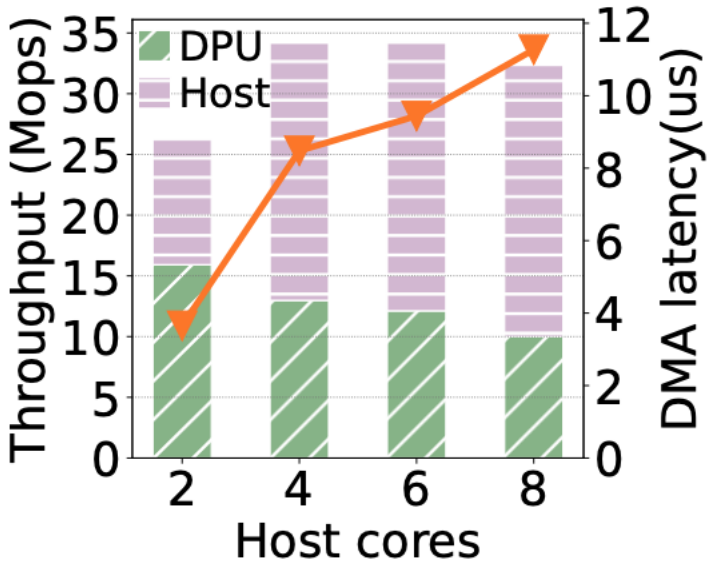
C5 – Performance with Resource Savings

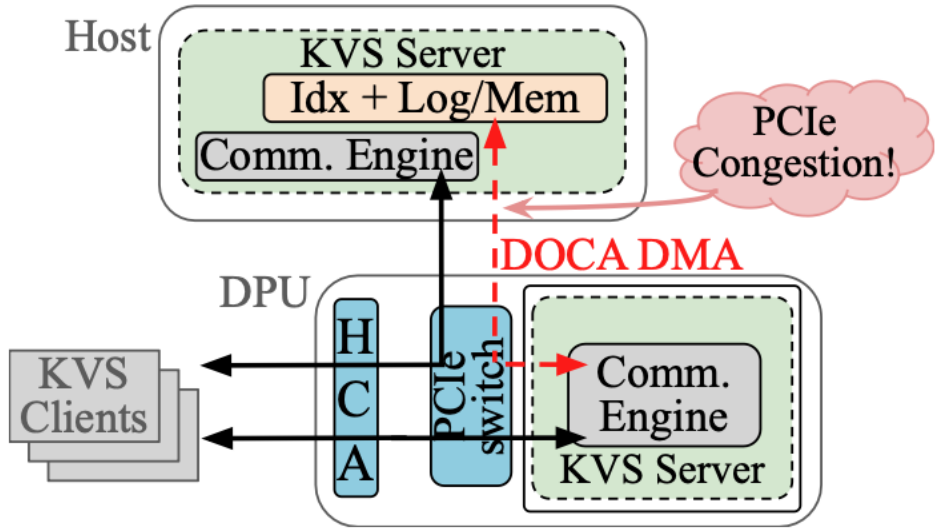
- Throughput of *DPU-KV-lat* and *DPU-KV-sav* (BF-2) > *DPU-only* KVS but < *CPU-only* KVS
- Dual communication engine-based design
- Utilizes DPU cores and spare CPU cores (enabled by C4)
  - One CPU core processes requests from DPUs (main CPU core)
  - Spare cores expose themselves as additional endpoints to client
  - Processes requests for same KVS memory backend



Dual communication engine (DPU-KV-dual)

C5 – Finding Spare Cores in *DPU-KV-dual*





Three host cores (1 main + 2 spare) sufficient

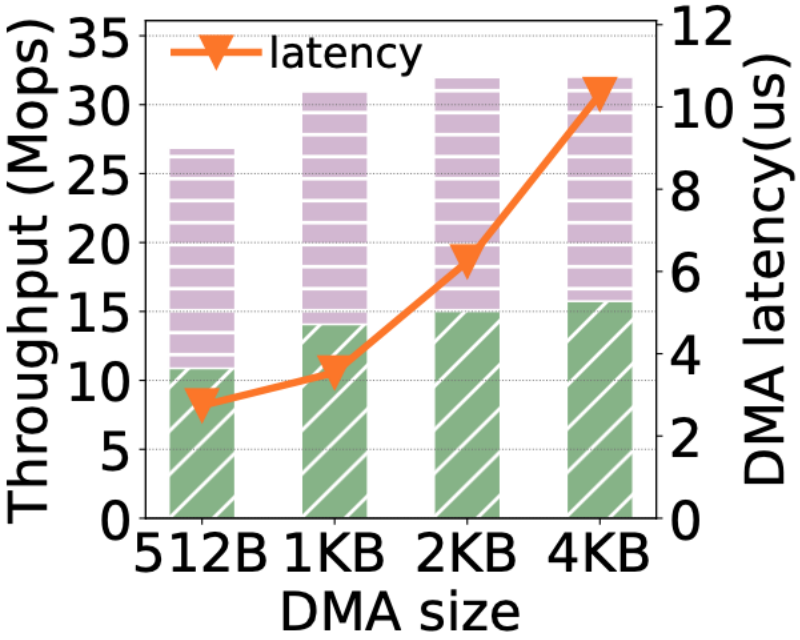
Adding more spares increases CPU use but not throughput due to host PCIe congestion

1. DPU's communication engine sends extracted KV requests to host
2. Host communication engine receives raw client packets

Causes increased PCIe traffic to/from host leading to congestion

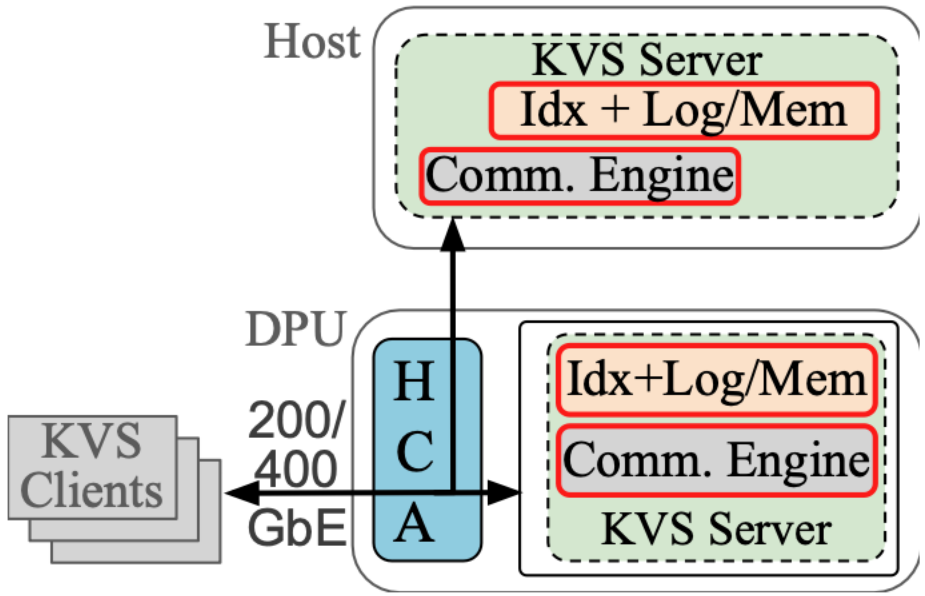
C5 – Host PCIe Congestion

- PCIe congestion on host depends on
  - Concurrent connections with client
  - DMA buffer size
- Measuring per-batch host-DPU DMA latency
  - More PCIe congestion → higher DMA latency
- Total throughput stagnates when DMA size is $> 1KB$
  - DPU throughput increase (4.7%) offset by host's throughput decrease (4.1%) for 2 KB - 4 KB
- *Host-DPU data movement cost not fixed and proportional to host PCIe congestion*



C5 – Mitigating Host PCIe Congestion

- Sharding-based design (*DPU-KV-shrd*)
- KVS partitioned and executed independently on both host and DPU
- Avoids host-DPU communication during KV request processing



DPU Testbed

DPU hardware

DPU	Mode	Compute	Memory	Port Speed	DOCA SDK	OS
BlueField-2 DPU	Separated Host	8 core A72 @ 2.5GHz	16GB DDR4	200 Gbps	v1.5	Ubuntu 20.04
BlueField-3 DPU	DPU	8 core A78 @ 3GHz	16GB DDR5	400 Gbps	v2.5	Ubuntu 22.04

- Allows keeping rest of the hardware and major portion of software setup same except for choice of DPU
- Comparison of BF-2 and BF-3 shows generational hardware impact
  - *BF-2 < Edge Host < BF-3*

Implementation Tips

- *DPU-KV* prototyped with MICA2 KVS
- Connection management
  - MICA uses *etcd*
  - DPU registers endpoints with *etcd*
    - ❖ *No code changes are required for the MICA2 KVS client*
- Porting to ARM architecture
  - DPDK APIs
  - x86 instructions

	MICA2	**Ported MICA2**
DPDK APIs	rte_eth_dev_filter_ctrl	rte_flow_create rte_flow_validate
x86 instructions	PAUSE RDTSC	YIELD CNTVCT_EL0

Implementation Tips

- Timestamp
  - Enable latency tracking by timestamping each KV request at client
  - Server echoes timestamp back in KV response
- DOCA DMA
  - Proposed *Key-Value QP* design uses DPU's DMA engine for host-DPU KV data exchange
  - DMA APIs available via DOCA SDK
    - Variation between DOCA SDK versions used by our BF-2 and BF-3
    - *doca_pe* added task completion callback, requiring DMA buffer length reset before reuse

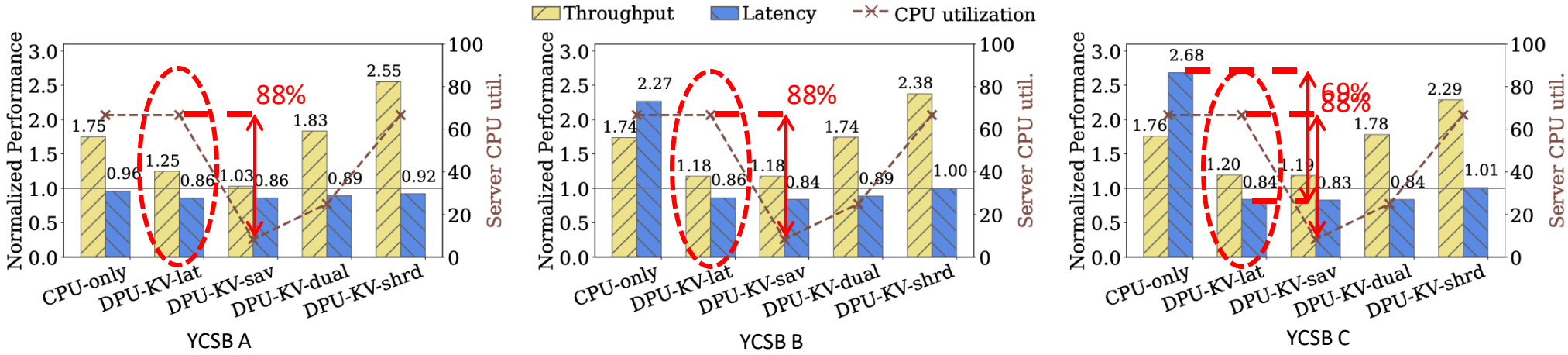
DMA API	BF-2	BF-3
Hardware initialization	doca_workq	doca_pe
DMA job submission	doca_workq_submit	doca_task_submit
Completion status check	doca_workq_progress_retrieve	doca_pe_progress

Evaluation

- *DPU-KV* (KVS server)
  - Each key space partition assigned to single core
    - ❖ Utilizes all DPU cores
    - ❖ CPU cores usage varies based on design
  - *No* modification to KVS's memory allocator and indexing schemes (processing engine)
  - Pre-allocate/register DMA buffers per core

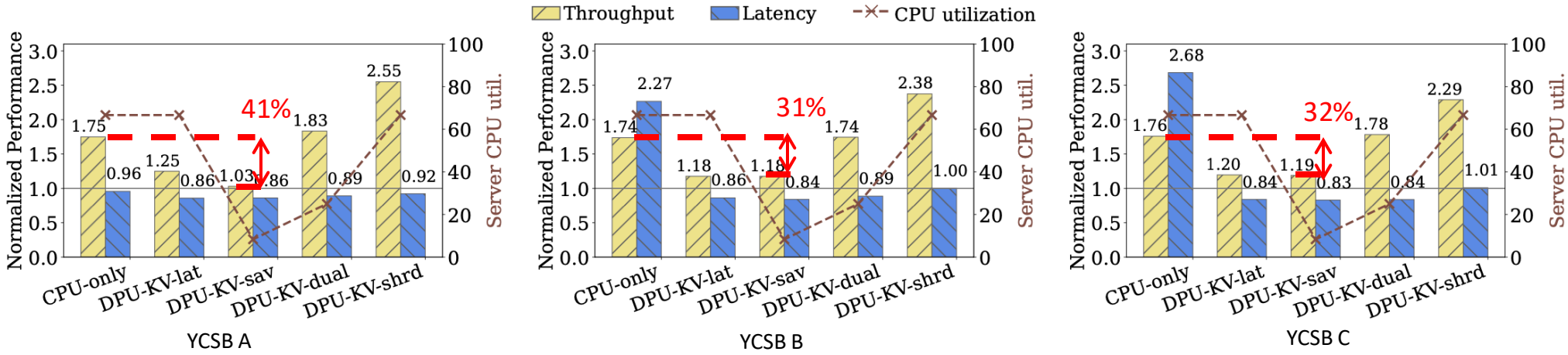
KVS designs	Core idea/principle	Host cores	DPU
	Baseline		
CPU-only	SOTA Ethernet-based KVS, MICA	8	
DPU-only	Coarse-grained KVS offloading	0	BF-2/3
	Fine-grained KVS offloading		
DPU-KV-lat	Communication engine offload	8	BF-2/3
DPU-KV-sav	One-to-many CPU-DPU core mapping	1	BF-2/3
DPU-KV-dual	Dual communication engine	3	BF-2
DPU-KV-shrd	KV sharding	8	BF-2

DPU-Offloaded KVS Performance with BF-2



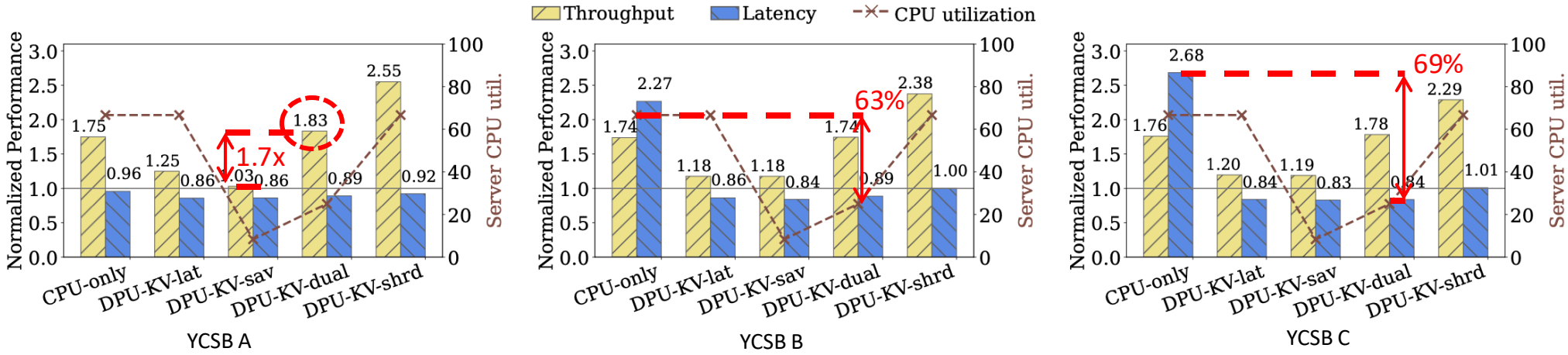
- *DPU-KV-lat* reduces latency by up to 16% and 69% over *DPU-only* and *CPU-only*
  - Communication engine offloading, overlapped KV request/response processing, reducing DMA operations per batch, etc.
- *DPU-KV-sav* reduces server CPU utilization by ~88% compared to *CPU-only* and *DPU-KV-lat*
  - One-to-many processing-to-communication engine core mapping
  - *DPU-KV-sav* matches *DPU-KV-lat* performance with fewer host cores
    - Freed host cycles via offloading can be used to run the non-offloaded KVS component (processing engine) on fewer host cores
  - *DPU-KV-lat* optimizations carry over effectively to *DPU-KV-sav*

DPU-Offloaded KVS Performance with BF-2



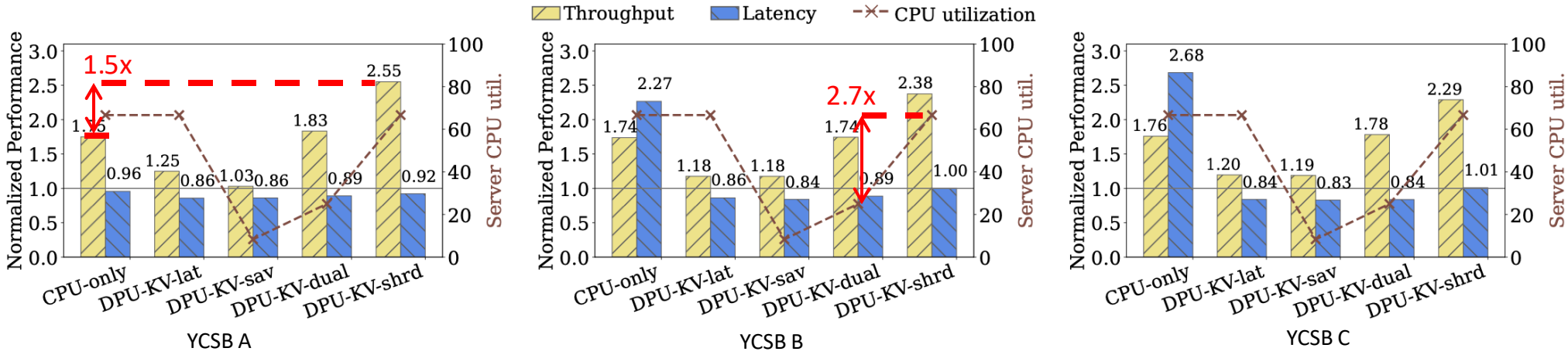
- *DPU-KV-sav* trades 32%–41% throughput compared to *CPU-only* for maximum host resource savings
  - BF-2's wimpy SoC cores cannot fully saturate host's KVS engine

DPU-Offloaded KVS Performance with BF-2



- *DPU-KV-dual* achieves up to 1.83x and 1.7x higher throughput over *DPU-only* and *DPU-KV-sav*
  - Throughput boost by utilizing spare host cores (freed by *DPU-KV-sav*) to handle extra KV requests
- *DPU-KV-dual* reduces latency by up to 69% and host core usage by 63% compared to *CPU-only*
  - Delivers slightly higher throughput than *CPU-only* (up to 1.07x)
- DPU-KV-dual's latency reduction from main host core handling DPU requests, reusing DPU-KV's latency optimizations

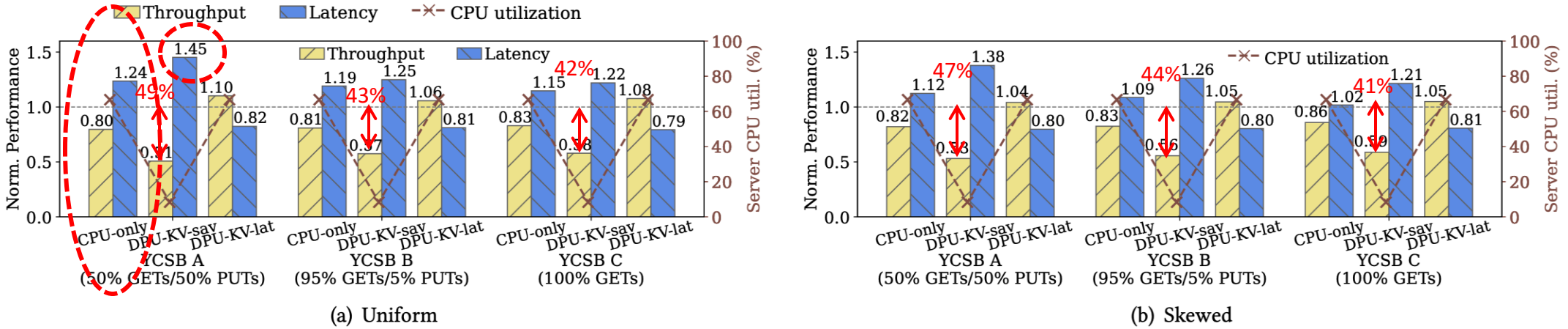
DPU-Offloaded KVS Performance with BF-2



- *DPU-KV-shrd* consumes 2.7x more host cores than *DPU-KV-dual*
  - Achieves up to 1.5x higher throughput than *CPU-only*

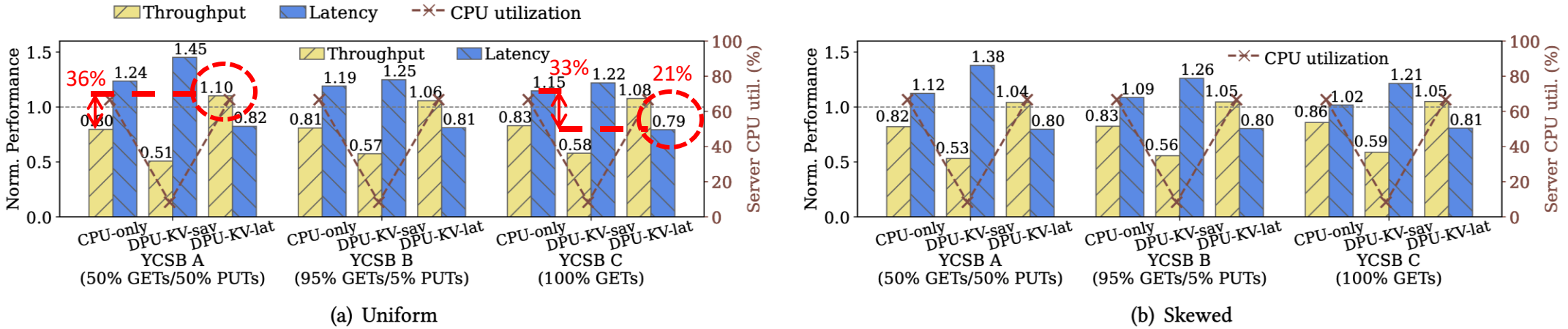
- Use *DPU-KV-shrd* to achieve high throughput
- *DPU-KV-dual* **ideal** for edge environments with limited host resources
  - Achieves lower latency and CPU-like throughput while saving host resources compared to *CPU-only* KVS

DPU-Offloaded KVS Performance with BF-3



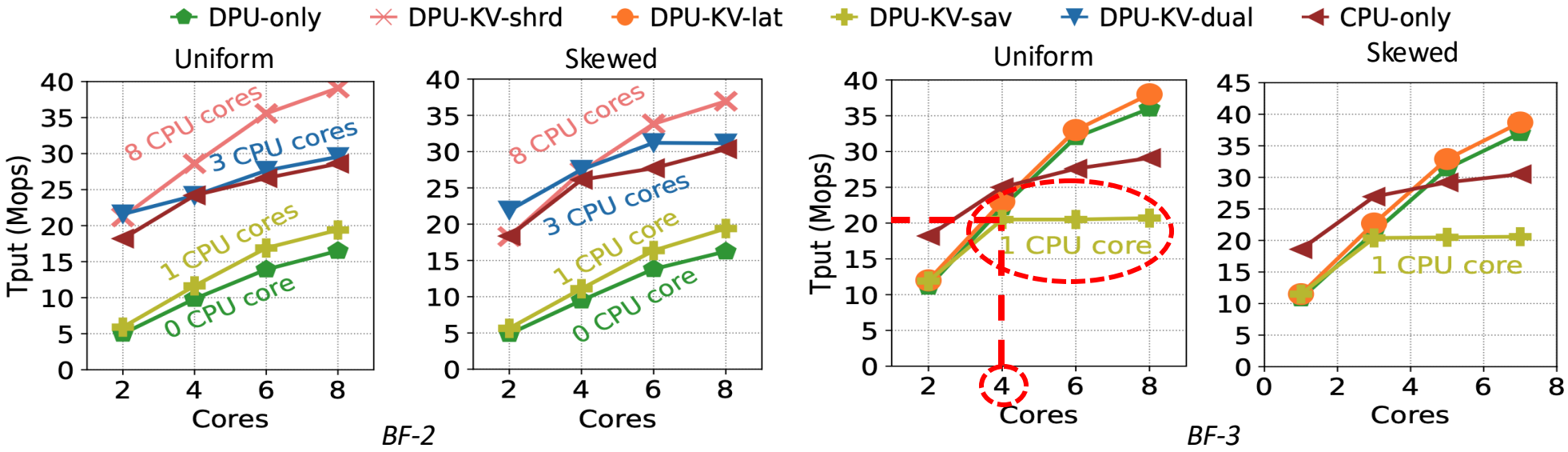
- *CPU-only* exhibits lower performance than *DPU-only*
  - *CPU-only* has 20% lower throughput and 24% higher latency than *DPU-only*
  - Brawnier SoC in BF-3 enables higher KV request processing than edge host
- *DPU-KV-sav* shows up to 49% lower throughput and 45% higher latency than *DPU-only*

DPU-Offloaded KVS Performance with BF-3



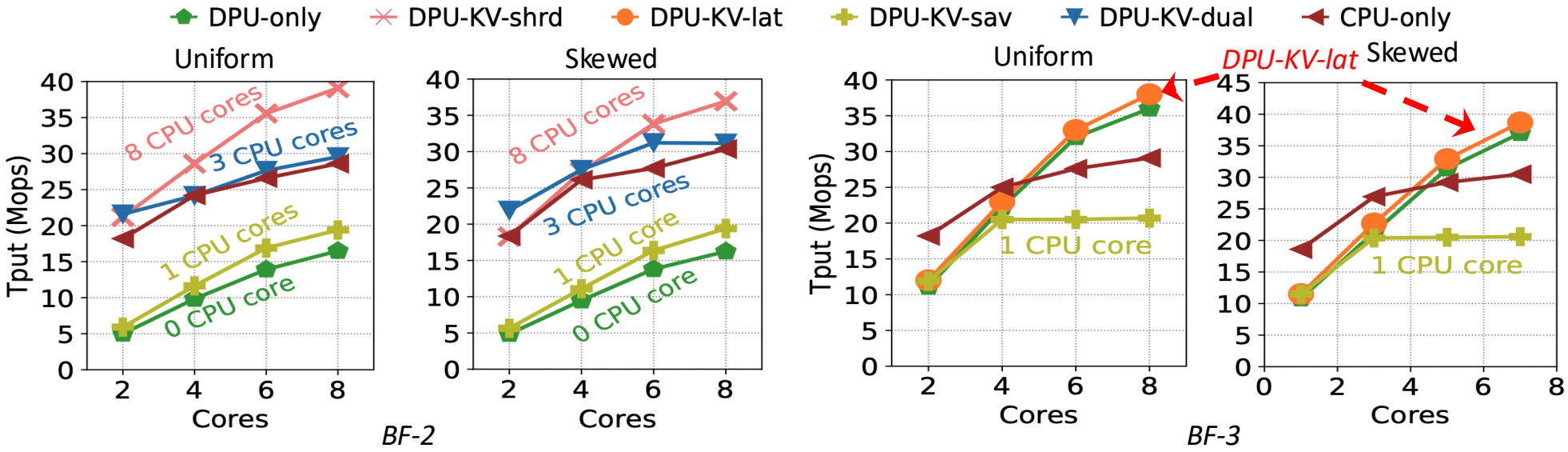
- *DPU-KV-lat* shows 33% and 21% lower latency, along with up to 36% and 10% higher throughput, compared to *CPU-only* and *DPU-only*
- Similar to BF-2, offloading KVS communication engine and optimizing latency
  - Fewer DMA operations, overlapped KV processing, and response optimization
- *DPU-KV-lat* can help edge applications using KVS achieve low latency and high throughput

Scalability Evaluation



- *DPU-KV-lat*, *DPU-KV-sav*, *DPU-KV-dual*, and *DPU-KV-shrd* throughput scales with DPU cores for BF-2
- With BF-3, *DPU-KV-sav* saturates at 20.5 Mops
  - Four BF-3 ARM cores enough to saturate processing engine on one host CPU core
  - Unlike *DPU-KV-sav* on BF-2

Scalability Evaluation



- DPU-KV-lat performance scales linearly
  - 1:1 core allocation between communication and processing engines
- Efficient <u>host-DPU data path enables linear scalability</u> of DPU-offloaded KVS with limited PCIe traffic

Summary

Problem

Edge key-value stores demand low latency and/or high throughput—requirements that existing edge KVS solutions or **coarse-grained KVS offloading** to DPUs at edge fail to meet

Objective

Assess the performance benefits of **fine-grained KVS offloading** over coarse-grained offloading and CPU-based KVS

Key Idea

Identify and offload **most CPU-intensive component** (KVS communication engine) to DPU and minimize **host-DPU data movement overheads**

Key Contributions

Proposed **DPU-KV**, a novel DPU-offloaded KVS that:

- Modularizes KVS and explores various fine-grained KVS offloading architectures
- Eliminates up to **83%** of metadata transfer for KV data using queue pair model
- Reduces latency by up to **68%**, host core usage by **63%**, and improves performance by **36%** compared to CPU-only and coarse-grained KVS offloading to DPUs

Thank you!

http://padsys.org/

UNIVERSITY OF CALIFORNIA
MERCED