61 THE CHIPS TO SYSTEMS CONFERENCE

SHAPING THE NEXT GENERATION OF ELECTRONICS

JUNE 23-27, 2024

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

THE CHIPS TO SYSTEMS CONFERENCE
SHAPING THE NEXT GENERATION OF ELECTRONICS
JUNE 23-27, 2024
MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

PHD: Parallel Huffman Decoder on FPGA for Extreme Performance and Energy Efficiency

Yunkun Liao[1,2,3], Jingya Wu[1], Wenyan Lu[1,4], Xiaowei Li[1,3], Guihai Yan[1,4]

SKLP, Institute of Computing Technology, Chinese Academy of Sciences[1]

University of Chinese Academy of Sciences[2]

Zhongguancun Laboratory[3]

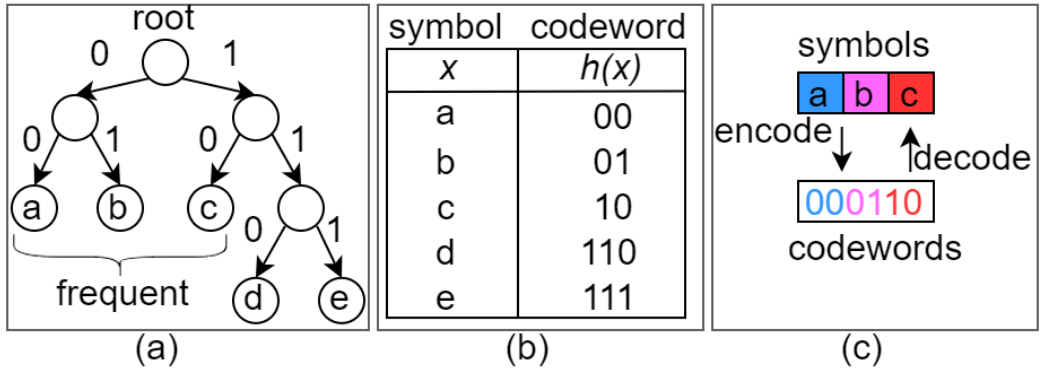
YUSUR Technology Co., Ltd.[4]

CONTENTS

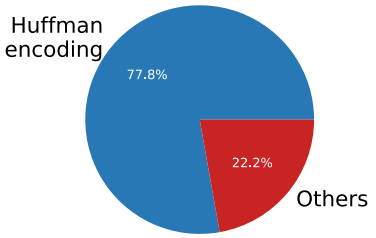
Background Knowledge

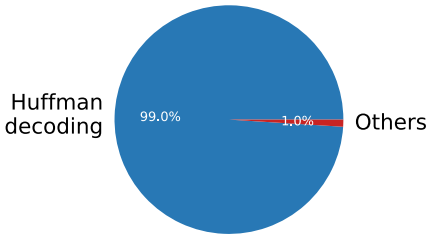
Huffman Coding

- A lossless data compression scheme.
  - Main idea: Encodes frequent symbols using shorter codewords and vice visa.
- Widely used in many data compression algorithms.
  - LZSS, ZIP, JPG, etc.
- Domain-specific accelerator design for Huffman coding is required.
  - This work accelerates Huffman decoding.



(a) Huffman coding tree; (b) Huffman coding table; (c) Huffman encoding and decoding.





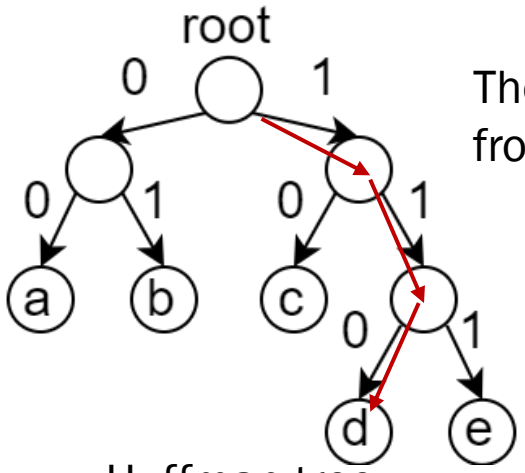
Profiling conducted on the LZSS compression[1]

An accelerator for Huffman decoding is required.

[1]https://github.com/hzxa21/15618-FinalProject

Huffman Decoding

- Bit-serial decoding based on traversing the Huffman tree.
- Bit-parallel decoding based on looking up codebook.
  - Dominated method in existing Huffman decoding accelerators.



Huffman tree

The codebook can be constructed from the Huffman tree.

addr	symbol
000	a
001	a
010	b
011	b
100	c
101	c
110	d
111	e

Regular codebook

How to decode multiple codewords in parallel?

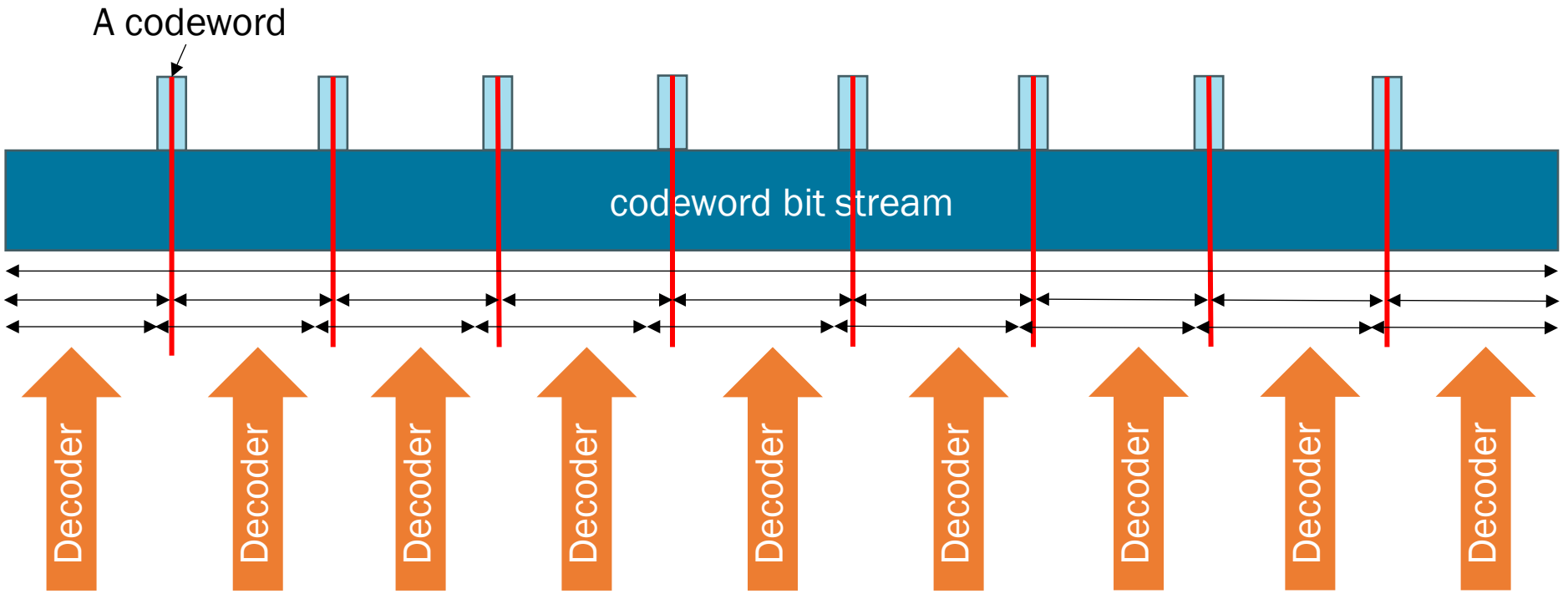
Decoding “110” to “d” requires three traversals.

Decoding “110” to “d” requires one lookup.

Huffman decoding requires codeword-level parallelism.

Parallelizing Decoding is Hard

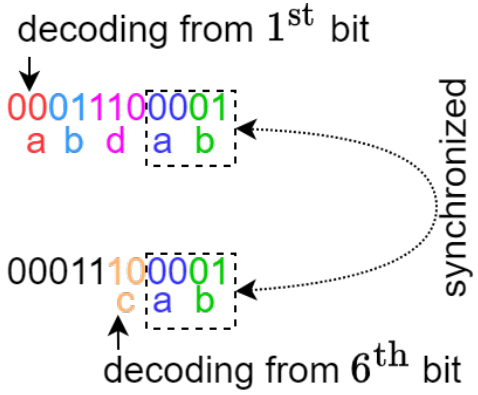
- Codeword-level parallelism can be obtained by partitioning the codeword bit stream.



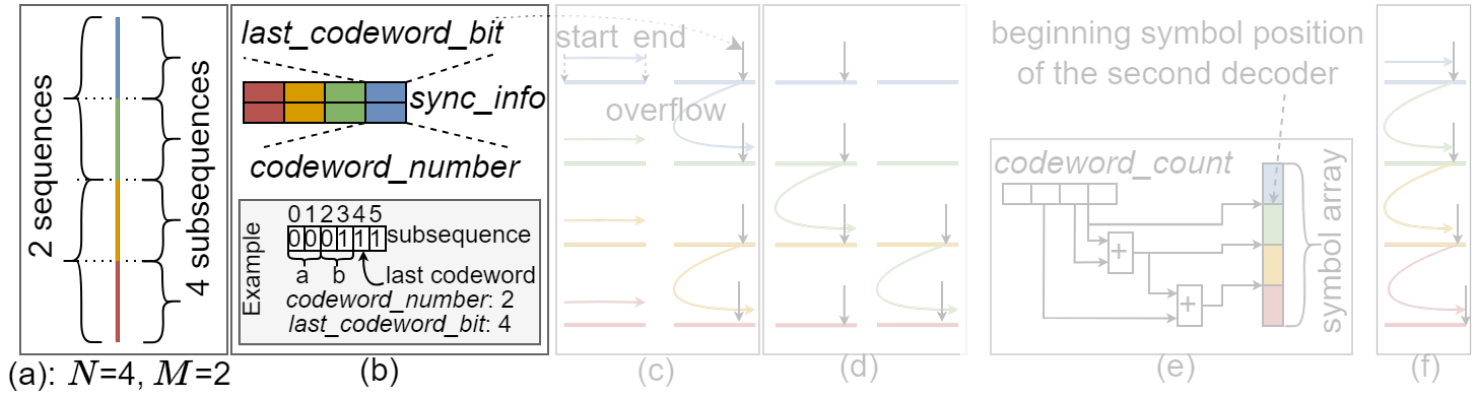
Parallelizing the decoding requires identifying codeword boundaries.

Parallelizing Decoding by Self-synchronization

- Almost all Huffman codes have the self-synchronization property[2].
- Achieve codeword-level parallelism with the self-synchronization.
  - Phase_1: Intra-sequence synchronization.
  - Phase_2: Inter-sequence synchronization.
  - Phase_3: Calculating the codeword boundaries.
  - Phase_4: Parallel decoding on N subsequences.



Self-synchronization property.

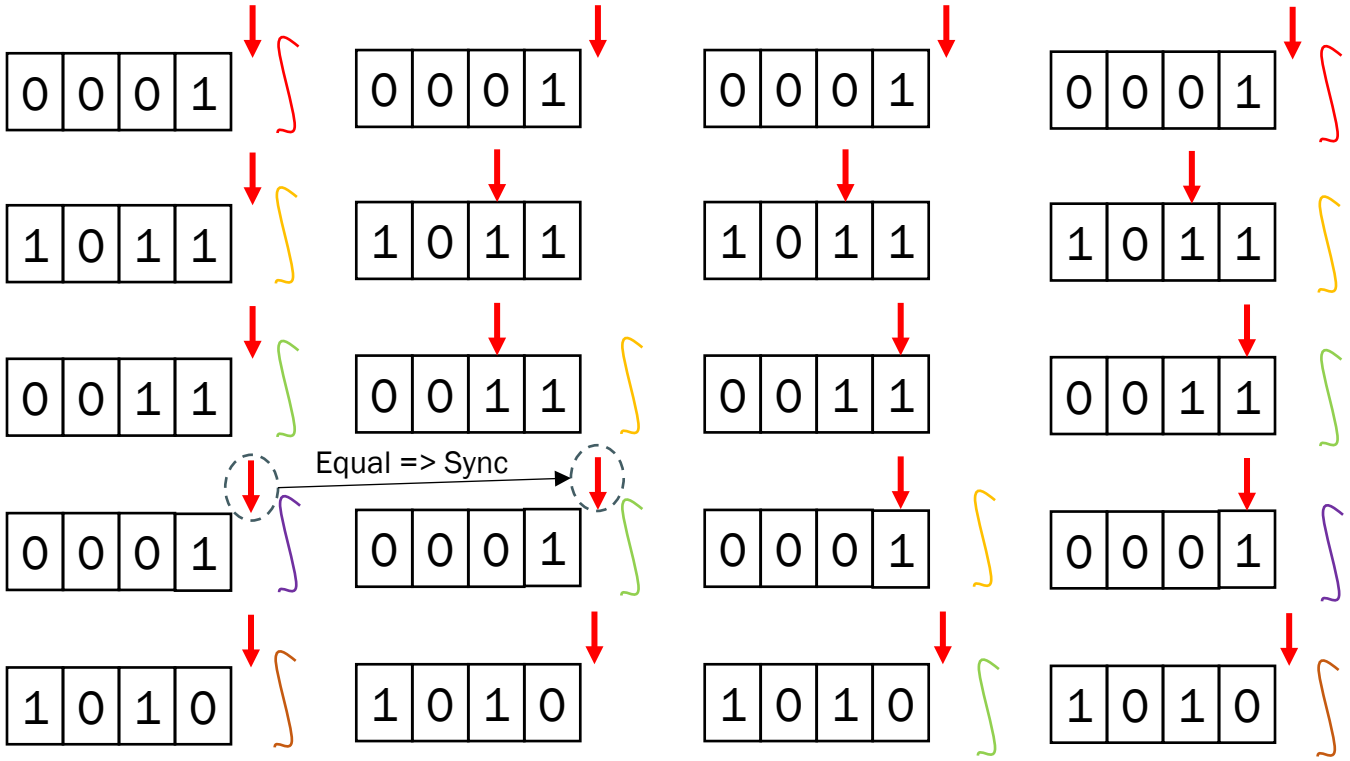


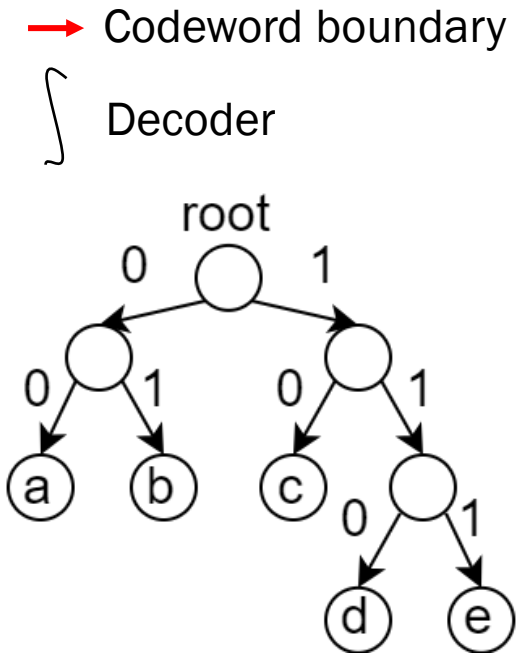
(a) Codewords splitting; (b) Synchronization information; (c) Phase_1; (d) Phase_2 (e) Phase_3 (f) Phase_4.

Self-synchronization trades computational complexity for parallelism.

[2]Ferguson, Thomas, et al. "Self-synchronizing huffman codes (corresp.)." IEEE Transactions on Information Theory 30.4 (1984): 687-693.

Parallelizing Decoding Example





Reference Huffman Tree

Decoder state table: Sync √, Not Sync ×。

∫	×	√	√
∫	×	×	√
∫	×	×	√
∫	×	√	√
∫	×	√	√

Assume decoding one symbol requires constant time T

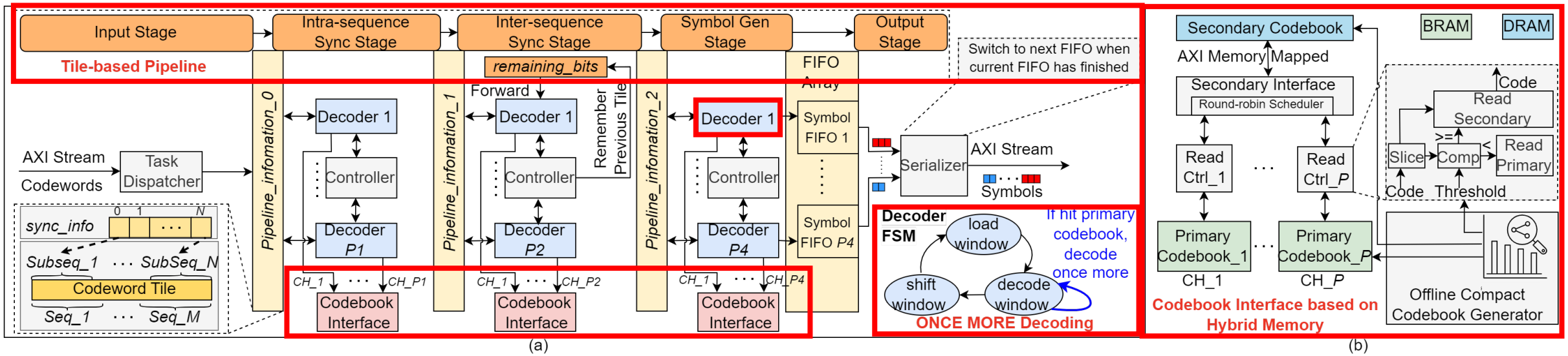
Serial decoding: $9T$

Parallel decoding : $8T$

Proposed Architecture: PHD

Overall Architecture of PHD

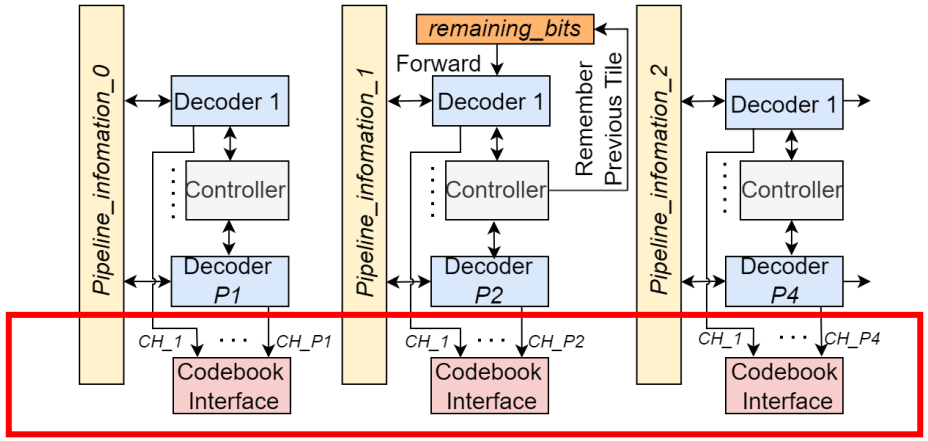
- PHD: The first self-synchronization-based Parallel Huffman Decoder architecture for FPGA.
  - Codebook interface based on hybrid memory.
  - ONCE MORE decoding optimization.
  - Tile-based pipeline dataflow.



PHD is the first accelerator to leverage the self-synchronization.

The Challenge of Codebook Storage

- The decoders of PHD is based on bit-parallel Huffman decoding, requiring codebook.
- Homogeneous storage designs.
  - BRAM-only: Fast, but with limited codebook size.
    - Regular codebook for S-bit Huffman code is 2^S bytes.
      - For HTTP/2 HPACK Huffman code[3], S is 30, requiring 2GB.
  - DRAM-only: Support large codebook, but slow.
- Low-latency and storage-efficient codebook interface is required.

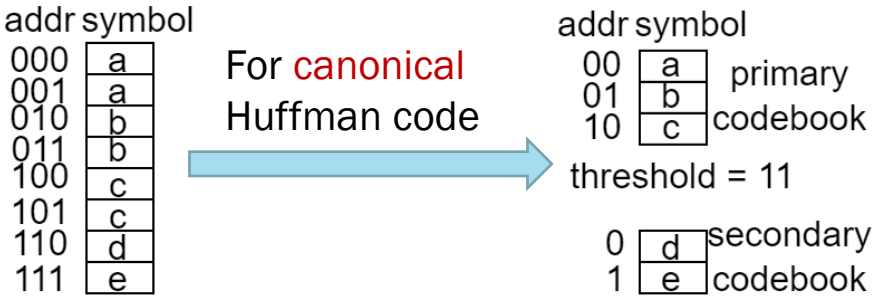


PHD should balance the lookup speed and BRAM consumption.

[3]Peon, Roberto, et al. HPACK: Header compression for HTTP/2. No. rfc7541. 2015.

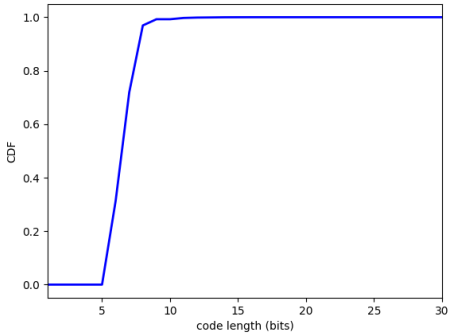
Codebook Interface based on Hybrid Memory

- Key enabler: Most Huffman codes are canonical. The regular codebook for canonical code can be split into one primary codebook and multiple secondary codebooks[4].
- Our insight: Primary codebook is for short codewords. The symbols of short codeword are more common in Huffman code by design.
- Our hybrid storage design:
  - Multi-channel BRAM for primary codebook
  - Shared-channel DRAM for secondary codebooks.

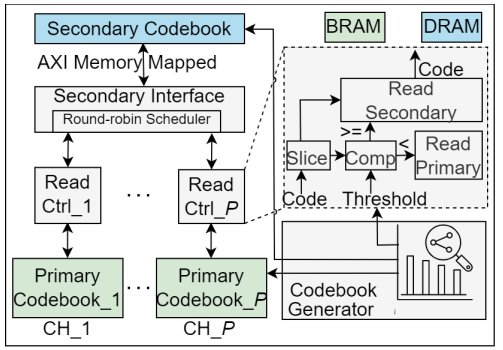


Regular codebook

Compact codebooks



CDF of code length from HPACK-Huffman files



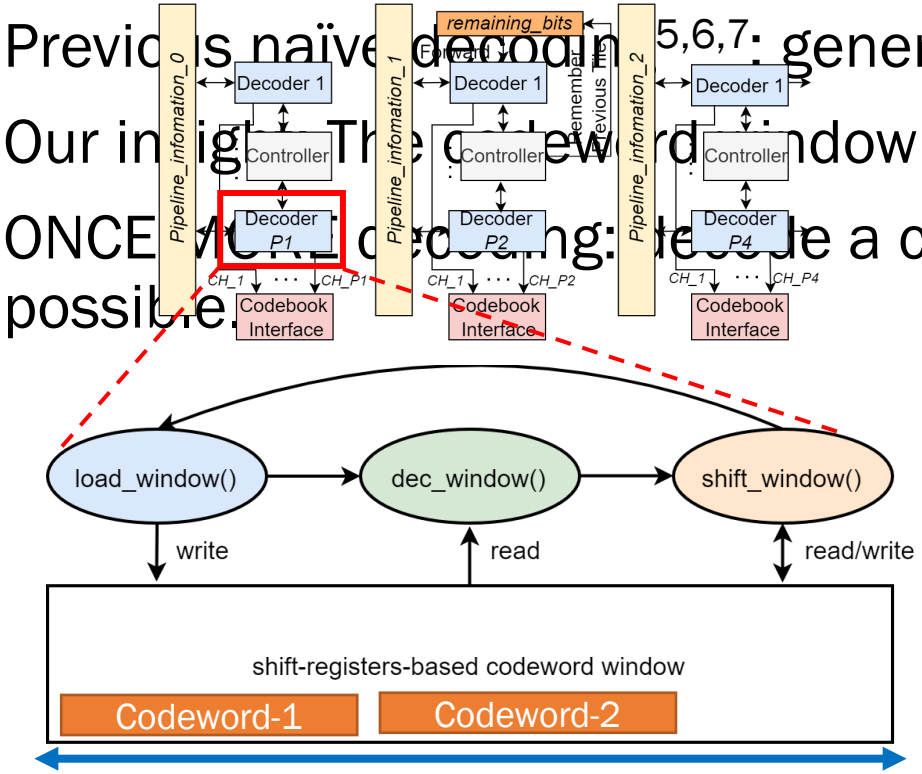
Codebook interface based on hybrid memory

PHD makes short (common) -codeword lookup fast.

[4]Yamamoto, Naoya, et al. "Huffman coding with gap arrays for GPU acceleration." In ACM ICPP, 2020.

ONCE MORE Decoding

- Codewords are sent to a shift-register-based window for decoding.
- Previous naïve decoding[5,6,7]: generate one symbol per iteration.
- Our insight: The codeword window may contains multiple short codewords.
- ONCE MORE decoding: decode a codeword window in one iteration as much as possible.



The shift register can contain the longest codeword.

PHD makes subsequence-level decoders fast.

[5]Weißenberger, André, et al. "Massively parallel Huffman decoding on GPUs." In ACM ICPP, 2018.
[6]Rivera, Cody, et al. "Optimizing huffman decoding for error-bounded lossy compression on gpus." In IEEE IPDPS, 2022.
[7]AMD Xilinx. "Vitis Accelerated Libraries." https://www.xilinx.com/products/design-tools/vitis/vitis-libraries.html.

The Challenge of Codeword-Tile Dependency

- On-chip codeword tile is used to support one-cycle codewords access.
- The size of on-chip codeword tile is limited and codewords are split into multiple tiles.
- There are dependencies between consecutive tiles.
  - A codeword may span two tiles.
- Ignoring the dependencies between consecutive tiles leads to incorrect decoding results of the remaining codeword tiles.

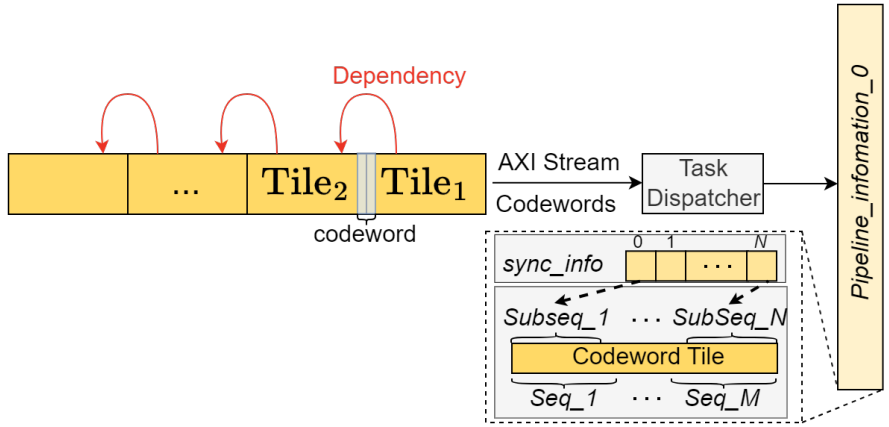
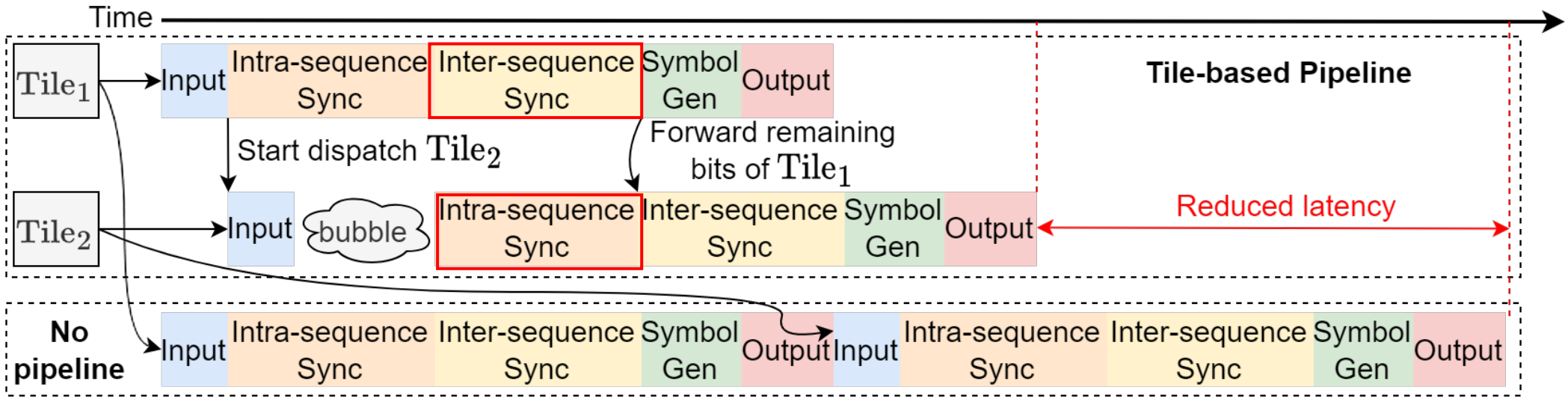


Illustration of the codeword-tile dependency

PHD should guarantee the correctness.

Tile-based Pipeline

- Our insight: no dependencies between codeword tiles at the subsequence level in the intra-sequence synchronization stage.
  - Intra-sequence Sync of Tile_2 can run in parallel with the inter-sequence Sync of Tile_1.
- Our methods: remember and forward the remaining bits of the previous tile to the subsequent tile in the inter-sequence synchronization stage.



PHD enables tile-level parallelism without compromising correctness.

Experiment Results

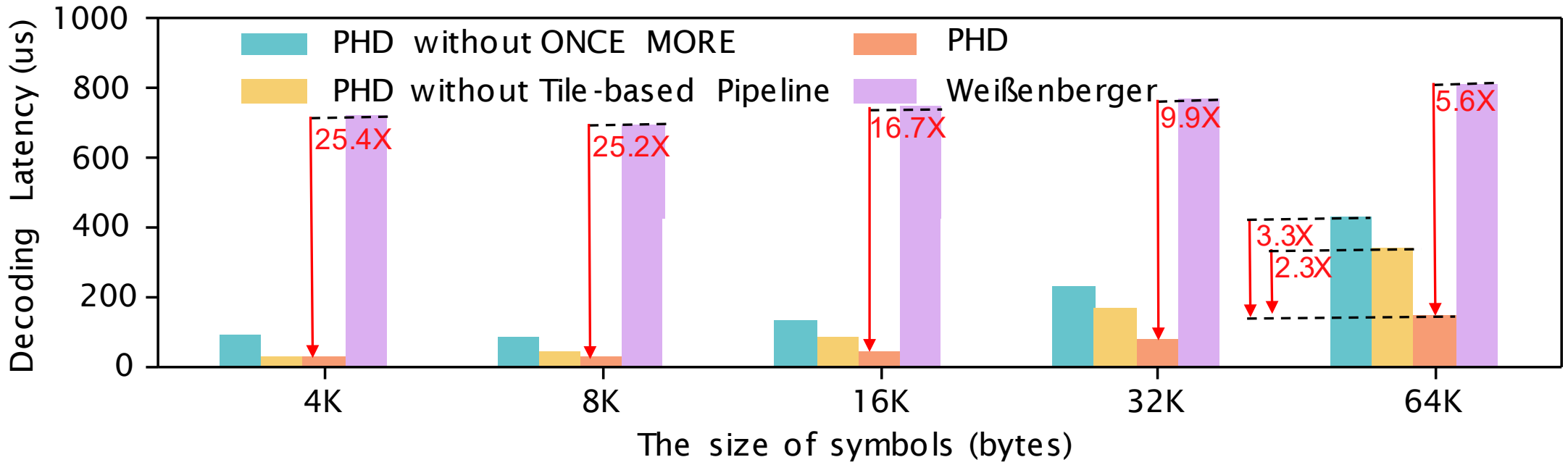
Setup

- Huffman code: HTTP/2 HPACK, used for compressing the HTTP/2 packet header fields.
- PHD: Described using the Xilinx HLS C++, Xilinx Vitis 2022.1 is used for cycle-accurate simulations, synthesis and implementations.
- Baselines:
  - CPU: a performance-optimized library called LS-HPACK[8], running on Intel Core-i5 12400 processor.
  - GPU: the open-source code of Weißenberger[9], adapted to HTTP/2 HPACK, running on NVIDIA GeForce RTX 4090 GPU.
  - FPGA: Bit-parallel, described using the Xilinx HLS C++.

[8]https://github.com/litespeedtech/ls-hpack
[9]https://github.com/weissenberger/gpuhd

Decoding Latency

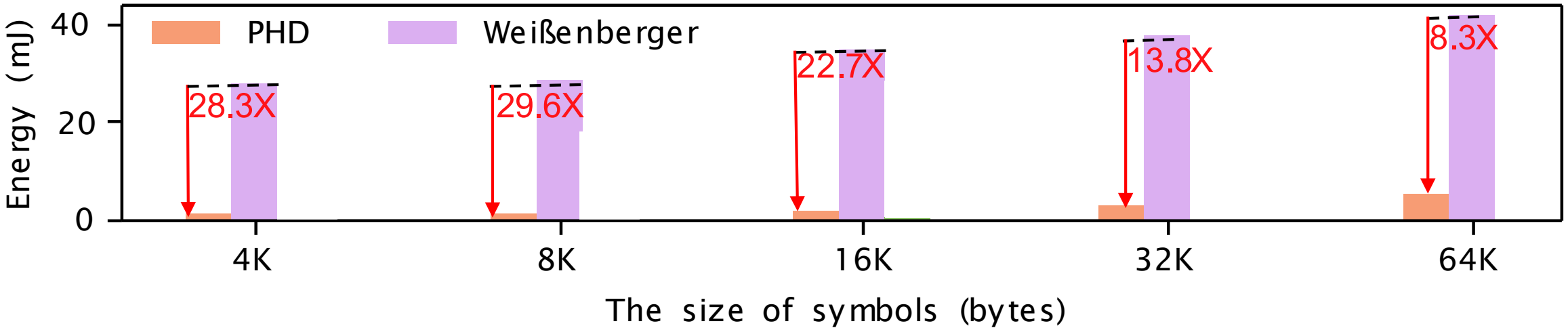
- Results:
  - Compared with CPU baseline: a latency reductions ranging from 24.5X to 60.6X.
  - Compared with bit-parallel FPGA baseline: a latency reduction ranging from 18.5X to 57.5X.
  - Compared with GPU baseline: latency reductions of 5.6X to 25.4X over Weißenberger.



PHD is faster !

Energy Consumption

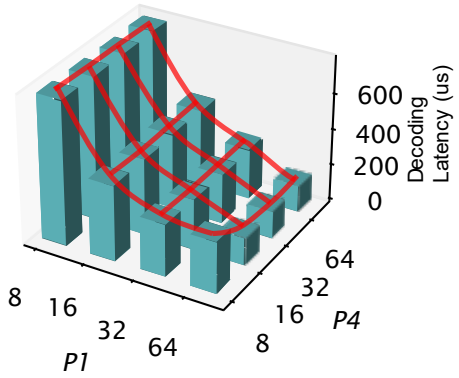
- Results:
  - PHD reduces energy by 8.3X to 29.6X over Weißenberger.

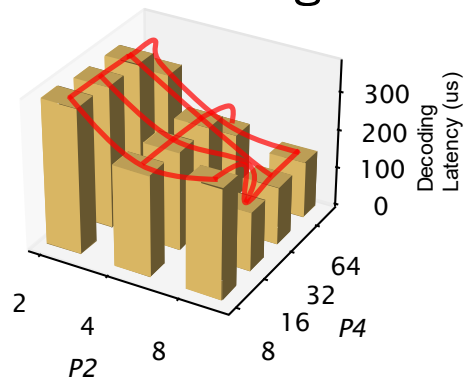


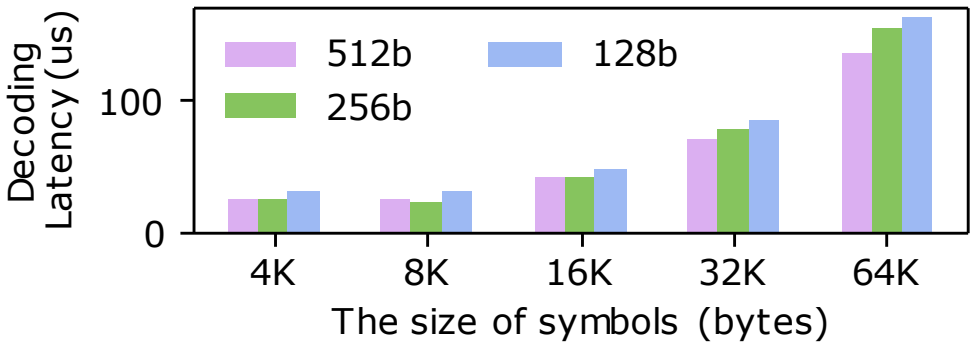
PHD is more energy-efficient !

Design Space Exploration

- Architecture parameters in PHD:
  - $P1$: Subsequence-level parallelism in intra-sequence synchronization stage.
  - $P2$: Subsequence-level parallelism in the inter-sequence synchronization stage.
  - $P4$: Subsequence-level parallelism in the symbol generation stage.
  - Subsequence size: workload capacity of each decoder.
- Results:
  - Large $P1$ and $P2$ parallelism are crucial for overall performance.
  - Smaller subsequence size leads to a little longer latency







PHD is parameterized !

Summary

Summary

- We presented PHD, the first accelerator targeting at self-synchronization-based parallel Huffman decoding.
  - PHD realizes bit-level, codeword-level and tile-level parallelism.
  - PHD implements a compact codebook interface based on hybrid memory.
  - PHD proposes ONCE MORE optimization to accelerate subsequence decoding.
- Future work:
  - An analytical performance model of PHD.
  - Generating and tuning PHD automatically.

If you need the code for research, please send an email to me! ☺

61 THE CHIPS TO SYSTEMS CONFERENCE

SHAPING THE NEXT GENERATION OF ELECTRONICS

JUNE 23-27, 2024

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

THANK YOU!