

Refine and Recycle: A Method to Increase Decompression Parallelism

Authors: Jian Fang¹, Jianyu Chen¹, Jinho Lee²,
Zaid Al-Ars¹, H. Peter Hofstee^{1,2}



Outline

- Motivation
- Snappy (De)compression
- Design Challenges
- Refine and Recycle Method
- Proposed Decompressor Architecture
- Experiments
- Summary

Snappy (De)compression Background

- LZ77-based, byte-level
- In Hadoop ecosystem
- Support Parquet, ORC, etc.
- Low compression ratio
- Fast compression and decompression

Motivation

- Snappy is widely used but not fast enough
 - Widely used in databses and big data processing
 - Optimized CPU throughput about 1.4GB/s in a CPU core (only 1% out of the main memory bandwidth)
- New and high-bandwidth connections for storage
 - PCIe Gen4 or OpenCAPI attached NVMe
 - 24 NVMe drives have around 75GB/s read bandwidth
 - Difficult for prior design to keep up with this high bandwidth
 - FPGAs can be used to decompress them

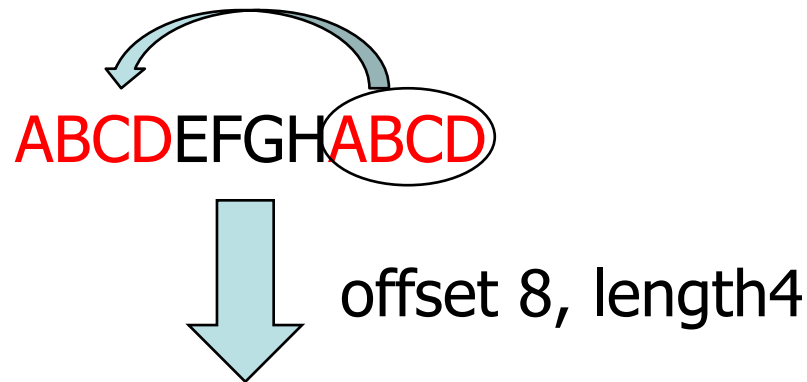
Example Motivating System: OpenPOWER MiHawk



- 24 NVMe
 - Each >3GB/s read
- 4x OpenCAPI
 - Each ~22GB/s
- 4x Gen4 x16
 - e.g. 200Gb/s E-net
- Would like each FPGA-based adapter to support 6-8 NVMe (20-25GB/s)

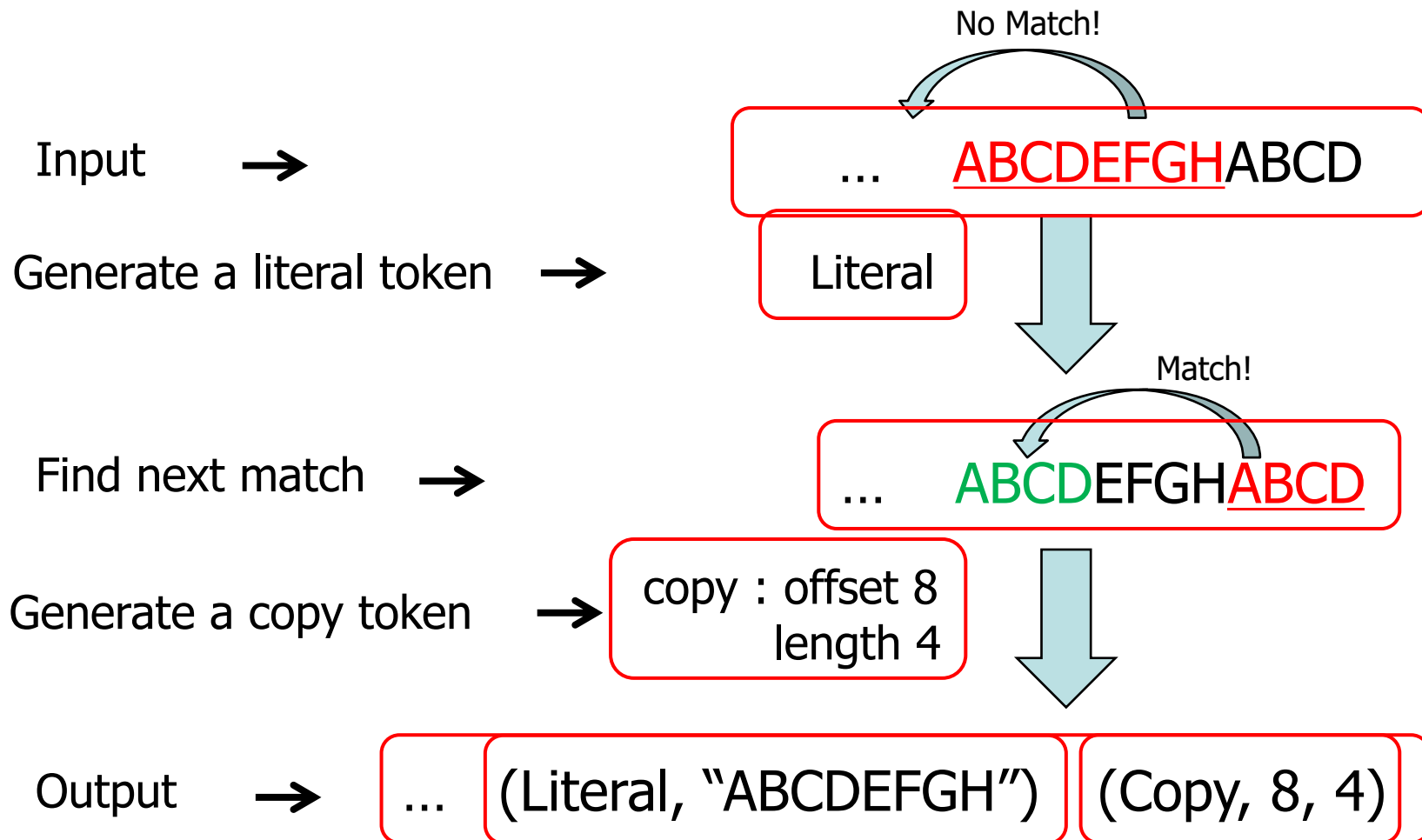
Snappy Compression

- Two kinds of token: Literal token and copy token
- Find match from previous data
- Example:

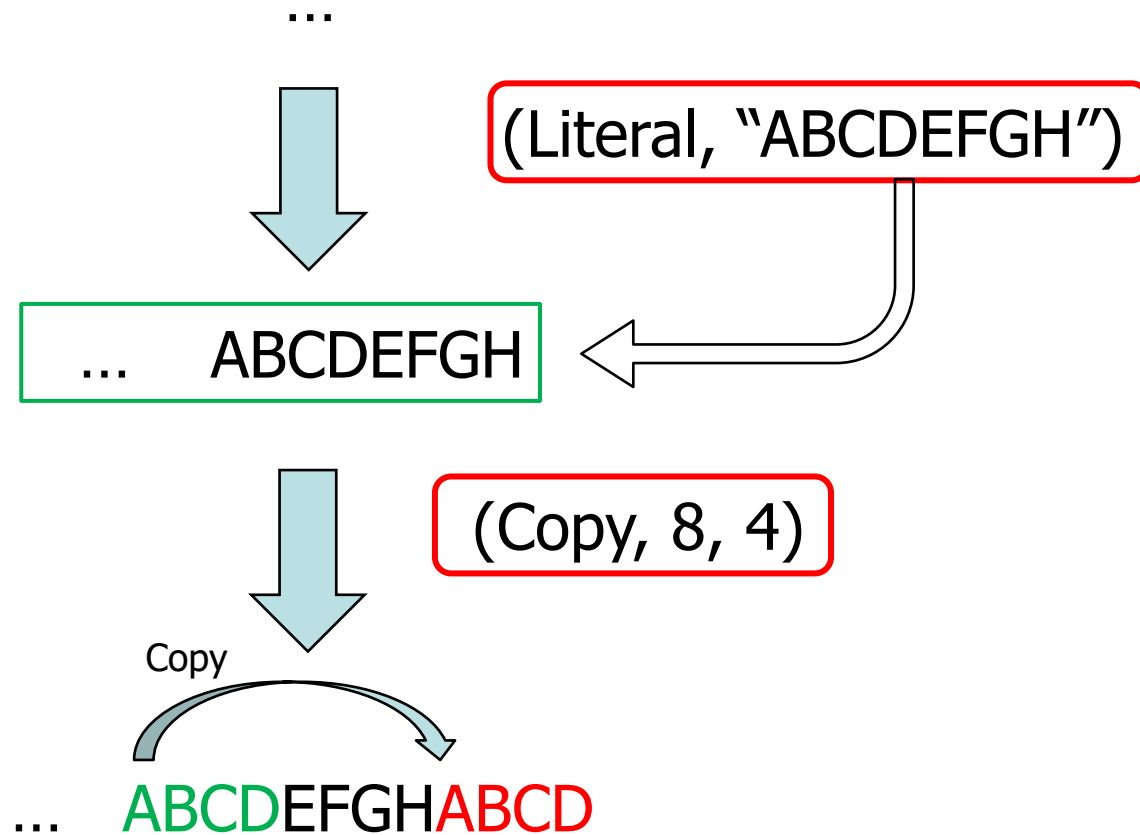


Use (8,4) to replace "ABCD"

Snappy Compression



Snappy (De)compression

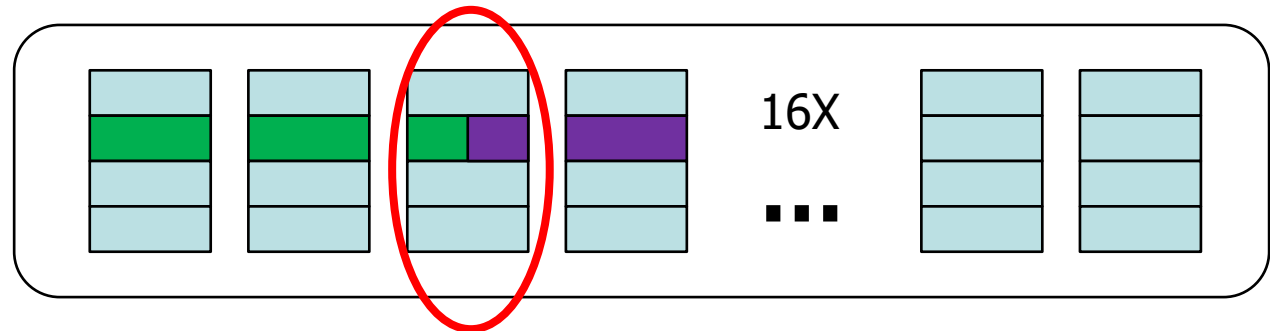


Design Challenge

- Various token size
 - Meta data: 1-3 bytes
 - Token size: 2-64K bytes
- Parallelize token execution
 - BRAM bank conflict

BRAM Bank Conflict

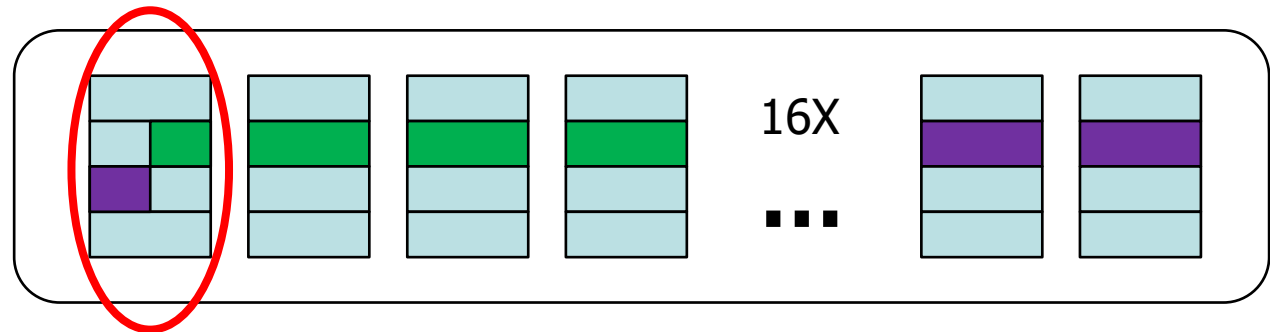
- Write – Write
 - Same block, same line



BRAMs

BRAM Bank Conflict

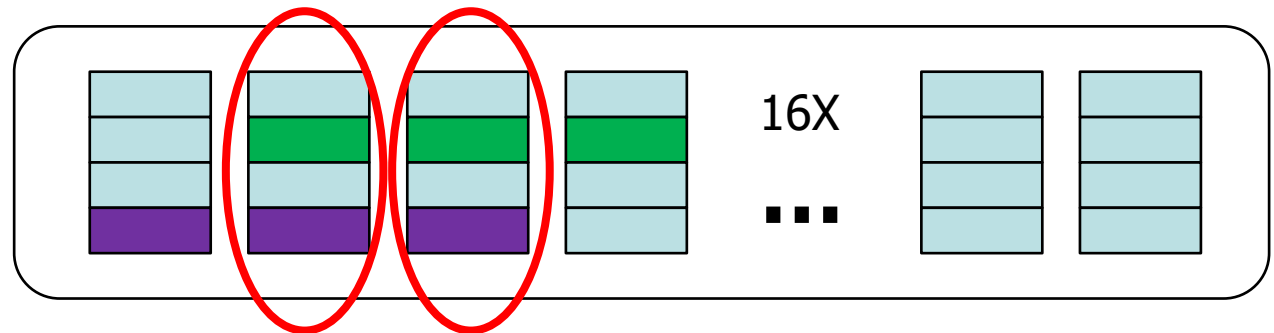
- Write – Write
 - Same block, different lines



BRAMs

BRAM Bank Conflict

- Read – Read



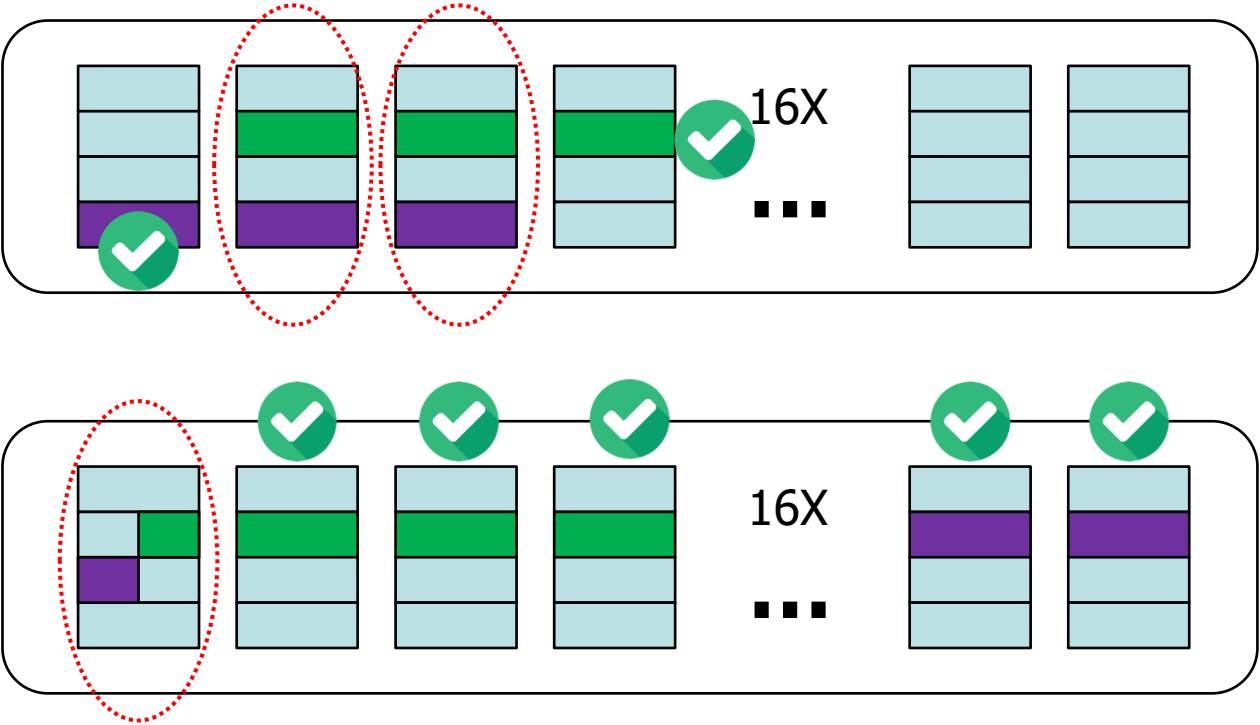
BRAMs

Design Challenge

- Various token size
 - Meta data: 1-3 bytes
 - Token size: 2-64K bytes
- Parallelize token execution
 - BRAM bank conflict
- Read-after-write dependency
 - Stall the pipeline

Refine Method

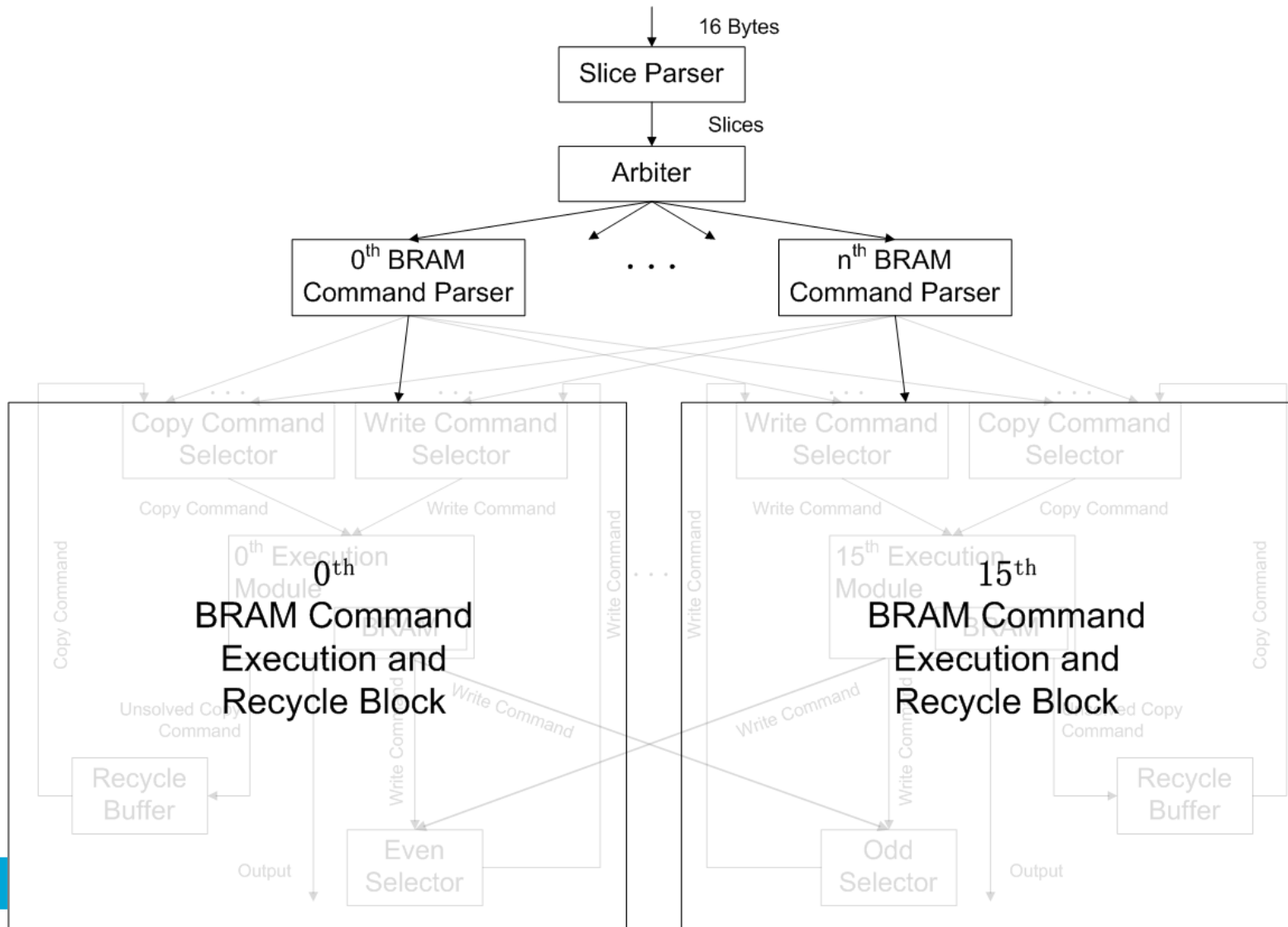
- Refine from token-level to BRAM bank-level
- Convert tokens into independent BRAM read/write commands



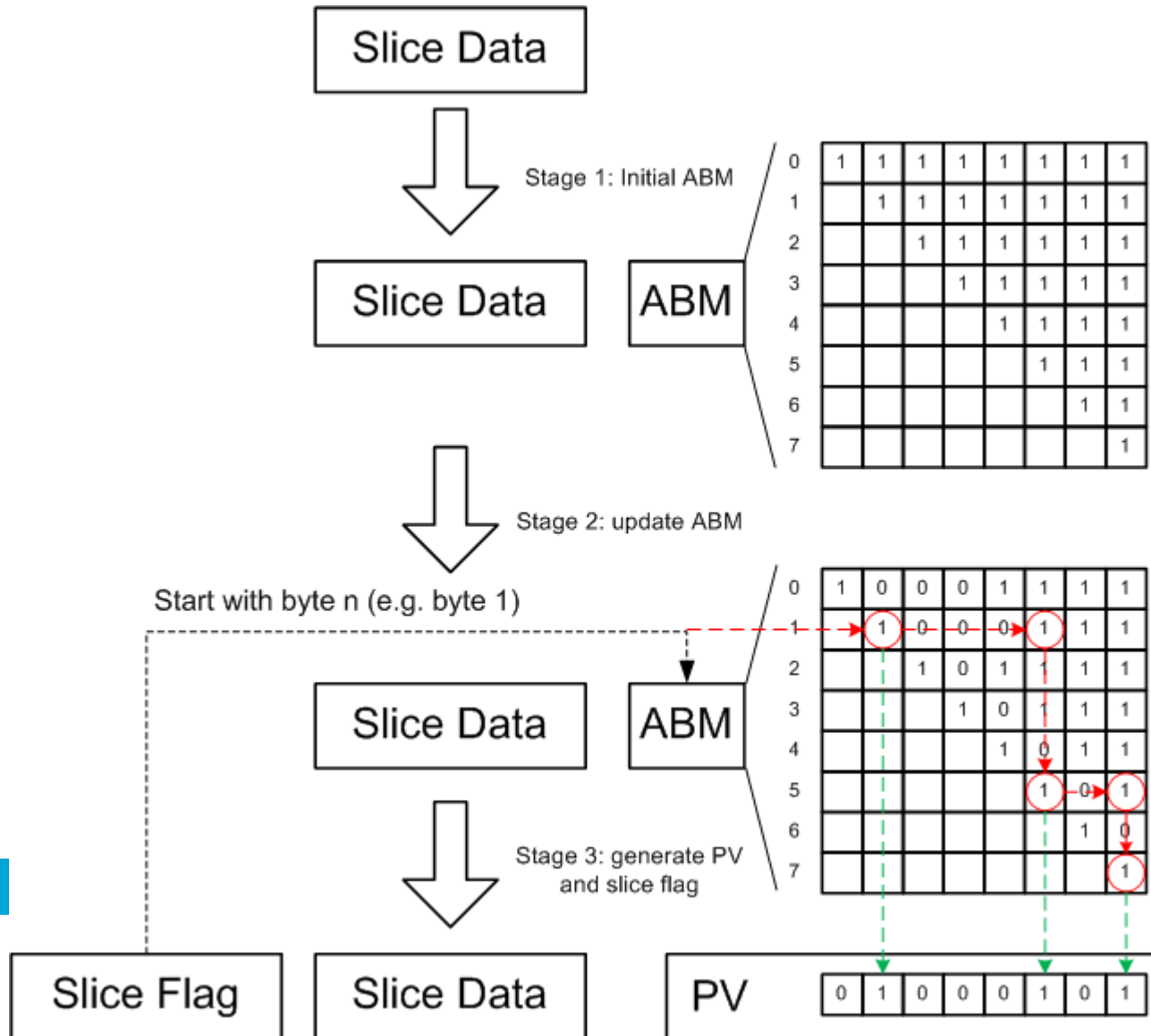
Recycle Method

- Stall?
 - -> throughput decrease
- Tomasulo/Scoreboarding?
 - Too complex
- Recycle and wait for next round execution

Architecture Overview



Slice Parser – find token boundary



Experiment

- Platform
 - ADM 9v3 (Xilinx VU3P FPGA)
 - CAPI 2.0 Interface (effective data rate 11GB/s)
 - Power 9 22-core CPU in Ubuntu 18.04.1 LTS

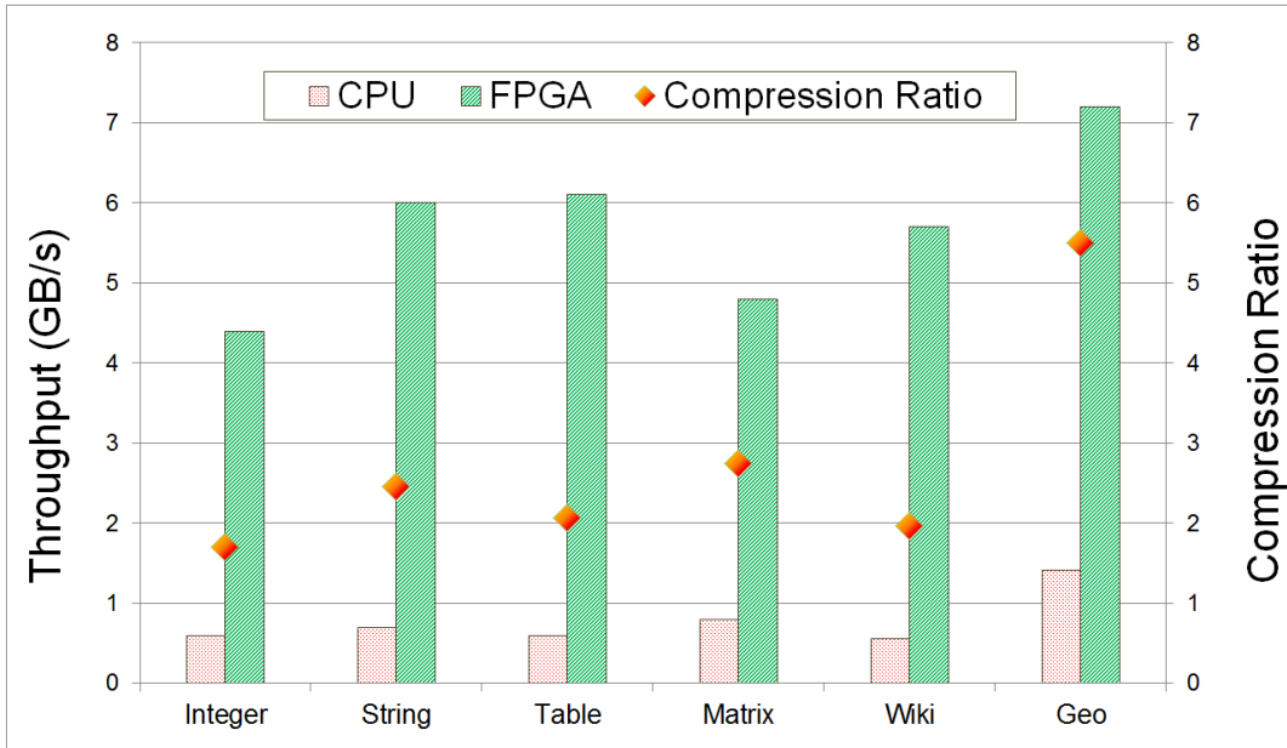
- Benchmark
 - TPC-H (two different column in “lineitem” table and the whole table)
 - XML from Wiki
 - Matrix data

Experiment - Results

| Resource | LUTs | BRAMs ¹ | Flip-Flops |
|-----------------|-------------|--------------------|-------------|
| Recycle buffer | 1.1K(0.3%) | 8(1.2%) | 1K(0.1%) |
| Decompressor | 56K(14.2%) | 50(7.0%) | 37K(4.7%) |
| CAPI2 interface | 82K(20.8%) | 238(33.0%) | 79K(10.0%) |
| Total | 138K(35.0%) | 288(40.0%) | 116K(14.7%) |

¹ One 18kb BRAM is counted as a half of one 36kb BRAM.

- End-to-end Throughput



- 250MHz
- 14% LUTs
- Up to 7.2GB/s
- 10x faster than CPU
- 10x more power efficient

Summary

- A Refine method to relieve BRAM bank conflicts
- A Recycle method to reduce impact of RAW dependency
- Assumption Bit Map to handle byte split in high speed
- 7.2 GB/s with 14.2% logic resource and 7% BRAM resource
- 5 engines can saturate OpenCAPI bandwidth
- 10x faster and 10x more power efficient

Thank You



**Open
Source**

<https://github.com/ChenJianyunp/FPGA-Snappy-Decompressor.git>