Energy-Efficient Parallel Data Stream Compression for IoT Applications

Shuhao Zhang
Assistant Professor, SUTD

The talk will be based on the following works

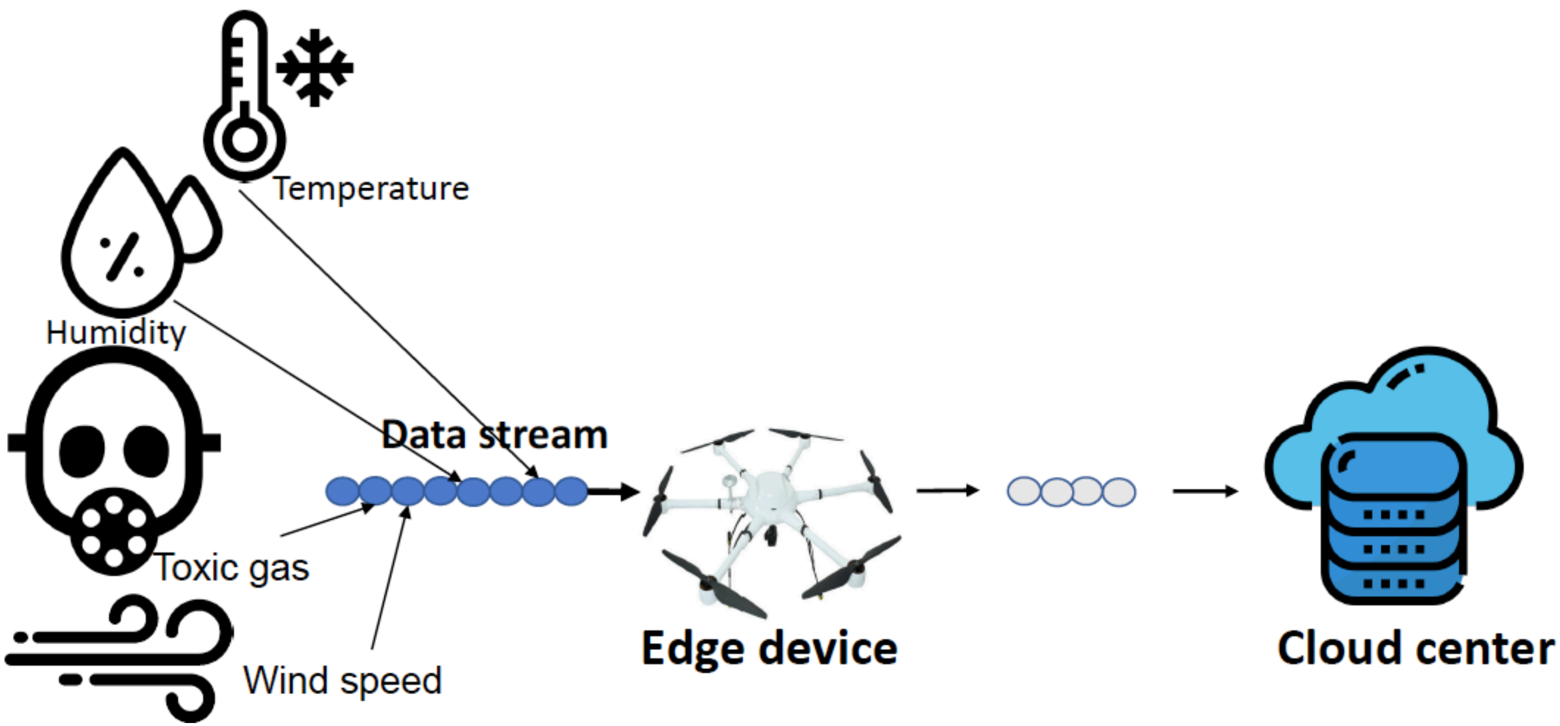
- *[ICDE 2023] Xianzhi Zeng*, and Shuhao Zhang.* Parallelizing Stream Compression for IoT Applications on Asymmetric Multicores
- *[SIGMOD 2023] Yancan Mao#, Jianjun Zhao, Shuhao Zhang, Haikun Liu, and Volker Markl*. MorphStream: Adaptive Scheduling for Scalable Transactional Stream Processing on Multicores
- *[ICDE 2023] Yu Zhang, Feng Zhang, Hourun Li, Shuhao Zhang, and Xiaoyong Du*. CompressStreamDB: Fine-Grained Adaptive Stream Processing without Decompression

**: my student*
#: my staff

Outline

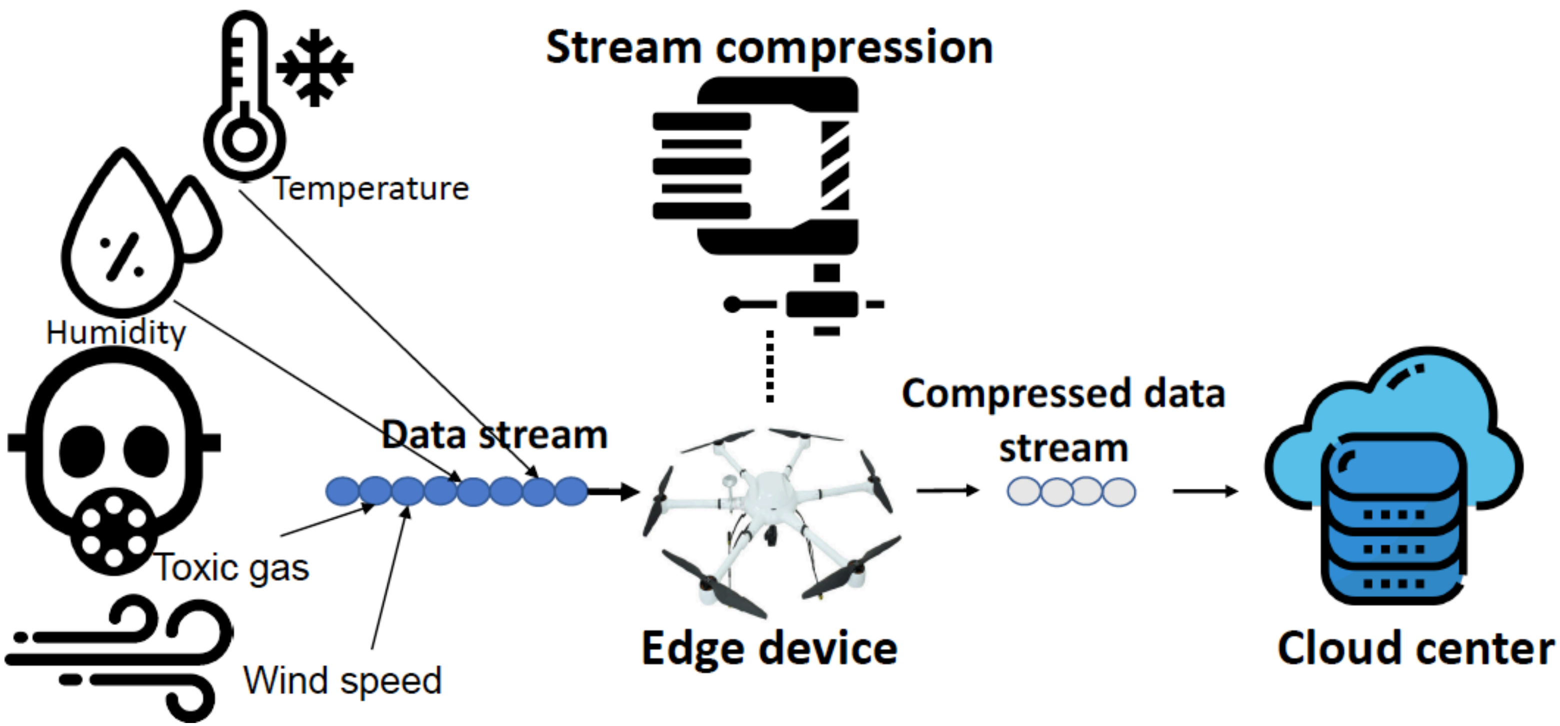
- Background

Background



Real-time data gathering at the patrol drone

Background



Real-time data gathering and Stream compression at the patrol drone

Design Requirements of Stream Compression for IoT

- Adopting compression DOES NOT guarantee "plug-and-play" performance benefits.
- Two requirements:
  - (R1) Low Latency Stream Compression
  - (R2) Low Energy Consumption

Opportunity: Asymmetric Multicore Processor

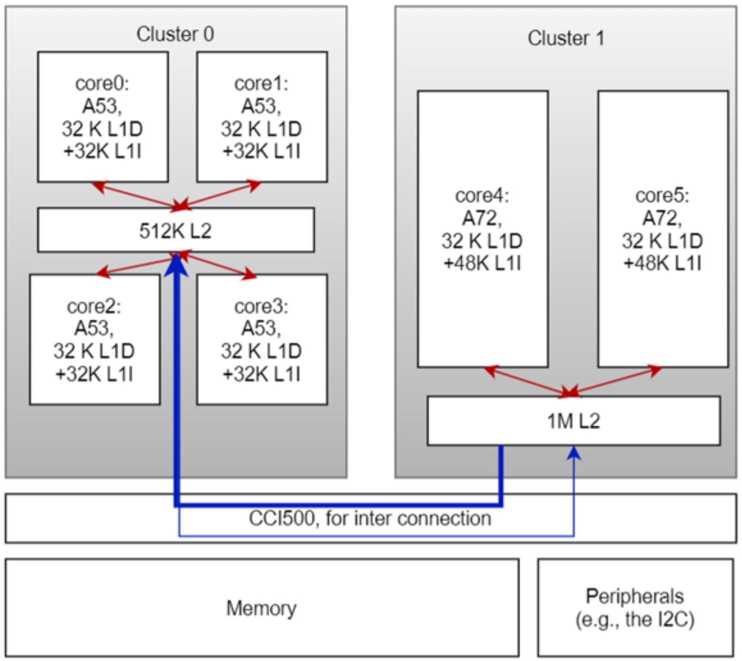


Figure 1: The 6-core AMP rk3399.

- The rk3399 processor, a 6-core AMP under the same ISA
- Be of both high-performance and energy-efficiency

Modern ARM machines with asymmetric multicores are typical choices for IoT devices

Design Challenges

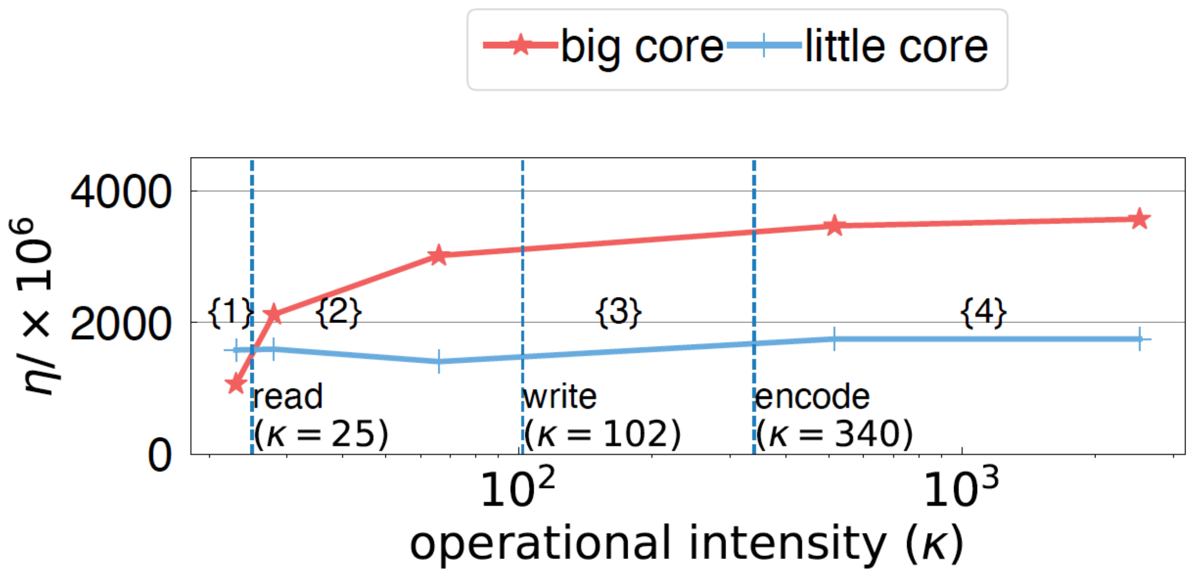
- Parallelizing stream compression on asymmetric multicores seems a natural choice to satisfy the aforementioned two requirements (<u>Low Latency, Low Energy)</u>
- However, the involved asymmetry effects require a careful system design.
  - See our observations …

Outline

- Background
- Observation and Motivation

Observation 1

- There are varying task-core affinities in different parts of stream compression procedure.



Observation 2

- There are large differences of communication costs among asymmetric cores.
  - C0: cross-core communication
  - C1: communication from big core to little core.
  - C2: communication from little core to big core.

Path	Bandwidth	Latency
intra-cluster $c0$	2.7 GB/s	70.4 ns
inter-cluster $c1$	0.7 GB/s	142.4 ns
inter-cluster $c2$	0.4 GB/s	420.8 ns

Bandwidth and latency of cross-core communication in rk3399

Limitations of Existing Work

I. Existing mechanisms consider the coarse-grained scheduling and <u>do not expose the fine-grained task-core affinities</u> in the workload.

II. They surprisingly <u>overlook the different costs of C1 and C2</u>, which is important to consider when scheduling decomposed tasks that involves heavy inter-task communications.

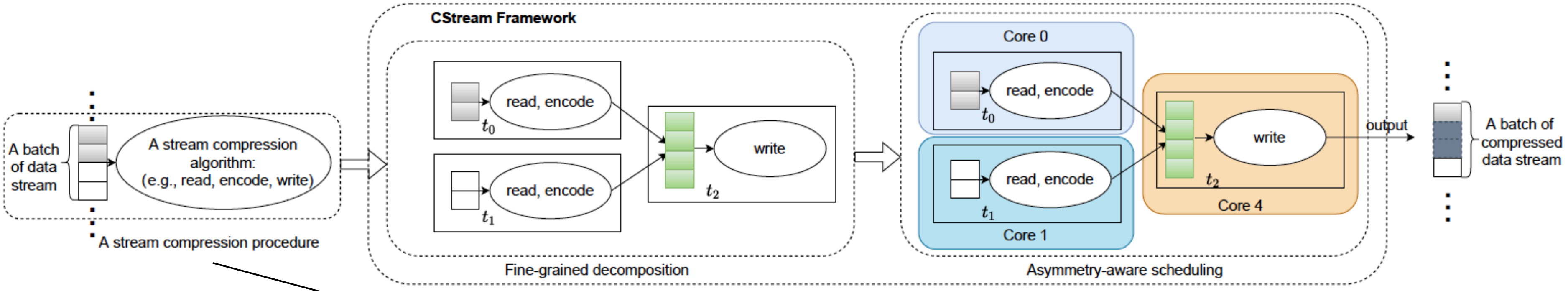
Outline

- Background
- Observation and Motivation
- Solution Overview

Our Proposal: CStream

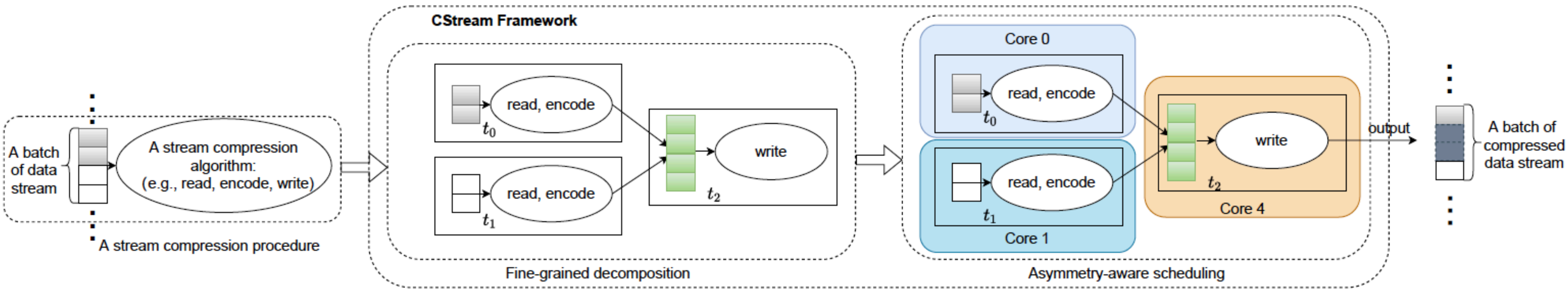
- We propose CStream, a novel framework of parallelizing stream compression for IoT applications.
- CStream parallelizes stream compression, such that it **minimizes energy consumption** while satisfying a **user-specified compressing latency** constraint.

Overview of CStream



A stream compression procedure is the process of executing a stream compression algorithm on a batch of data streams. For simplification, we use {Algorithm – Dataset} to denote a stream compression procedure, e.g., {LZ4 – Stock}.

Overview of CStream



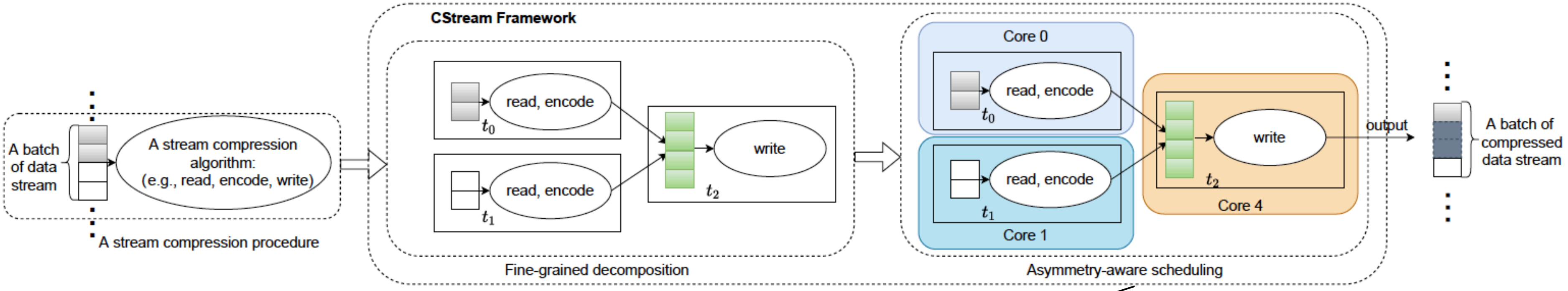
Algorithm 2: Stateful stream compression

Input: input stream $inData$
Output: output stream $outData$
while $inData$ *is not stopped* **do**
  (s0) read the tuples from $inData$;
  (s1) pre-process ;
  (s2) state update ;
  (s3) state-based encoding;
  (s4) write compressed data to $outData$;
end

Exploring Pipelining Parallelism: Each step (s0, s1, …) can run in a pipeline fashion. Task fusion can be applied when communication overhead is large.

Exploring Data Parallelism: We can replicate each step to further explore data parallelism.

Overview of CStream



The decomposed tasks are scheduled to asymmetric multicores to minimize total energy consumption (E) without violation of user-specified compressing latency constraint (L_{set}), guided by a **novel cost model**.

Outline

- Background
- Observation and Motivation
- Solution Overview
- Problem Formulation and Cost Models

Problem Formulation

$$minimize(E_{est} = \sum_i e_i) \quad (1)$$

s.t., $\forall t_i \forall j$,

$$L_{set} \geq L_{est} = \max(l_i) = \max(l_i^{comp} + l_i^{comm}), \quad (2)$$

$$C_j \geq \sum_{t_i \text{ at core } j} \eta_i \quad (3)$$

- e_i stands for energy consumption of task t_i.
  - (1): our goal is to minimize total energy consumption subject to two constraints.
  - (2): enforces that latency within constraint, which is determined by the pipeline bottleneck.
  - (3): enforces that resource demands within constraint.

The Cost Model Overview

The model estimates both energy consumption (e_i) and compressing latency (l_i) of each task:

- 1) the e_i is estimated by the operational intensity (κ_i) of each task
- 2) the l_i is the summation of <u>computation latency</u> and <u>communication latency</u> of the each task t_i.

Estimation of e_i

- We estimate e_i as a proportional relationship to the instructions per unit time (η_i) and the latency (l_i), and an inverse proportional relationship to the instructions per unit energy (ζ_i)

We use operational intensity (κ_i) to estimate η_i

$$e_i = \frac{\eta_i \times l_i}{\zeta_i}$$

The estimation of instructions of unit energy consumption (ζ_i) involves different parameter values including the boundary of regions, the growth rate (i.e., a), and the intercept (i.e., b).

Estimation of l_i

- The compression latency (l_i) of a task t_i is the sum of two non-overlapping components l_i^{comm} and l_i^{comp}.
  - l_i^{comm} (communication latency) varies depending on where the task and its upstream tasks are scheduled.

Please checkout our paper for the detailed models

Outline

- Background
- Observation and Motivation
- Solution Overview
- Problem Formulation and Cost Models
- Implementation and Evaluation

Implementation Details

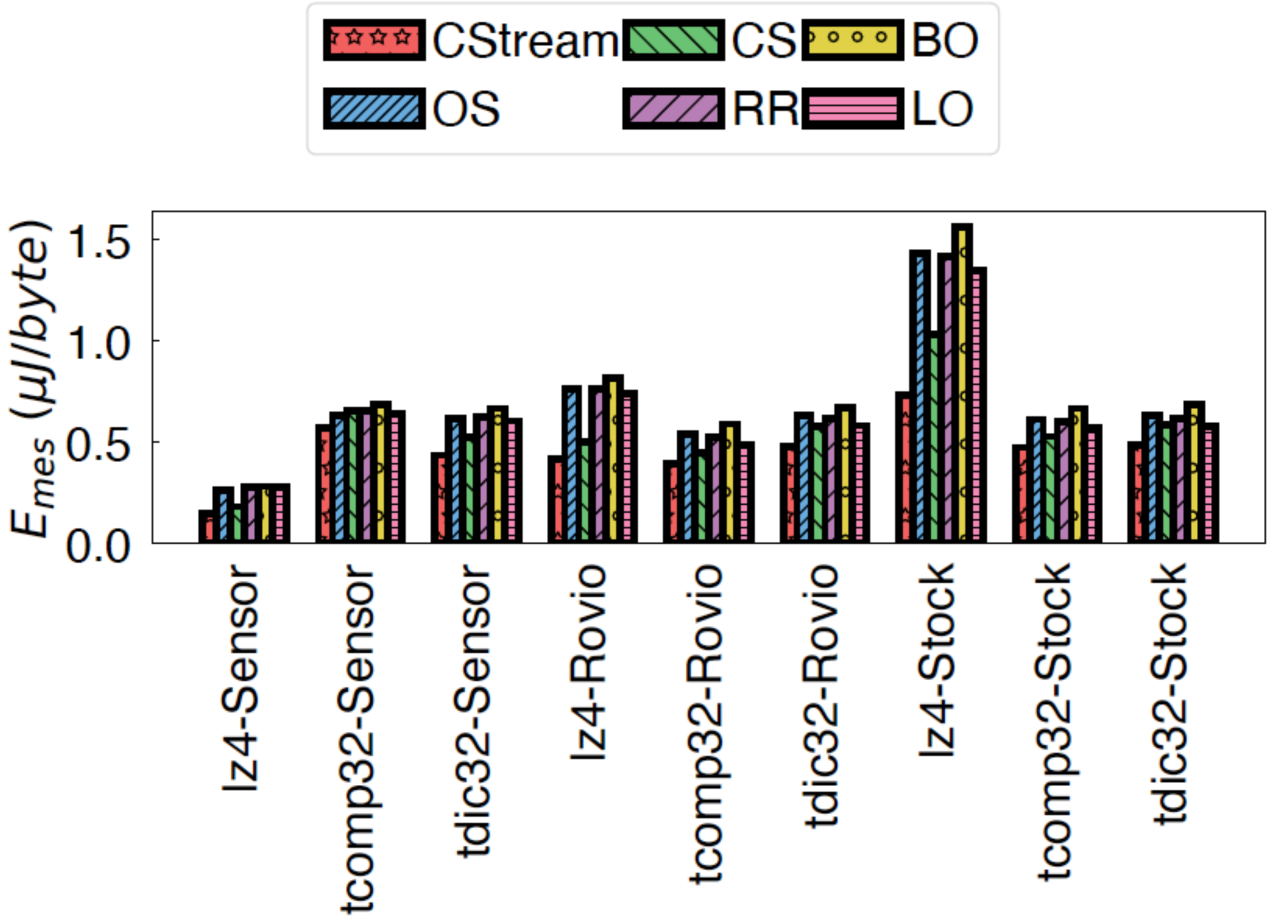
- Based on the propose cost model, Cstream searches for optimal scheduling plan by enumerating all possible plans with *dynamic programming*.
- To adapt to dynamic environment, Cstream further equips with a *PID control-based* regulation mechanisms to calibrate the cost model and conduct re-scheduling.
- We have further developed an energy meter that provides accurate measurement with low overhead[1].

1 Technical Report (https://tonyskyzeng.github.io/downloads/tr_cstream/TR_CSTREAM.pdf)

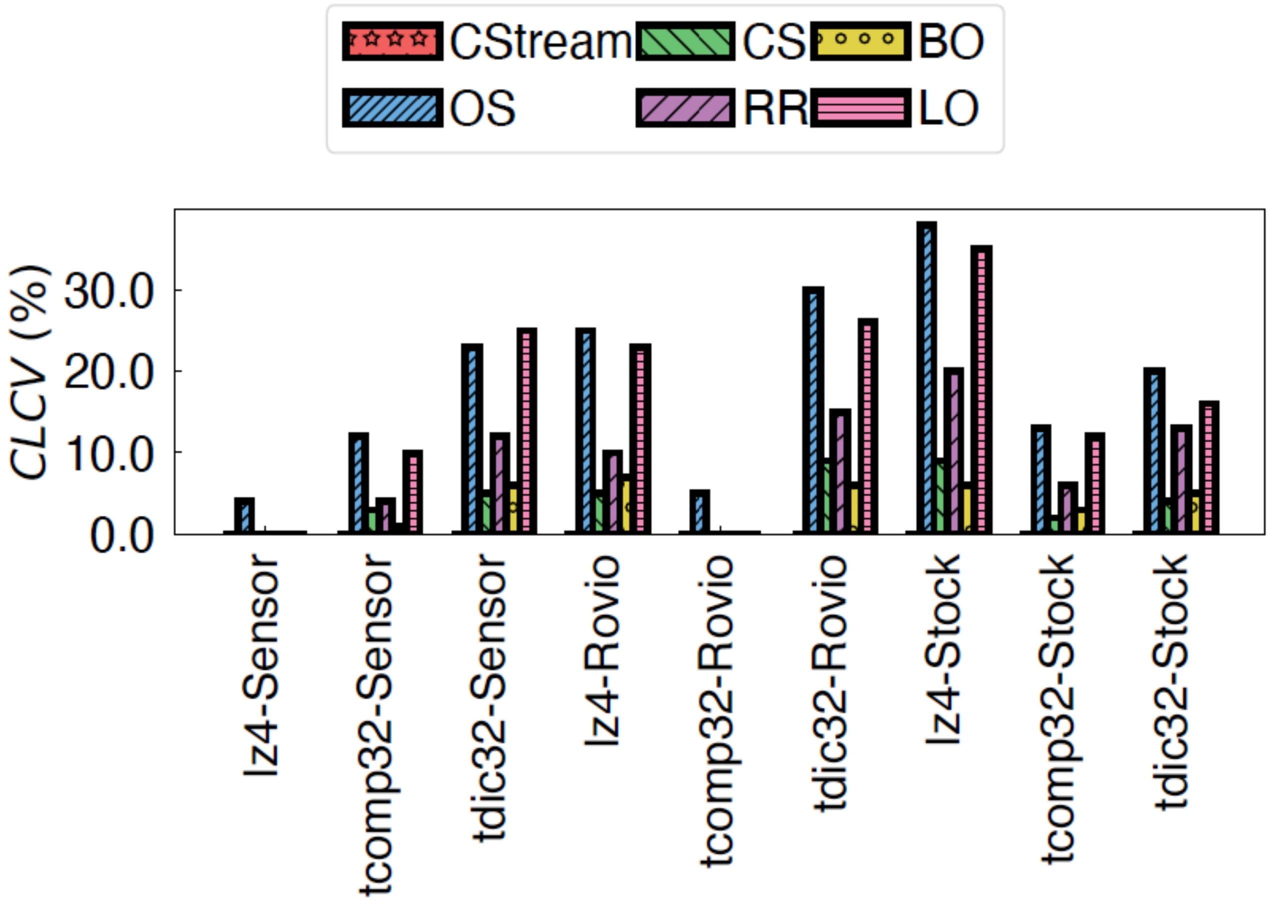
Evaluation

- Workloads:
  - Algorithms: tcomp32, lz4, and tdic32.
  - Datasets: Sensor, Rovio, and Stock.
- Hardware:
  - Radxa Rockpi 4a (rk3399 asymmetric multicores processor)
- Evaluation Metrics:
  - Compressing latency constraint violation ($CLCV$ for short)
  - energy consumption, denoted as E_{mes}
- Competing mechanisms:
  - OS, CS (Mobicom’21), RR, BO, and LO

End-to-End Comparison on Energy Consumption



End-to-End Comparison on Compressing Latency Violation



More Experimental Results Show that …

- CStream is able to self-adjust to dynamic environment.
- It is able to perform well under varying:
  - Compressing latency constraint
  - Batch size
  - Vocabulary duplication
  - Symbol duplication
  - Dynamic range

Outline

- Background
- Observation and Motivation
- Solution Overview
- Problem Formulation and Cost Models
- Implementation and Evaluation
- Conclusion and Outlook

Conclusion

- CStream achieves the following desired properties:
  - 1) when the compressing latency constraint (Lset) set by the user is relatively loose, it can achieve the least energy consumption
  - 2) when encountering a tight Lset, its latency constraint violation is always minimized.

Outlook

- This work opens up multiple interesting directions for further exploration. E.g.,
  - Exploring the more complex trade-off among information loss, compressibility, energy consumption, and compressing latency with **transactional state management**[2].
  - Exploring compression-aware stream operation to run directly on IoT devices **without decompression**[3].

[2]MorphStream: Adaptive Scheduling for Scalable Transactional Stream Processing on Multicores, *Yancan Mao, Jianjun Zhao, Haikun Liu, and Shuhao Zhang, Volker Markl*, SIGMOD2023

[3]CompressStreamDB: Fine-Grained Adaptive Stream Processing without Decompression, *Yu Zhang, Feng Zhang, Hourun Li, and Shuhao Zhang, Xiaoyong Du*, ICDE2023

Want to know more of our other works?

- Welcome to visit our website:
  - https://shuhaozhangtony.github.io/team/

Thanks