An Operating System for RISC-V based SSD Controllers

Philippe Bonnet -- DIKU, bonnet@di.ku.dk

Ivan Luiz Picoli, Malthe Larsen, Lucas Bürgi, Nathan Frison(*),
Elias Coppens(*), Nicklas Björkeroth, Alberto Lerner (**)

(*) ENS Paris
(**) Computing Flows, GmbH

Funded by the European Union

CHORYS

https://www.youtube.com/watch?v=vAi5VQtN2bM

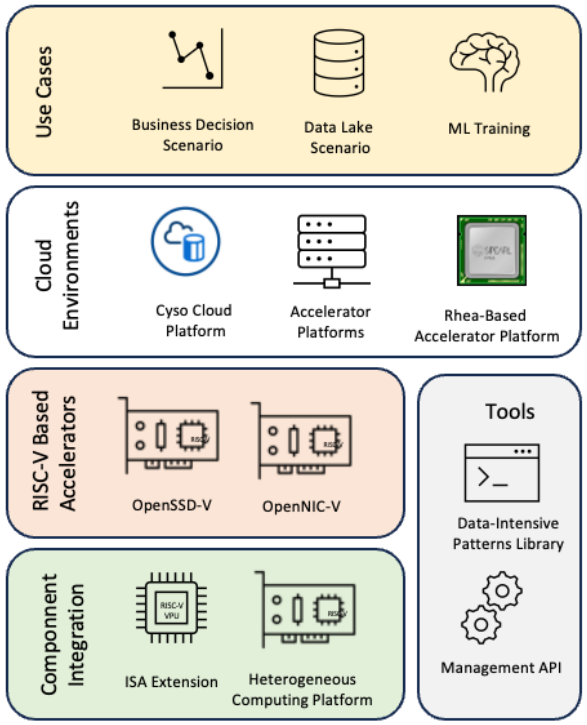
Open Source for Cloud/Edge to support European Digital Autonomy

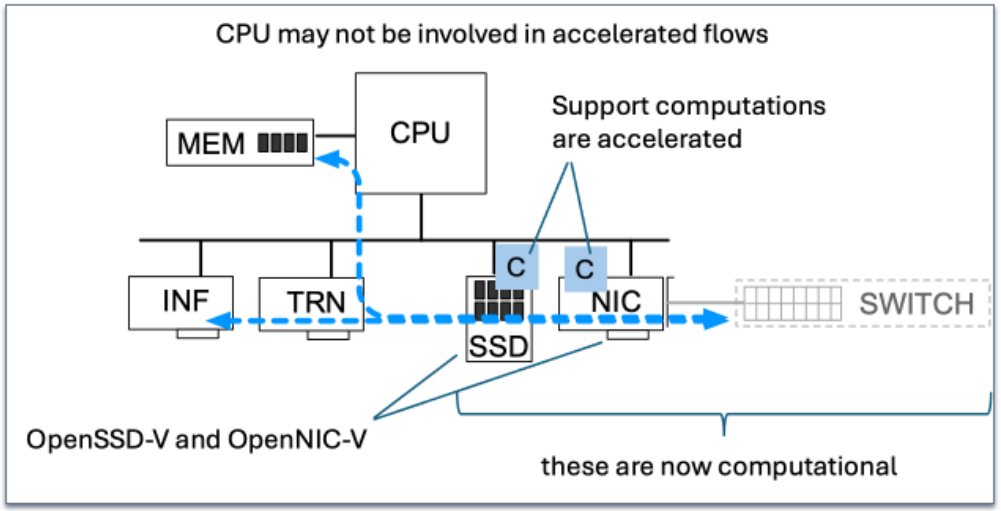
- Horizon Europe Projects: HORIZON-CL4-2024-DIGITAL-EMERGING-01-21
  - Prototypes of cloud and edge servers demonstrated in relevant centralized and distributed environments and allowing full computing infrastructure deployments based on European processor technology, thereby establishing a full Open Computing Architecture stack, which supports emerging processing architectures (e.g. RISC-V).
  - Standards and best practices consolidating the European Open Computing Architecture, as well as its interfaces to current industry standards.
- Three projects: HIGHER, CAPE and CHORYS

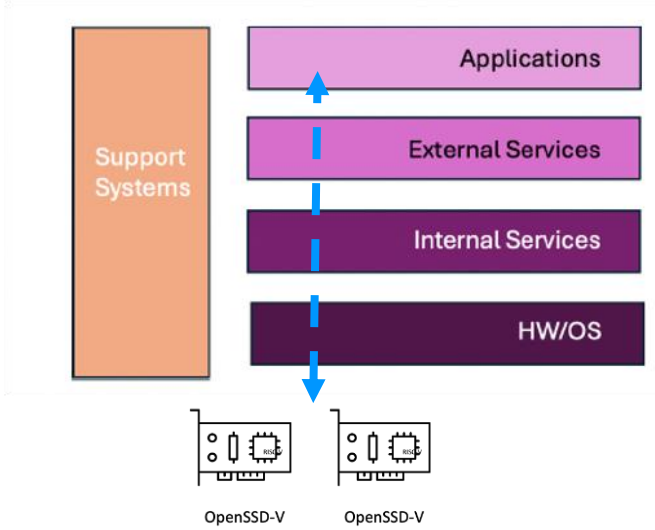
CHORYS

Project website: https://chorys.eu

Project coordinator: UCPH
https://www.diku.dk

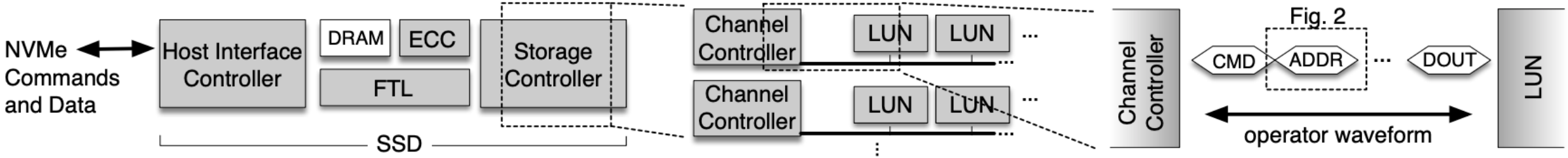


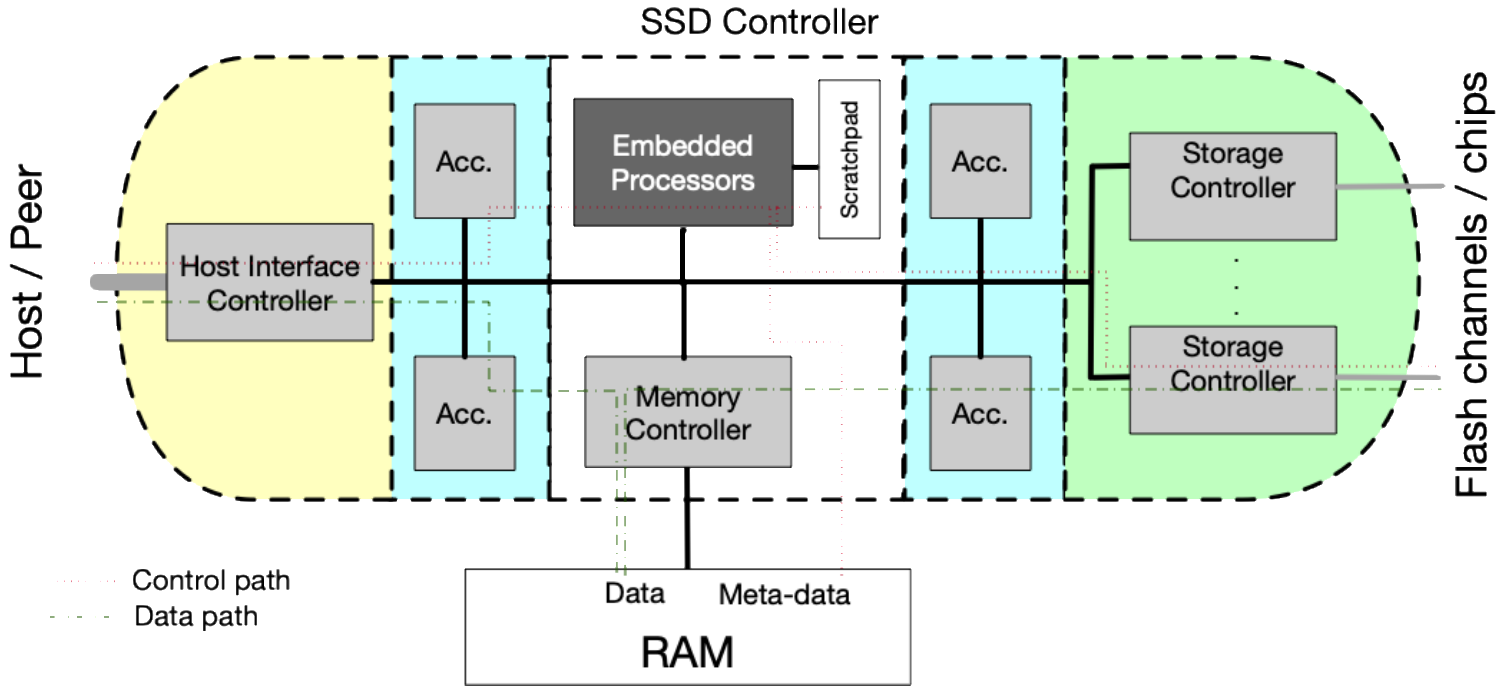


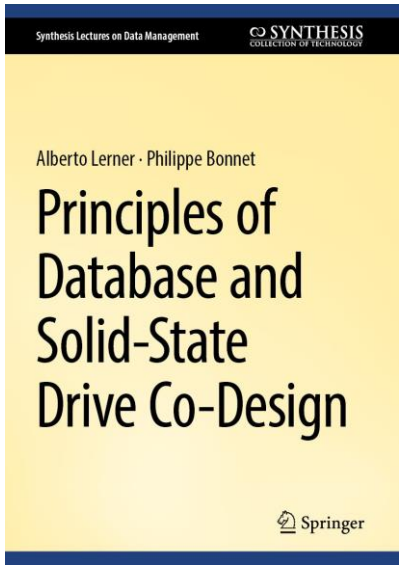


Open and Programmable Accelerators for Data-Intensive Applications in the Cloud
Turning devices that move data, i.e., Solid-State Drive (SSD) and Network Interface Card (NIC) into **Near-Data Processing (NDP) accelerators** based on RISC-V cores.

What is an SSD?







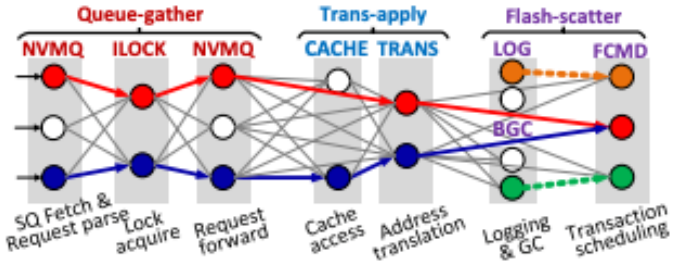
The Case for Programmable SSDs

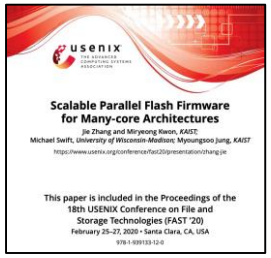
1. Leverage end-to-end perspective for trade-off between cost, sustainability and performance
2. Avoid designing systems against/around arbitrary or hidden SSD constraints
3. Actually minimize data movement

We posit that SSD controllers should be equipped with an operating system, operating at microseond-scale, that abstracts IP sensitive hardware components and supports a programming model for SSD applications, including but not restricted to the conventional FTL.

The Case for an SSD Operating System

Embedded operating systems are already used in the context of existing multi-threaded firmware and prototype programmable SSDs.





The staged pipeline model improves scalability in multi-core SSD firmware.

It introduces fundamental limitations:

- **Interferences**: Threads within a stage share mutable data structures, requiring careful and brittle synchronization
- **Platform dependence**: The model embeds platform-specific assumptions—such as cache-coherent memory and OS thread scheduling—directly into the firmware logic
- **Opacity**: Trade-offs between the number of stages, the number of threads per stage, the properties of the memory hierarchy, and the

The Case for a new SSD Operating System

Property	Embedded Linux [3]	FreeRTOS [12]	µC/OS-II [27]
Multiprocessing	Thread-based; symetric multi-processing across many cores	Task-based; symetric multipro-cessing across few cores	Single-core only
Footprint	≥8 MB RAM, ≥32 MB flash	~2–5 kB RAM, ~5–15 kB flash	~4 kB RAM, ~6–24 kB flash
Evolution	Device driver model; device tree	No driver model	No driver model
Testability	No deterministic scheduler	deterministic preemptive sched-uler; portable layer for host sim-ulation	deterministic scheduler
Memory Model	Virtual memory with MMU	Flat physical address space	Flat physical address space
Need for Atomics	atomics required	atomics required	mutual-exclusion through global interrupt disable
Predictable Timing	>10 µsec across thread exe-cutions or between interrupt and thread execution with PRE-EMPT_RT	millisecond-level tick resolu-tion; sub-µsec interrupt latency	worst-case execution time (WCET) analysis; µsec latency

Design

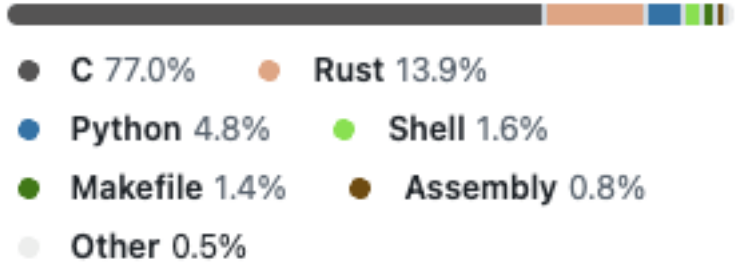
Opinionated OS design

1. The execution model should structurally prevent interference across concurrent requests, rather than relying on ad-hoc synchronization.
2. The execution model should support the description and management of flows of data across operations, as SSD firmware fundamentally processes streams of requests through sequences of data movements.
3. The memory model should make platform assumptions explicit, so that the same firmware can be deployed on platforms with different memory hierarchies without re-design.

ssd_os Abstractions

- ssd_os supports a range of platforms: For each platform, a collection of drivers is defined for accelerators and peripherals. **Each platform defines a compilation target** with a well-defined RISC-V ISA and ABI.
- ssd_os provides **region-based memory management.** Regions are defined statically, at compile time. At run-time, programs manage memory through an application-specific allocator.
- An application is a **tagged-token dataflow graph**, where each vertex is a task and each edge is a directed channel that carries a token. More specifically, application is a forest of multi-

ssd_os Implementation




```c
typedef void *tag_context;
typedef void *(tag_function) (void *)

typedef struct {
    tag_function *function;
    tag_context context;
} tag;

typedef struct {
    int16_t edge_idx;
    tag_context context;
} task_output;

typedef void ( init_fn ) ( void ) ;
typedef task_output ( task_fn ) (tag);
typedef tag ( poll_fn ) ( void ) ;

struct task {
    uint8_t type; //e.g., source, stage
    uint16_t id;
    init_fn *init_fn;
    poll_fn *poll_fn; // null for stages
    task_fn *task_fn;
    uint8_t num_out_edges;
    struct task *out_edges;
};
```

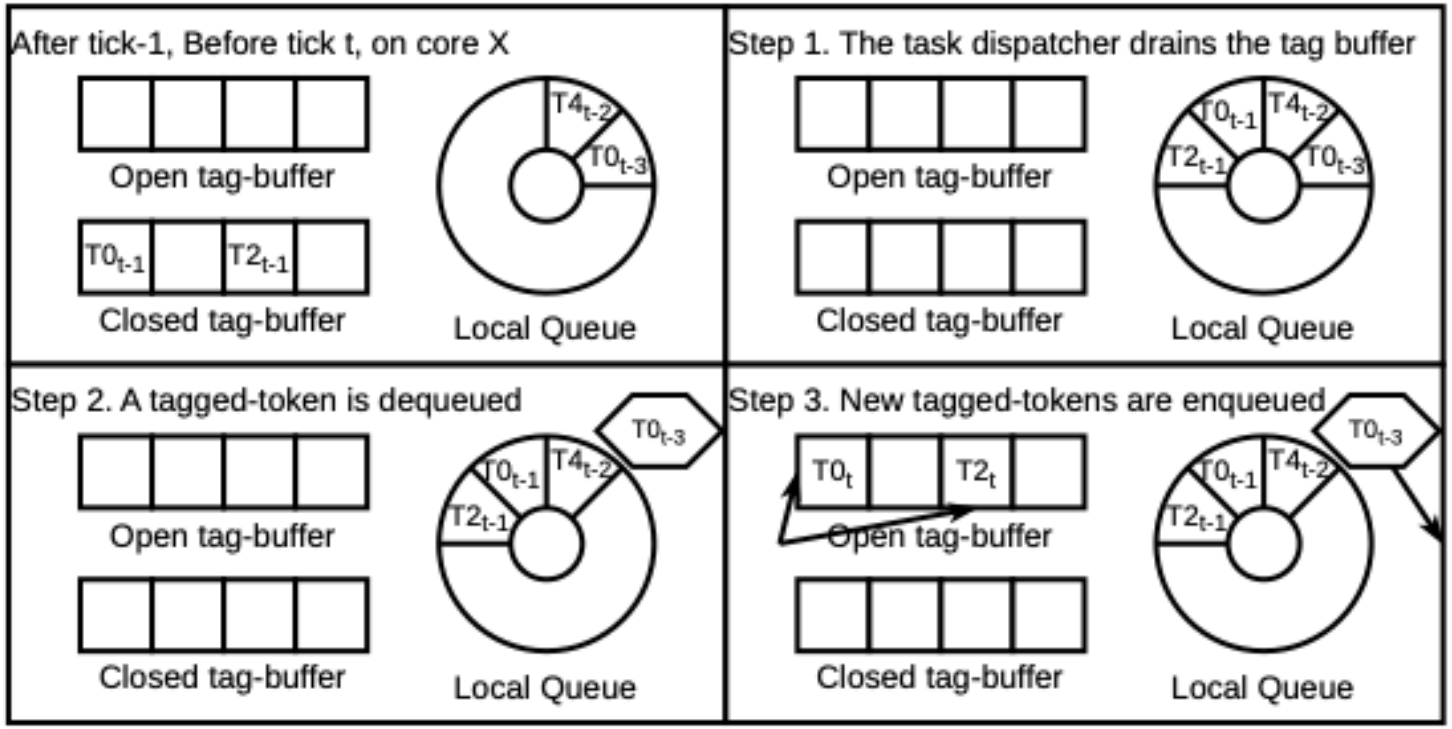
- Ssd_os is written in C and packaged as a li
  16K SLOC of C (and 800 SLOC of assembly).
- An application is a collection of relocatabl
  that complies with the programming model, ar
  any language that is ABI compatible with C (
- At deployment time, instructions and core-lo
  structures fit in 128 KB, while shared data
  16KB.
- ssd_os runs on both 32-bit and 64-bit versio

Execution Modes

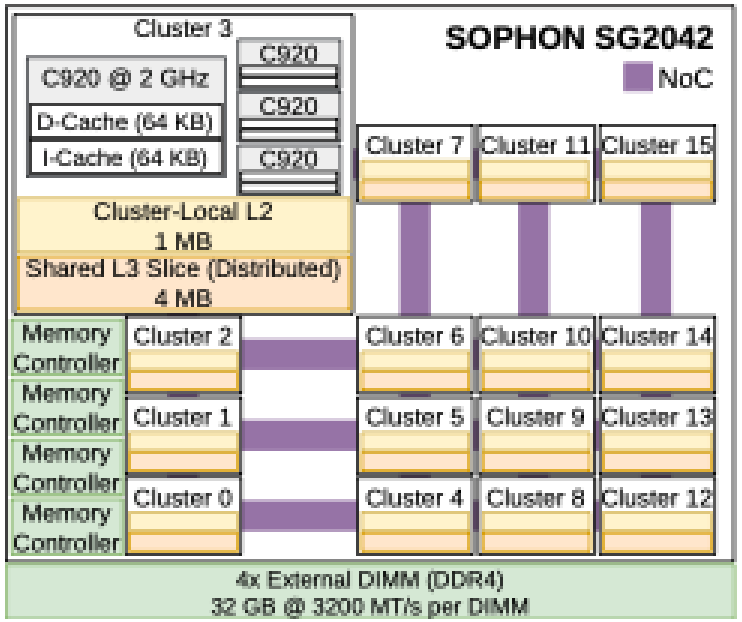
Three execution modes:

- **Best-Effort**: underlying coherent platform, supporting atomics to provide traditional synchronization for efficient token exchange, without guarantees.
- Deterministic No assumptions about the underlying platform. This execution model guarantees a deterministic history of task activations. Tick system, designed to improve testability.
- Synchronous mode: Extends the deterministic tick system so that all ticks have the same fixed-length duration. The goal of such a synchronous system is to support worst case execution time analysis on a predictable platform.

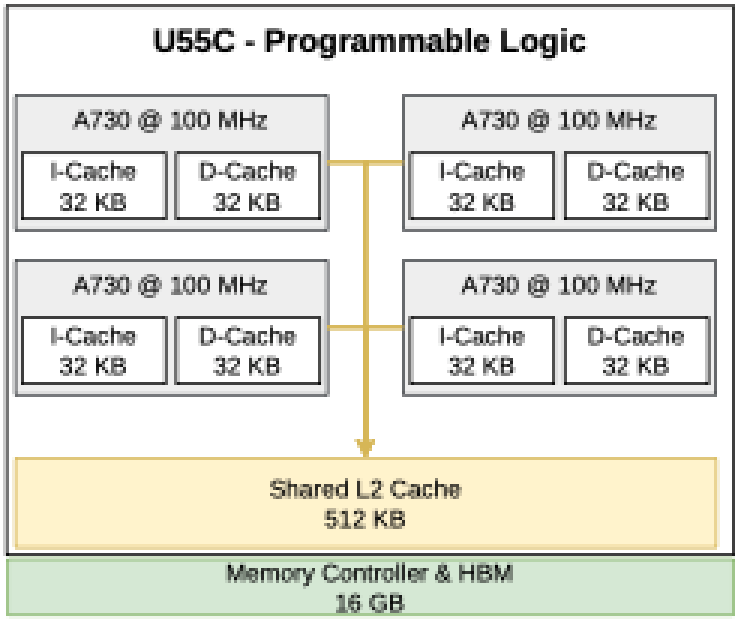
Tag Buffers-Based Scheduling



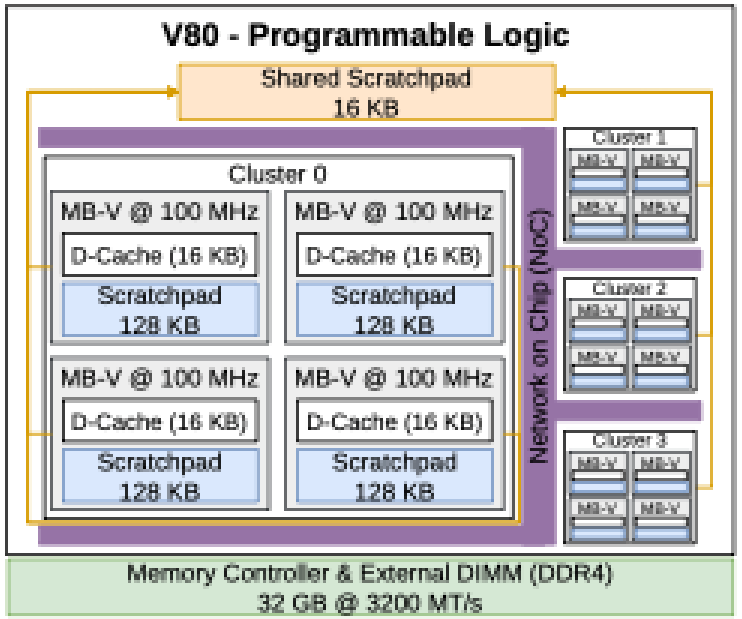
Experimental Platforms



(a) Milk-V

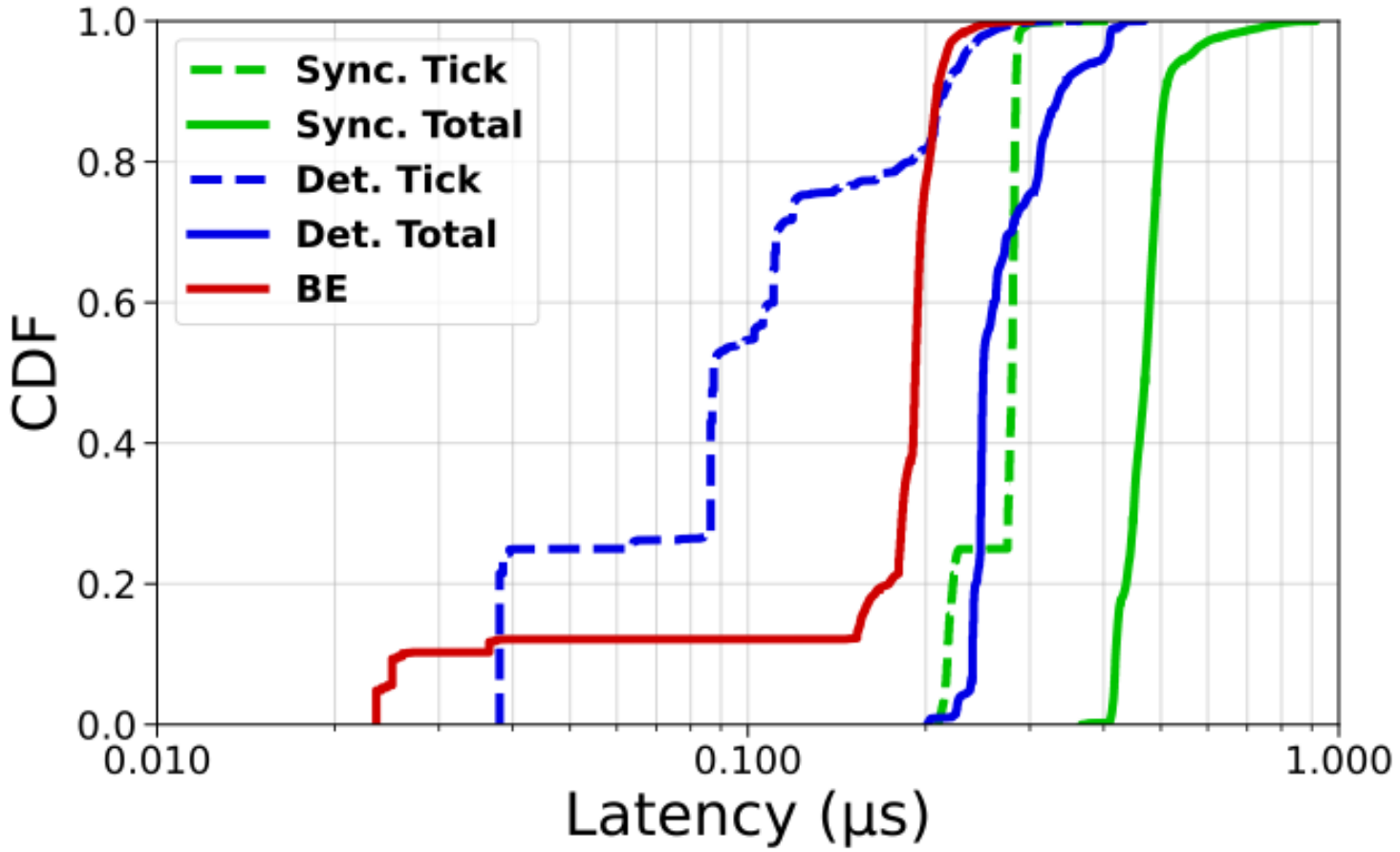


(b) A730

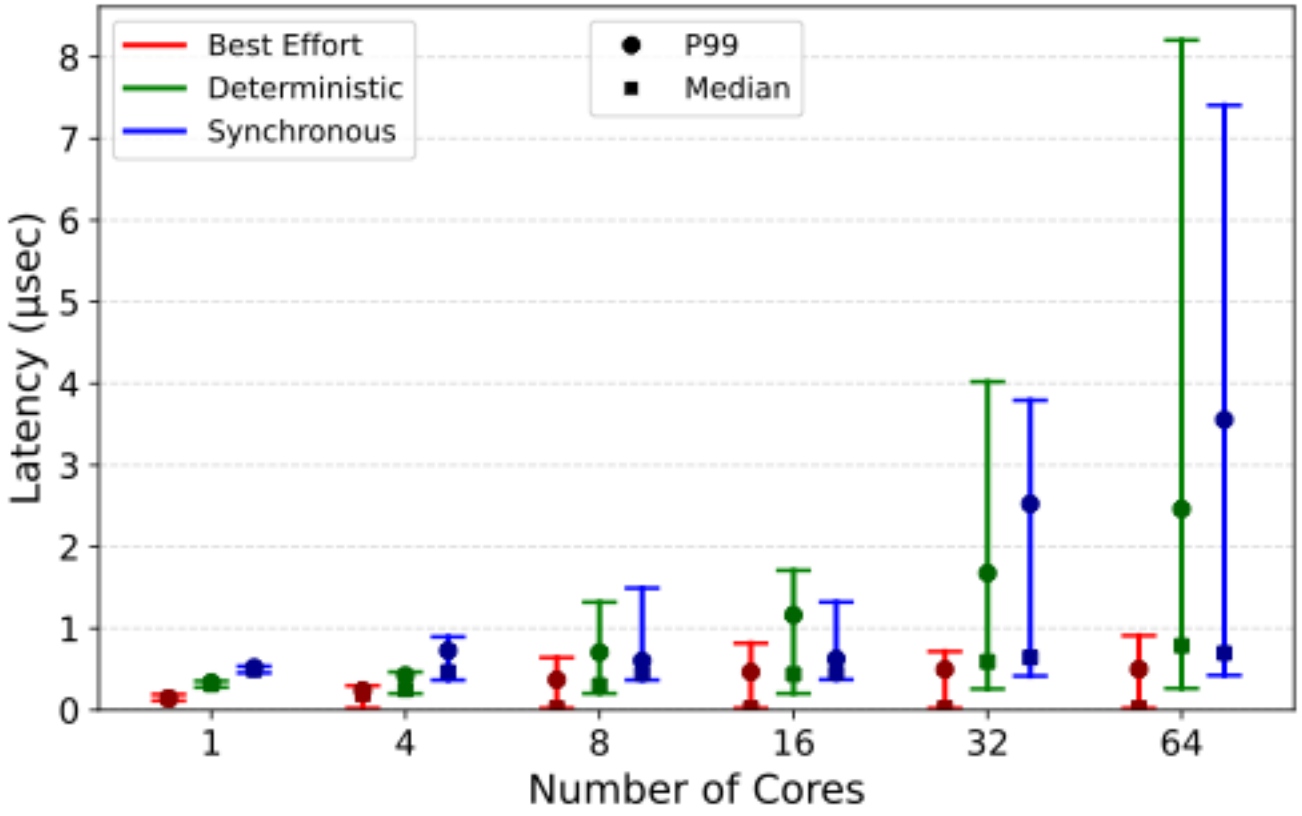


(c) MB-V

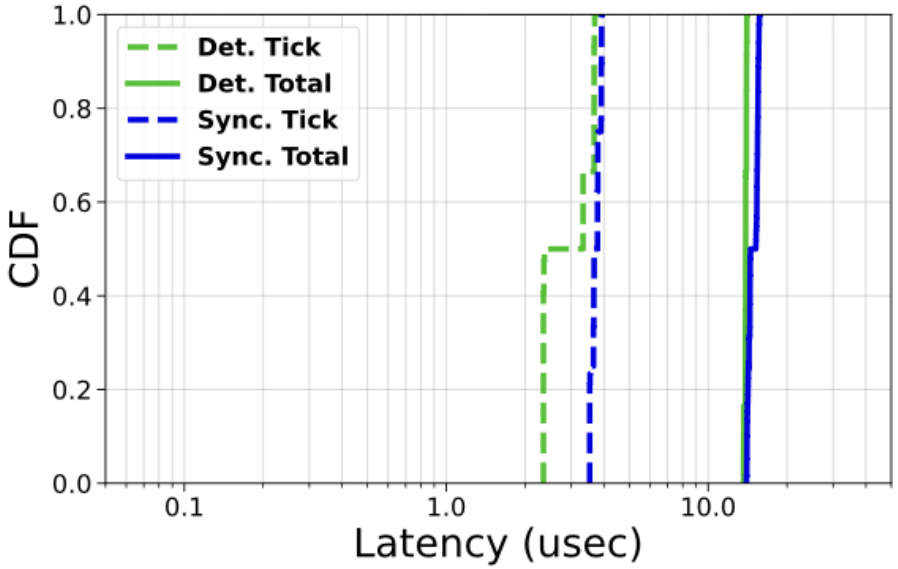
Task transition performance #1 [Milk-V]

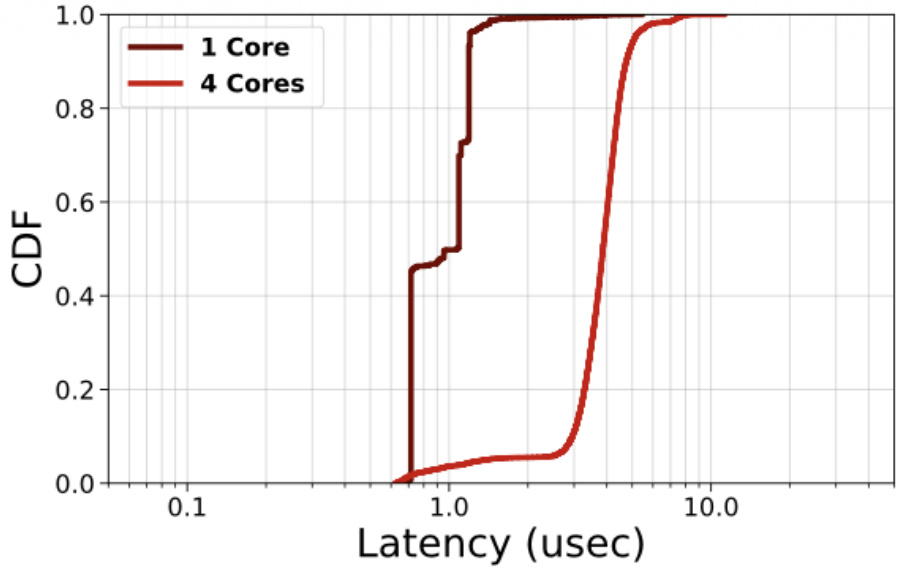


Task transition performance #2 [Milk-V]

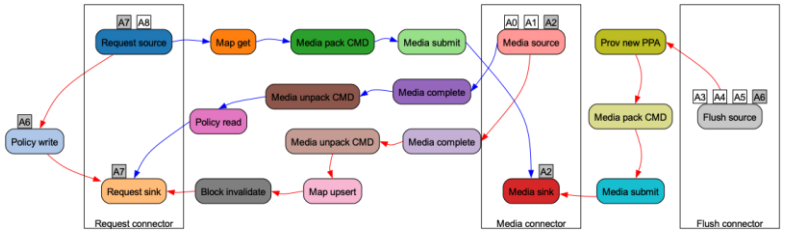


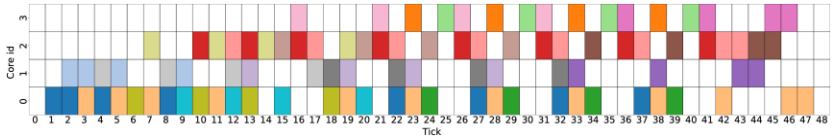
Task transition performance [MB-V, A730]

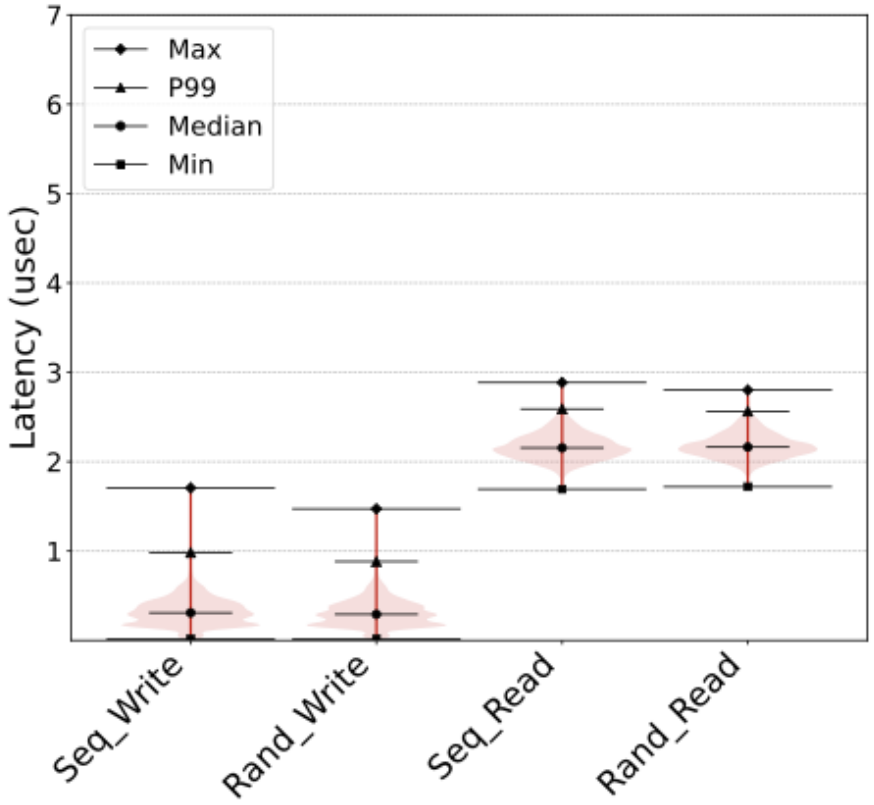


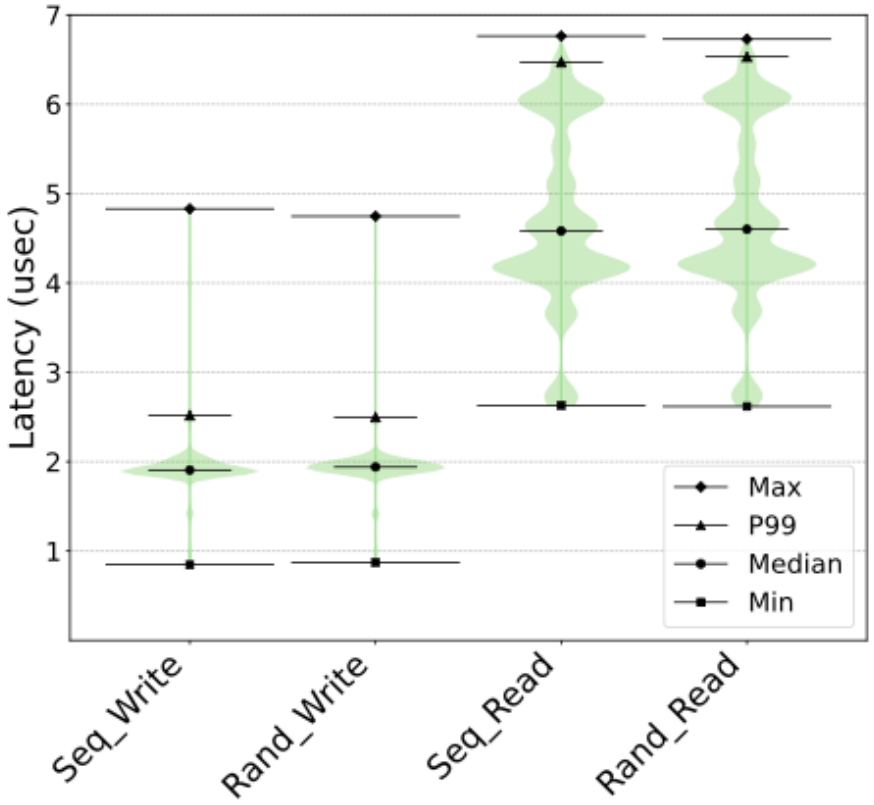


FTL Latency [Milk-V]









Conclusion

- New multiprocessing, real-time operating system for RISC-V aimed at dataflow processing
  - Execution modes adapted to coherent [with atomics] and non-coherent memory systems [without atomics]
  - Micro-second scale tick system
  - Support for deterministic simulation testing
- Future work
  - More application programs (complete FTL, Parquet SSD, …)
  - End-to-end integration within OpenSSD-V
  - New platforms