Transactional Memory for Processing-In-Memory: A Software-based Implementation for the UPMEM platform

Paolo Romano

TÉCNICO LISBOA

EMERALD 2024

PRR
Plano de Recuperação e Resiliência

REPÚBLICA PORTUGUESA

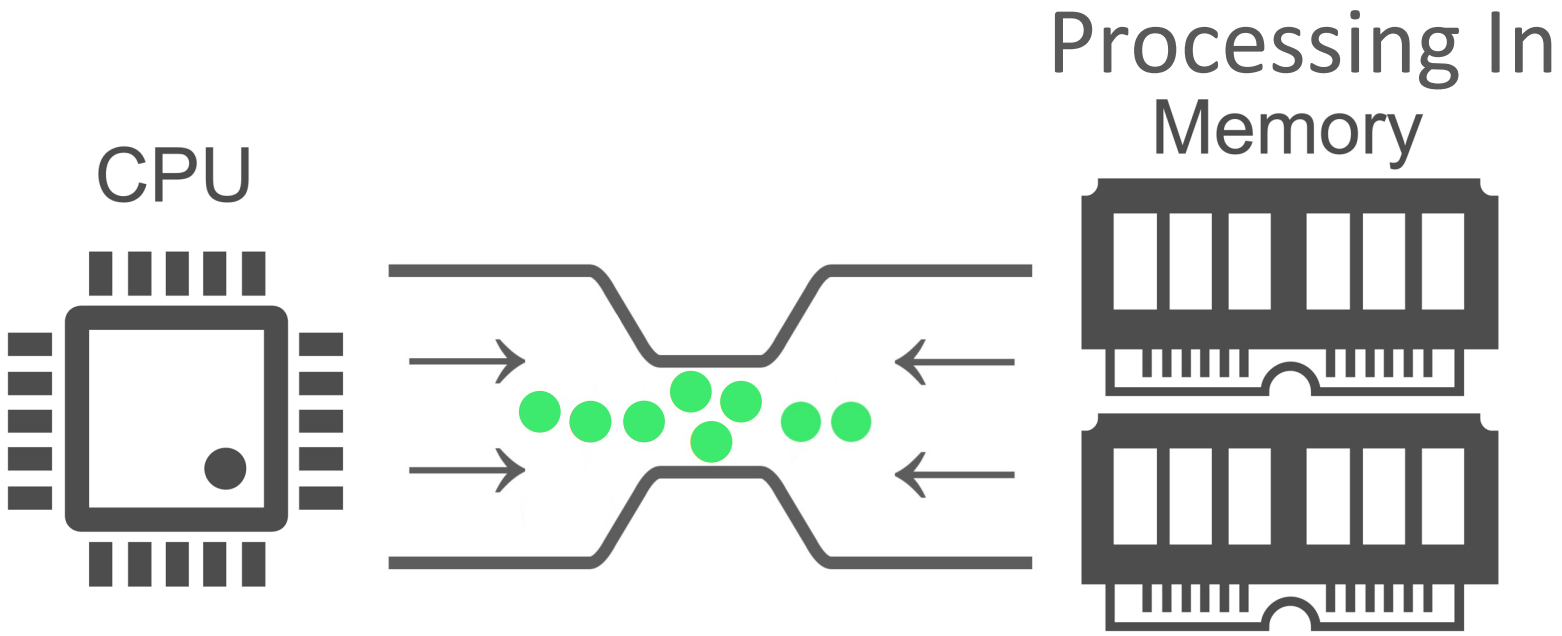
Financiado pela União Europeia
NextGenerationEU

Joint work with

André Lopes
PhD Student

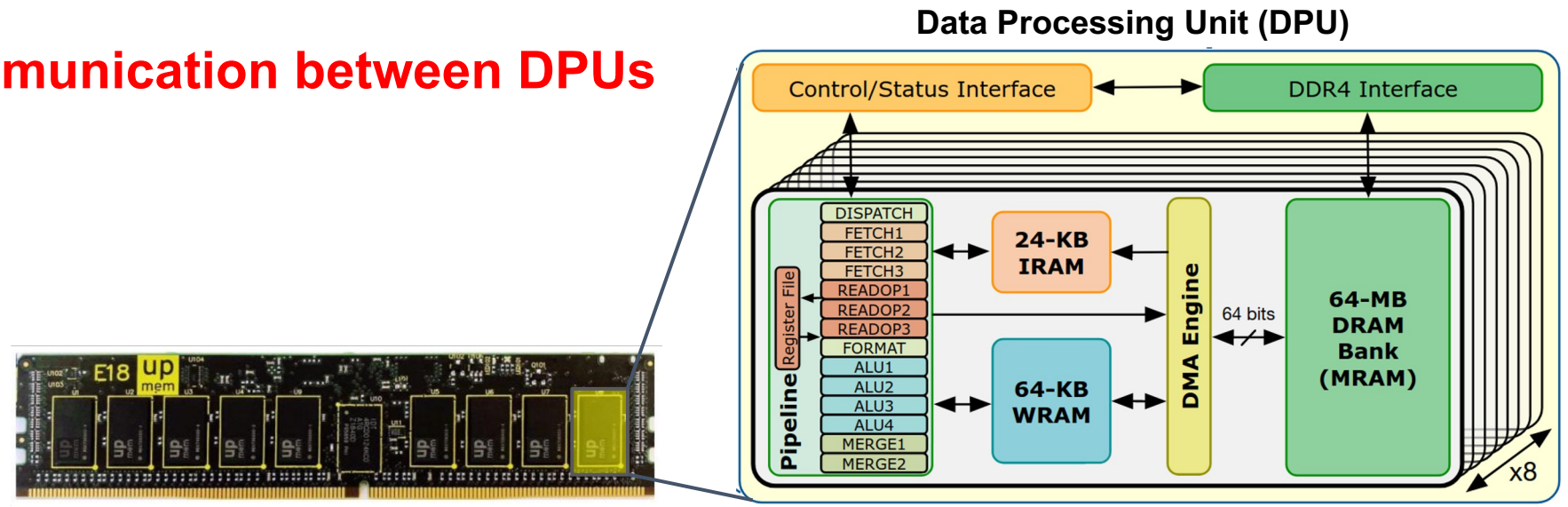
Daniel Castro
Postdoc

Data movement bottleneck



UPMEM PIM Hardware

- Commercially available PIM system
- Each DPU has **24 hardware threads**
- 2560 DPU ➔ > **60K threads** in total!
- **No direct communication between DPUs**

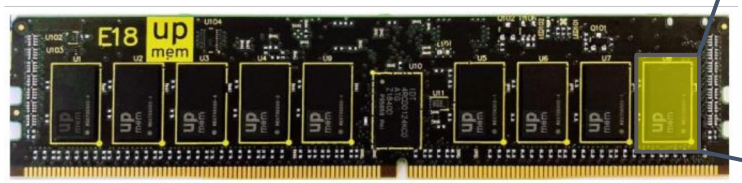


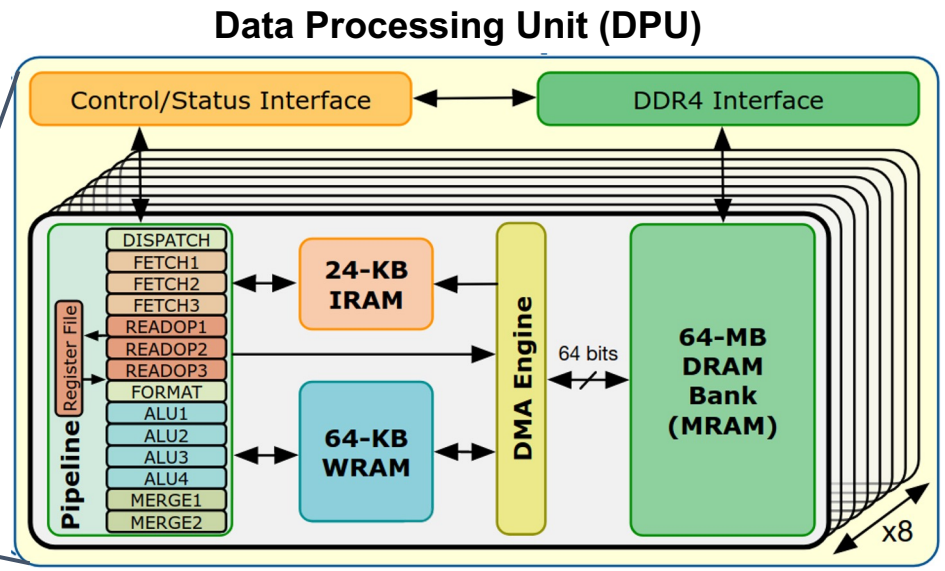
UPMEM PIM Hardware

- Commercially available PIM system
- Each DPU has **24 hardware threads**
- 2560 DPU ➔ > **60K thread**
- **No direct c ween DPUs**

Problem: synchronize access to shared memory

plications ed to be parallel!





Transactional Memory

- **Simpler alternative** to lock-based synchronization
- Programmers only need to identify **atomic blocks**
- Pros:
  - **Simplicity**: reduction software development cost
  - **Efficiency**: speculative concurrency control schemes

```
atomic
{
    A.withdraw(10)
    B.deposit(10)
}
```

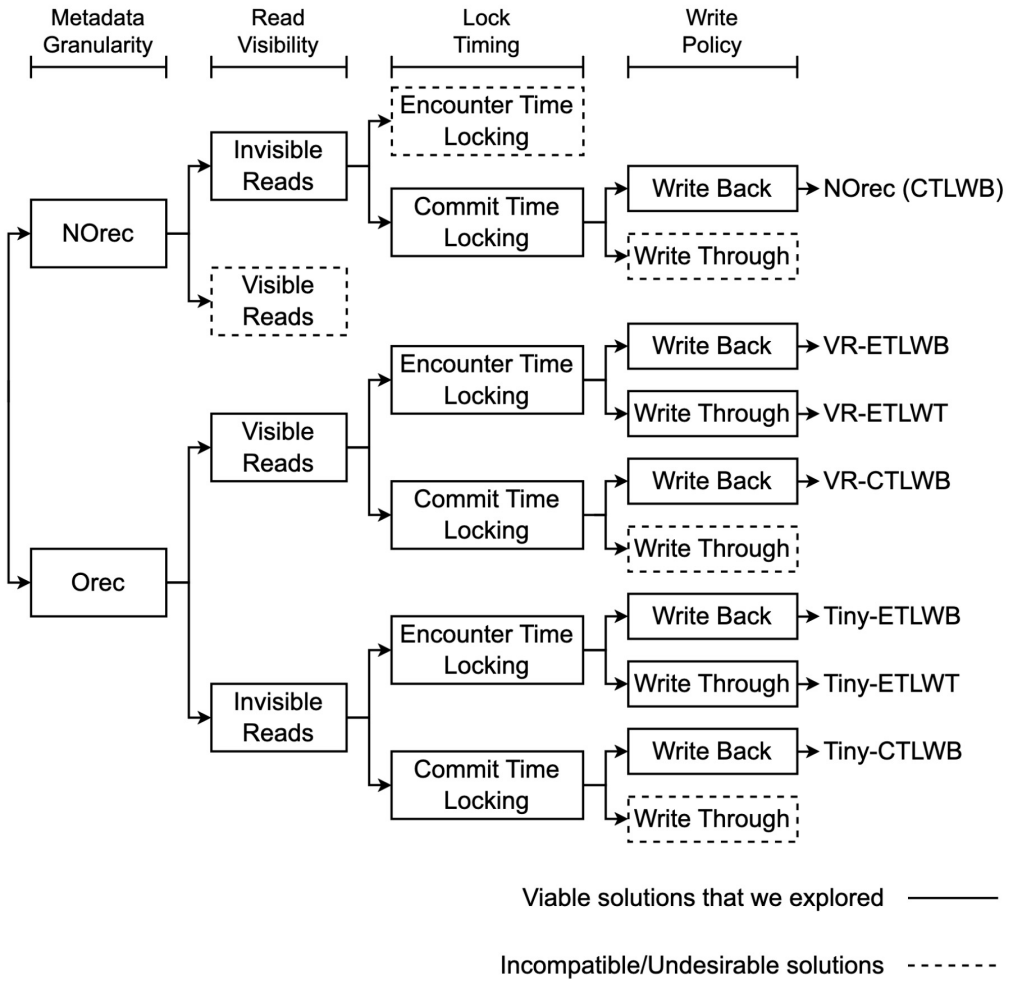
PIM-STM [ASPLOS'24]

- First **Software TM (STM) for PIM** devices
- Simplify the development of parallel applications for emerging PIM devices
- Set of **STM implementations** for the **UPMEM PIM system:**
  - Exhaustive study of the **design space of STM** algorithms
- PIM-STM library exposes a conventional STM API…

  ...but with some restrictions dictated by performance considerations

Investigated STM Designs

- **Metadata Granularity**
  - Granularity at which conflicts are detected
- **Read Visibility**
  - Visibility of reads by concurrent transactions
- **Lock Timing**
  - When locks are acquired
- **Write policy**
  - When writes are applied



Key Challenges

- Optimize STM implementations for UPMEM hardware features

1. No **direct communication** between DPUs

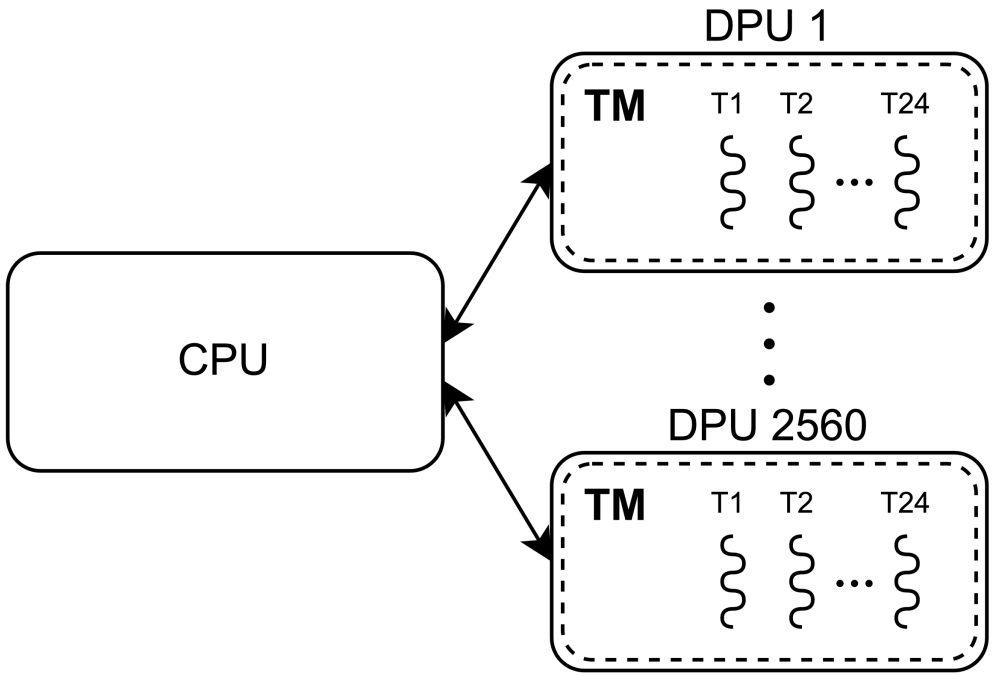
2. **Communication and computations** can't be overlapped

3. Usage of **WRAM vs MRAM** for STM metadata

4. **Atomic operations** with restricted semantics

PIM-STM: Key design choice

- No support for **distributed transactions**
- Key design choice to **promote high locality ➔ high performance**
- Transaction abstraction as an **intra-DPU synchronization primitive**



PIM-STM: Key Challenges & Contributions

- Optimize STM implementations for UPMEM hardware features

1. No **direct communication** between DPUs

2. **Communication and computations** can't be overlapped

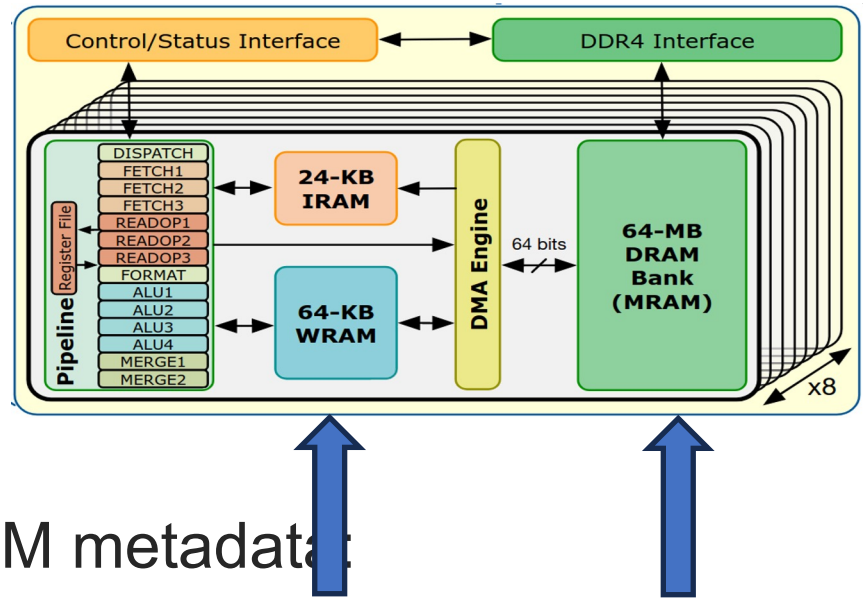
3. Usage of **WRAM vs MRAM** for STM metadata

4. **Atomic operations** with restricted semantics

Hardware-specific features: WRAM vs MRAM for STM metadata

- **WRAM**: 64**KB** fast scratchpad memory
- **MRAM**: 64**MB** slower DRAM

Where to store STM metadata ?
(e.g., read-/write-sets?)



- Trade-offs associated with using WRAM for STM metadata:

- Scarce resource that applications may use for other purposes

➔ Not necessarily available for the STM library

+ STM instrumentation overhead can be significantly mitigated:

➔ Up to 5.1x speed-ups in TM-intensive applications!

PIM-STM: Key Challenges & Contributions

- Optimize STM implementations for UPMEM hardware features

1. No **direct communication** between DPUs

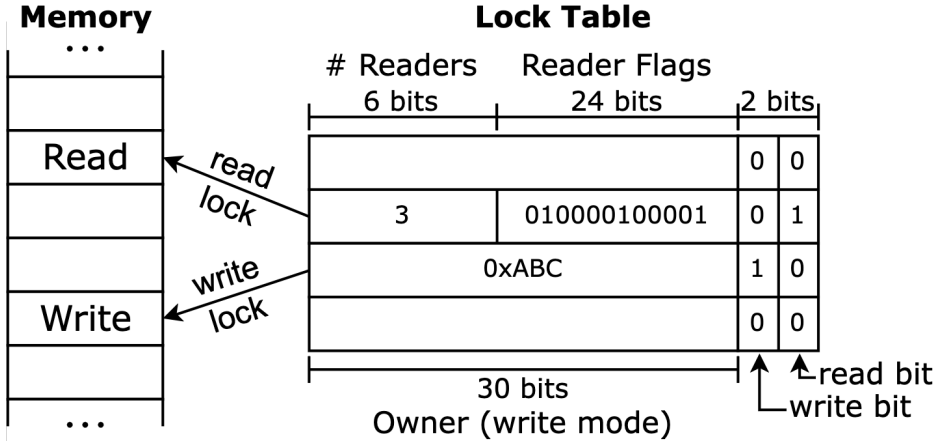
2. **Communication and computations** can't be overlapped

3. Usage of **WRAM vs MRAM** for STM metadata

4. Atomic operations with restricted semantics

Hardware-specific features: Atomic instructions with restricted semantics

- Orec-based STM designs rely on lock-tables
- In CPU-based STMs, lock table entries are manipulated atomically via CAS
- But CAS is not available in UPMEM!
- Only available atomic instructions:
  - `acquire/release <addr>`
  - `<addr>` mapped to a bit of a 256 bits-wide hw register ➔ **aliasing**



Hardware-specific features: Atomic instructions with restricted semantics

- Why not using `aquire/release <addr>` to implement STM locks directly?

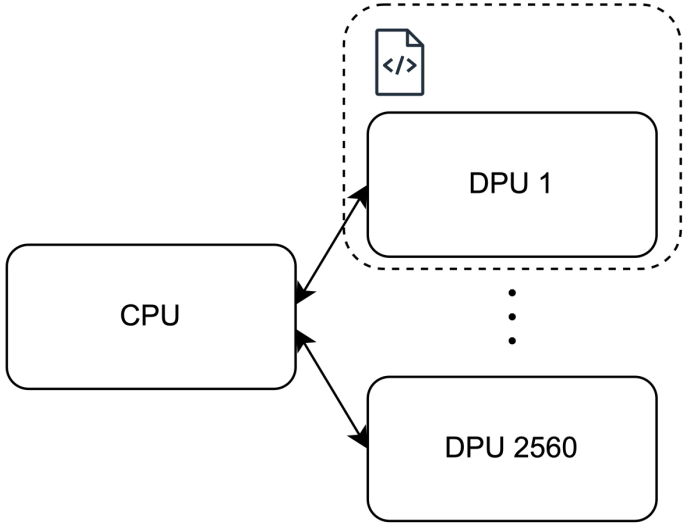
- **Problem**:
  - **Aliasing can cause spurious contention on STM locks**
  - **Lock contention ➔ transaction abort to avoid deadlocks**

- **Solution: lock-table's entries guarded via a "latch"**
  - Latches held only while manipulating/querying a lock table entry
  - Negative impact of aliasing is strongly reduced:
    - Upon contention: **wait** instead of **abort** (no deadlocks!)
    - Latch duration << STM locks duration (whole transaction) ➔ **wait time is small**

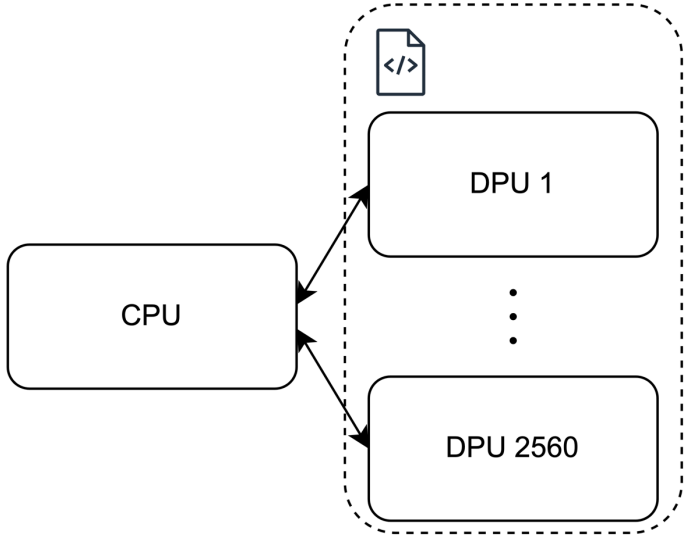
Evaluation

Key research questions:

1. Which STM designs work best for workloads that use a single DPU



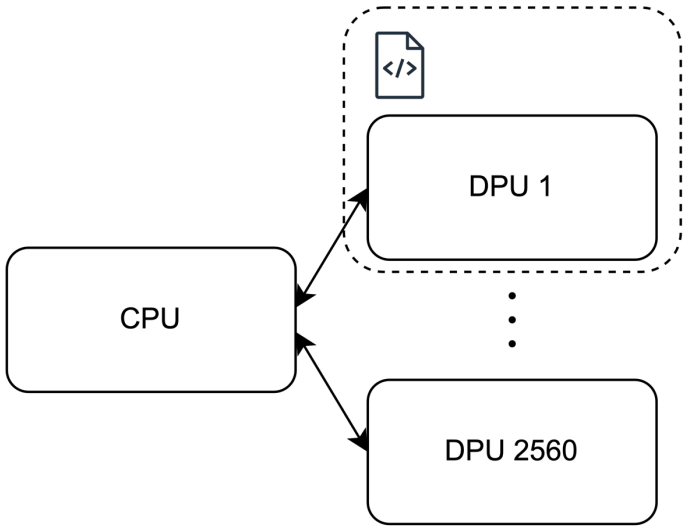
2. Efficiency with multi DPUs vs CPU based implementation



Evaluation: Single-DPU Study

Key research questions:

1. Which STM designs work best for workloads that use a single DPU



Benchmarks

- Concurrent data structures
  - ArrayBench
  - Linked-List
- Porting of STAMP
  - KMeans (AI/ML)
  - Labyrinth (Circuit Routing)

Evaluation: Single-DPU Study

- No one-size-fits-all solution
- Metadata Granularity
  - NOrec tends to perform better due to handling less metadata
- Read Visibility
  - Visible reads tens to perform worse due to spurious aborst
- Lock Timing
  - Commit time locking tends to perform worse due to wasted work
- Write Policy
  - Limited impact on performance

Evaluation: Single-DPU Study

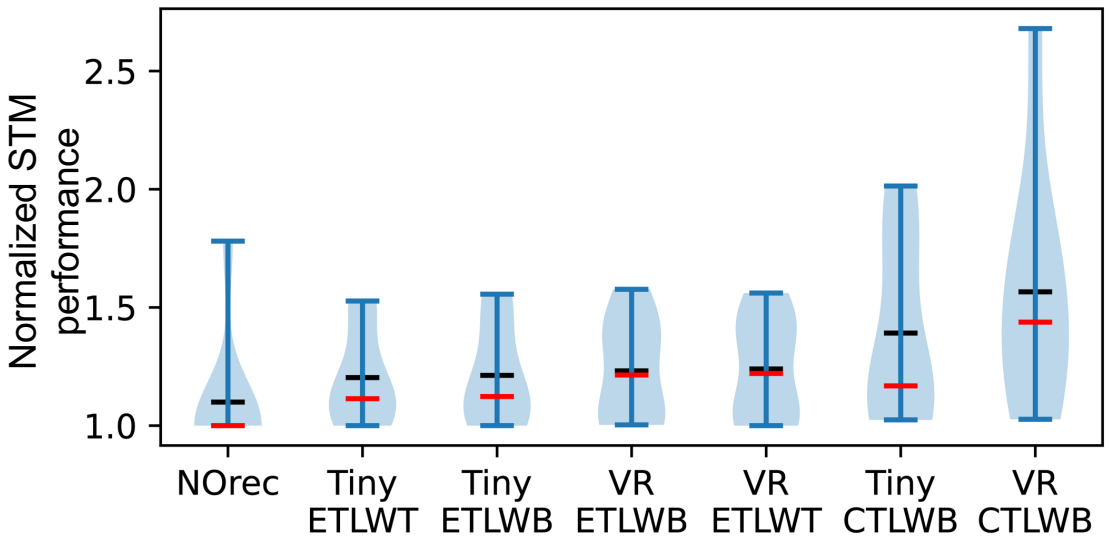
- No one-size-fits-all solution

- Metadata Granularity
  - NOrec tends to perform better due to handling less metadata
- Read Visibility
  - Visible reads tens to perform worse due to spurious aborst
- Lock Timing
  - Commit time locking tends to perform worse due to wasted work
- Write Policy
  - Limited impact on performance

Details in our ASPLOS'24 paper

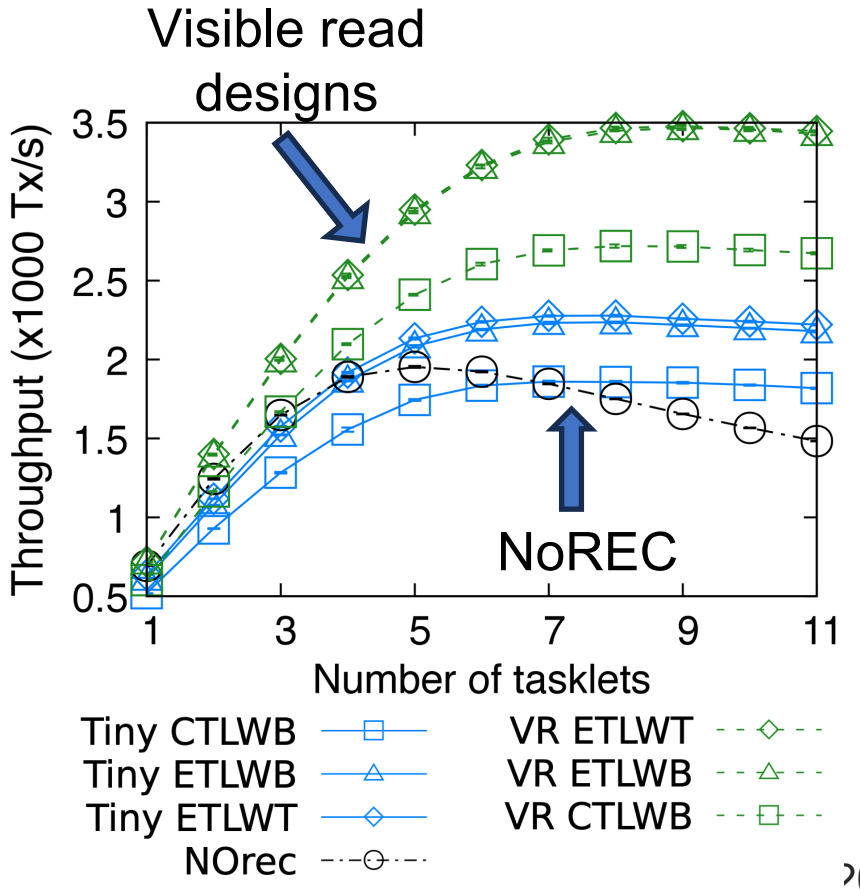
Evaluation: Single-DPU Study

- No one-size-fits-all solution



NoREC is on average the best performing appoach

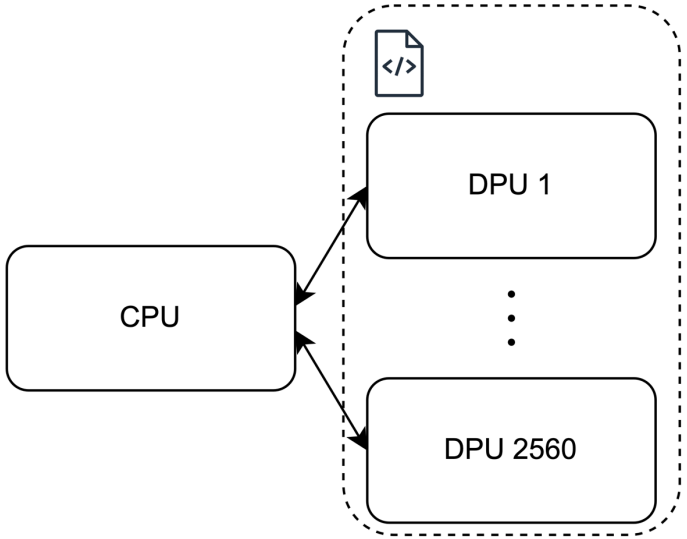
Yet, NOREC can be the worst performing solution in some scenarios!



Evaluation

Key research questions:

2. Efficiency with multi DPUs vs CPU based implementation



Benchmarks

- Labyrinth
  - Independent instances on DPUs
- KMeans
  - Adapted to distribute data across DPUs
  - Results produced by each DPU are merged by the CPU

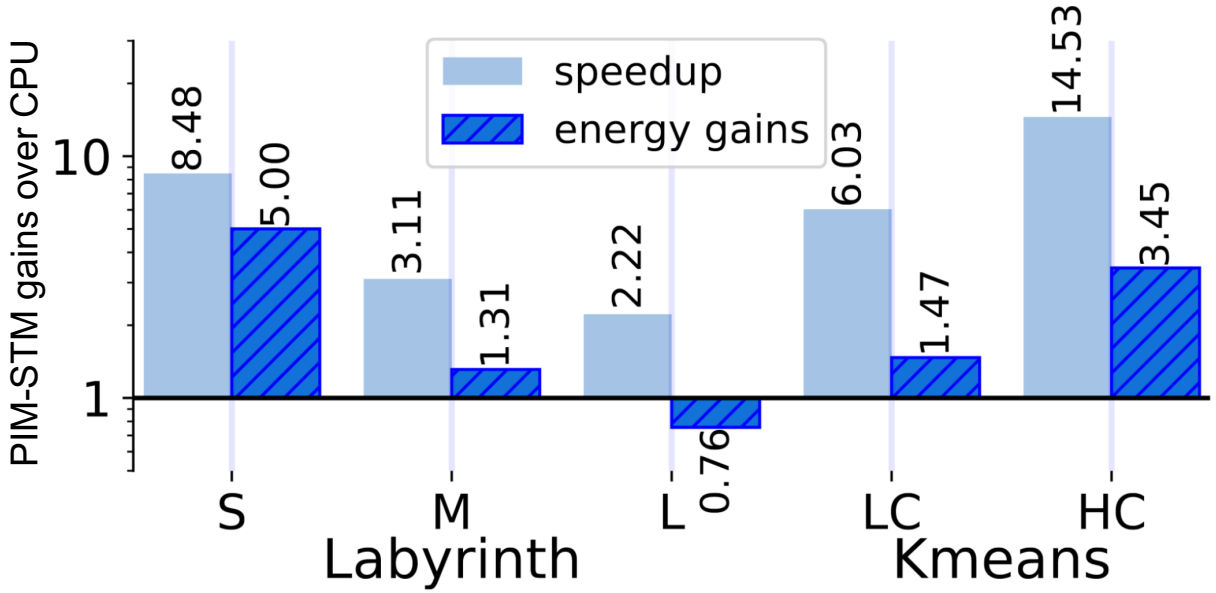
Evaluation: Multi-DPU Study

- PIM achieves performance gains wrt CPU in all scenarios:
  ➔**Up to 14.53x**

- PIM achieves **energy** gains in all but 1 scenarios
  ➔ **Up to 5x**

+ excellent performance potential

- room for improving energy efficiency



Conclusions

- First implementation of **STM for PIM** devices
- Restricted semantics: no distributed transactions
  - Key trade-off: sacrifice generality in favour of efficiency
- Exhaustive study of the **design space of STM** algorithms
- Comparison with CPU-based implementation
  - Remarkable speed-ups (up to 14x)
  - Less satisfactory regarding energy efficiency (up to 31.5% higher energy consumption)

Future work

- Algorithmic extensions
  - Investigate efficient ways to support distributed transactions:
    - Revisit trade-off between generality and efficiency

- Target new application domains
  - Deterministic parallelization of Blockchains
  - In-memory storage for AI inference pipelines

Thank you!