中国科学院高能物理研究所
Institute of High Energy Physics
Chinese Academy of Sciences

高能所计算中心
IHEP Computing Center

Design and implementation of Zstd compression algorithm for HEP data processing based on FPGA

Xuyang Zhou (zhouxuyang@ihep.ac.cn)

Computing Center, Institute of High Energy Physics, CAS

2022-10-25

Outline

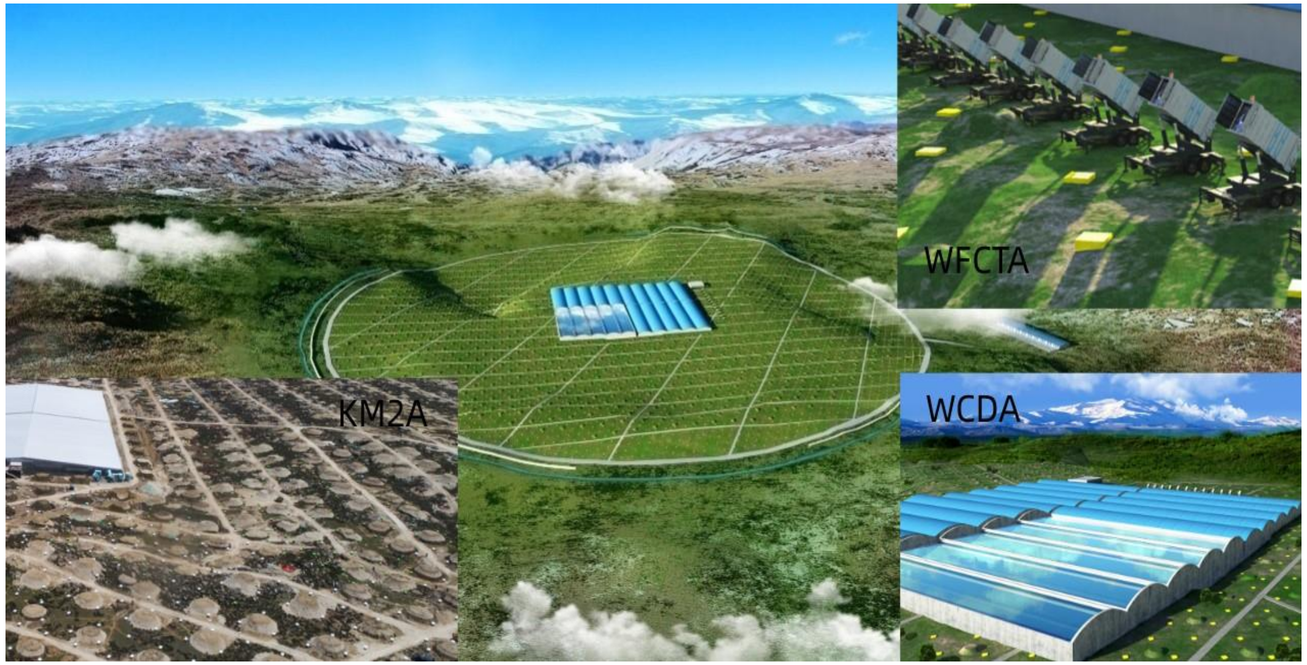
Background

- Massive experimental data of high energy physics & I/O bottleneck
  - PB or even EB level data
  - The I /O bottleneck has become one of the main problems faced by high-energy physical data processing.
- ROOT is to directly use the CPU of the computing cluster for compression
- Computational Storage is a "storage computing" technology
- We design a new architecture of Zstd compression kernel based on FPGA, and combine it with ROOT software framework

LHAASO Introduction

- LHAASO (Large High Altitude Air Shower Observatory) is the highest, largest and most sensitive cosmic ray detector in the world
- LHAASO: About 9PB per year
  - WCDA : ~ 10TB/day
  - KM2A : ~ 2.5TB/day
  - WFCTA : ~ 300GB/day

WFCTA



KM2A

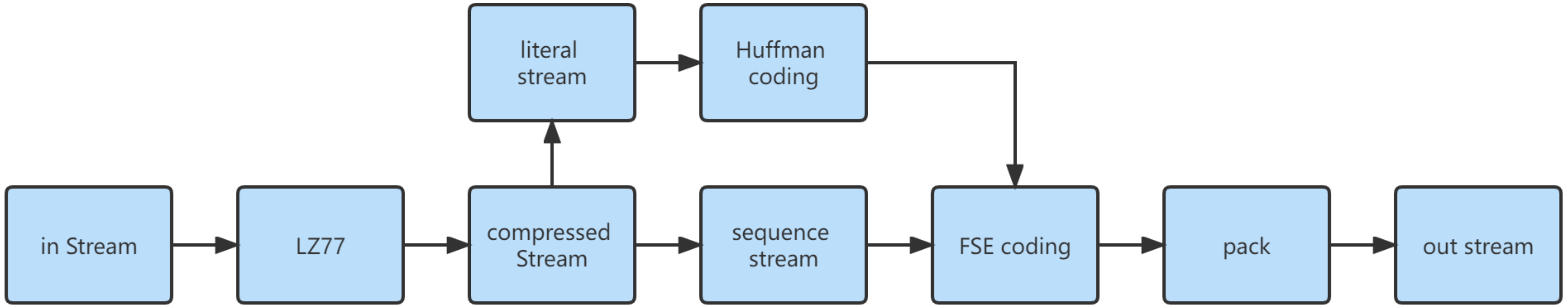
WCDA

LHAASO Decode

- The LHAASO data decoding task is to convert the data acquired by the DAQ system into a ROOT file
- The data compression takes more than 1/3 of the time
- LHAASO's experiment requires that all events be stored in offline analysis
- Challenges
  - CPU pressure, computing cluster queuing, slow compression…

Zstd Introduction

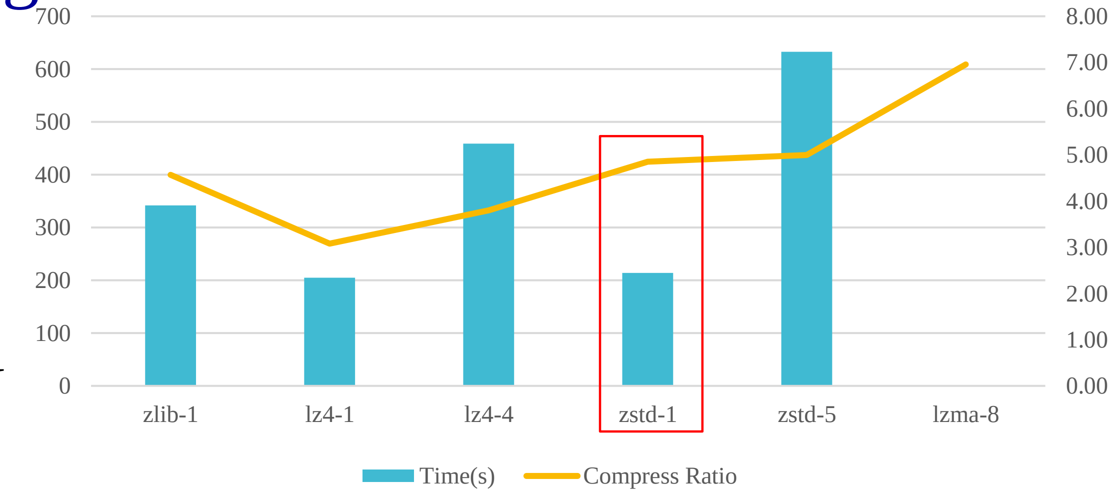
- Zstd is a fast lossless compression algorithm created by Facebook, targeting real-time compression scenarios at zlib-level and better compression ratios



[GitHub - facebook/zstd: Zstandard - Fast real-time compression algorithm]

Why do we do this?

- Explore the new data processing mode under the "storage and computing integration" architecture
- Explore and research the integration of computational storage technology and FPGA heterogeneous data processing
- In this case , we use Zstd algorithm
- Using FPGA to accelerate algorithm is the trend in the future
  - large number of CU and much faster than CPU
  - Programmable, flexible circuit modification
  - Power efficiency ratio better than GPU

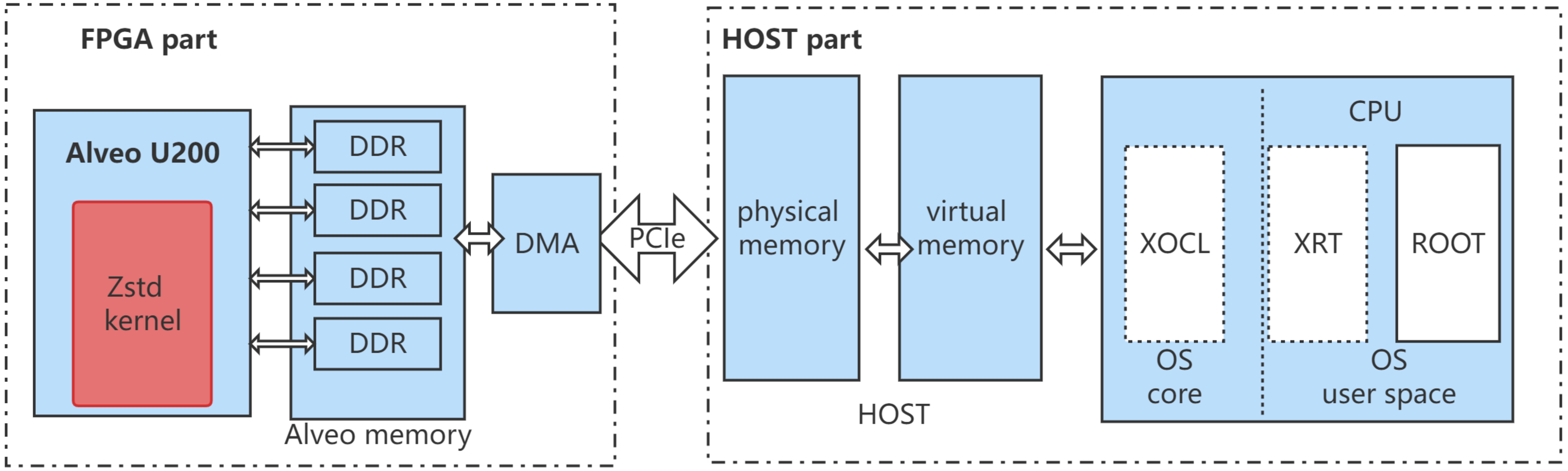


Zstd vs other compress algorithms in ROOT

Outline

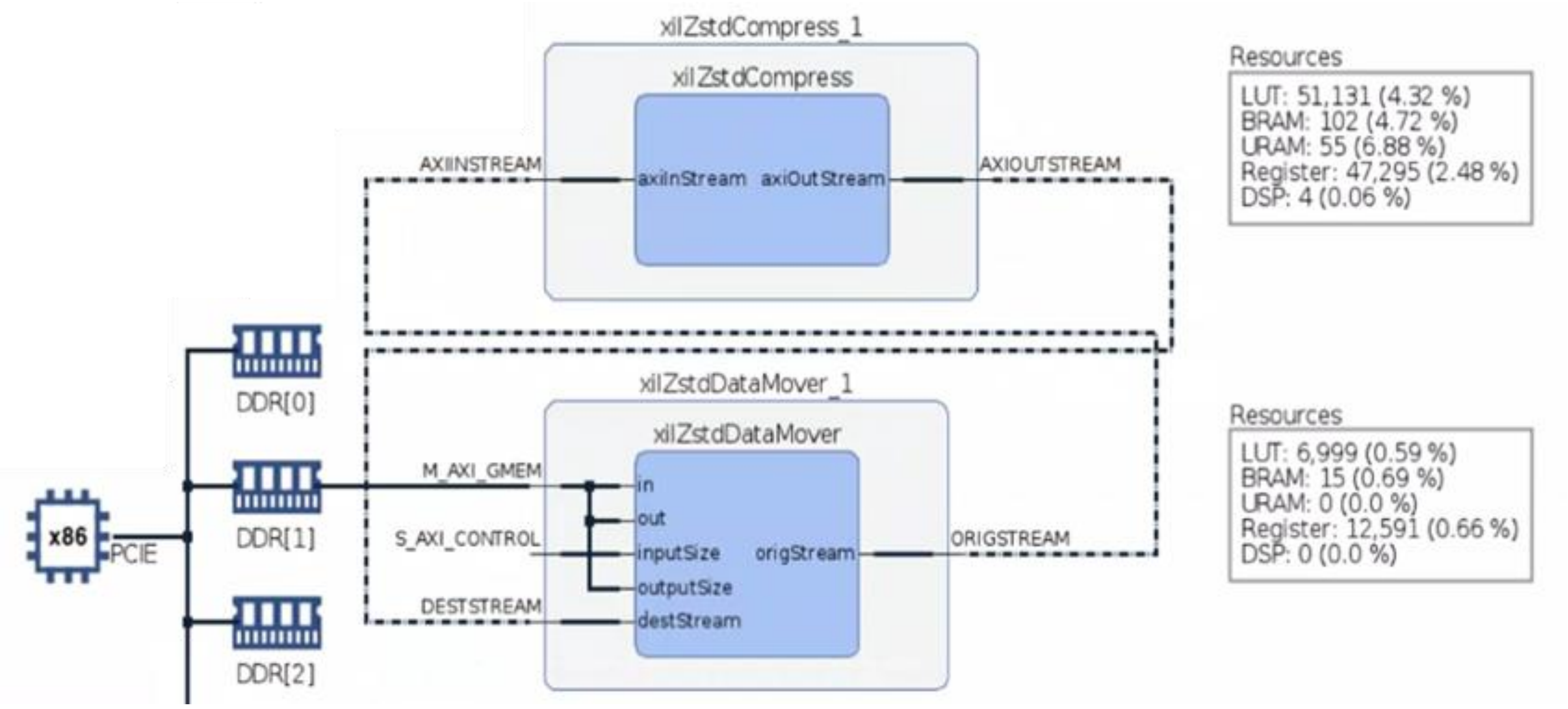
Overall design

- FPGA part : Implement the Zstd algorithm
- HOST part : The operation of other parts of data decoding
- FPGA and Host communicate via PCIe



Zstd Based on FPGA

- Use Vitis HLS to implement the Zstd algorithm kernel on U200



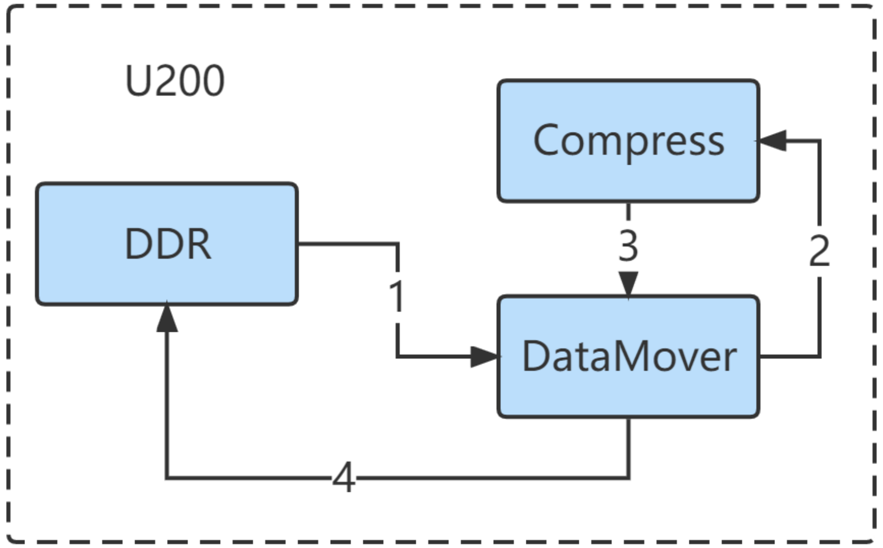
Resources

LUT: 51,131 (4.32 %)
BRAM: 102 (4.72 %)
URAM: 55 (6.88 %)
Register: 47,295 (2.48 %)
DSP: 4 (0.06 %)

Resources

LUT: 6,999 (0.59 %)
BRAM: 15 (0.69 %)
URAM: 0 (0.0 %)
Register: 12,591 (0.66 %)
DSP: 0 (0.0 %)

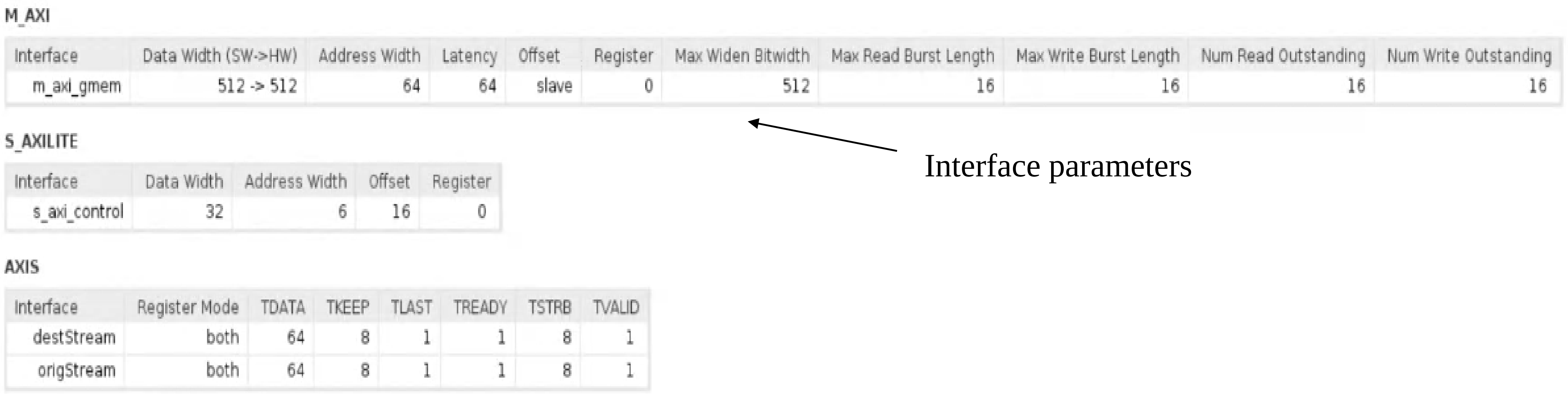
- Module Flow Chart



DataMover Module(1/2)

- Three protocols are used:
  - AXI: The data is accessed by the kernel through memory(DDR)
  - AXI-Stream: Data transmission between kernels
  - AXI-Lite: Register reads/writes
- DataMover Module Process
  1. Read data(512 bits) from device DDR
  2. Convert the data to AXI-Stream format and transfer it to the Compress module
  3. Then convert the AXI-Stream to data(512 bits)， then write to device DDR

DataMover Module (2/2)



M_AXI

Interface	Data Width (SW->HW)	Address Width	Latency	Offset	Register	Max Widen Bitwidth	Max Read Burst Length	Max Write Burst Length	Num Read Outstanding	Num Write Outstanding
m_axi_gmem	512 -> 512	64	64	slave	0	512	16	16	16	16

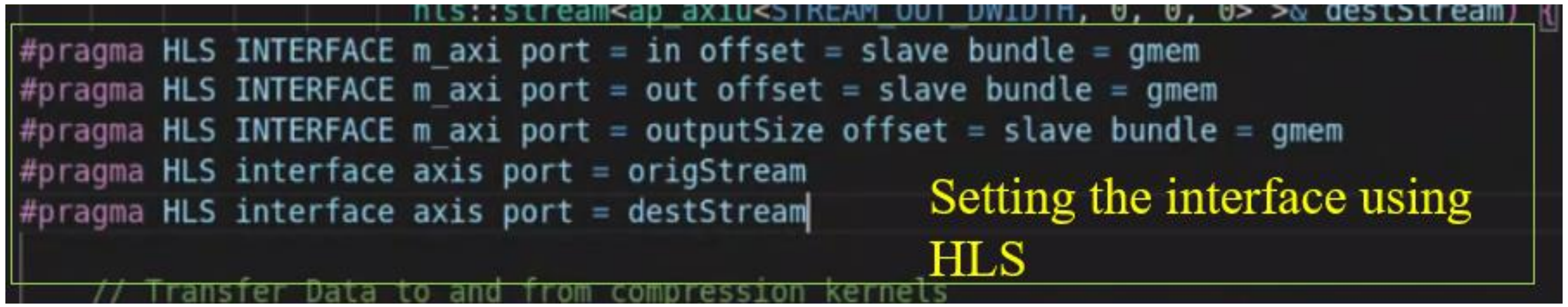
S_AXILITE

Interface	Data Width	Address Width	Offset	Register
s_axi_control	32	6	16	0

AXIS

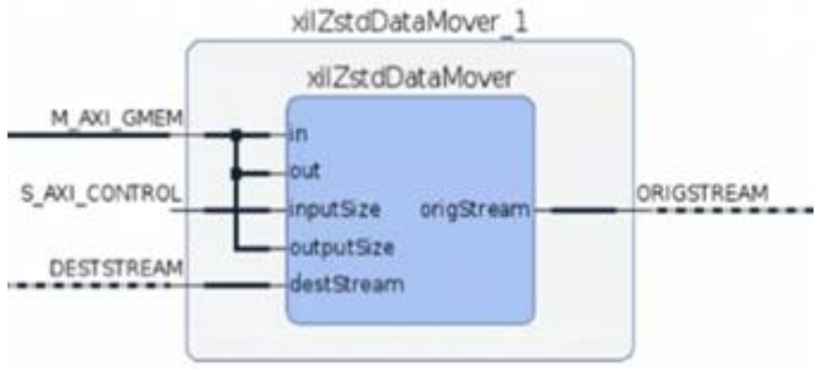
Interface	Register Mode	TDATA	TKEEP	TLAST	TREADY	TSTRB	TVALID
destStream	both	64	8	1	1	8	1
origStream	both	64	8	1	1	8	1

Interface parameters



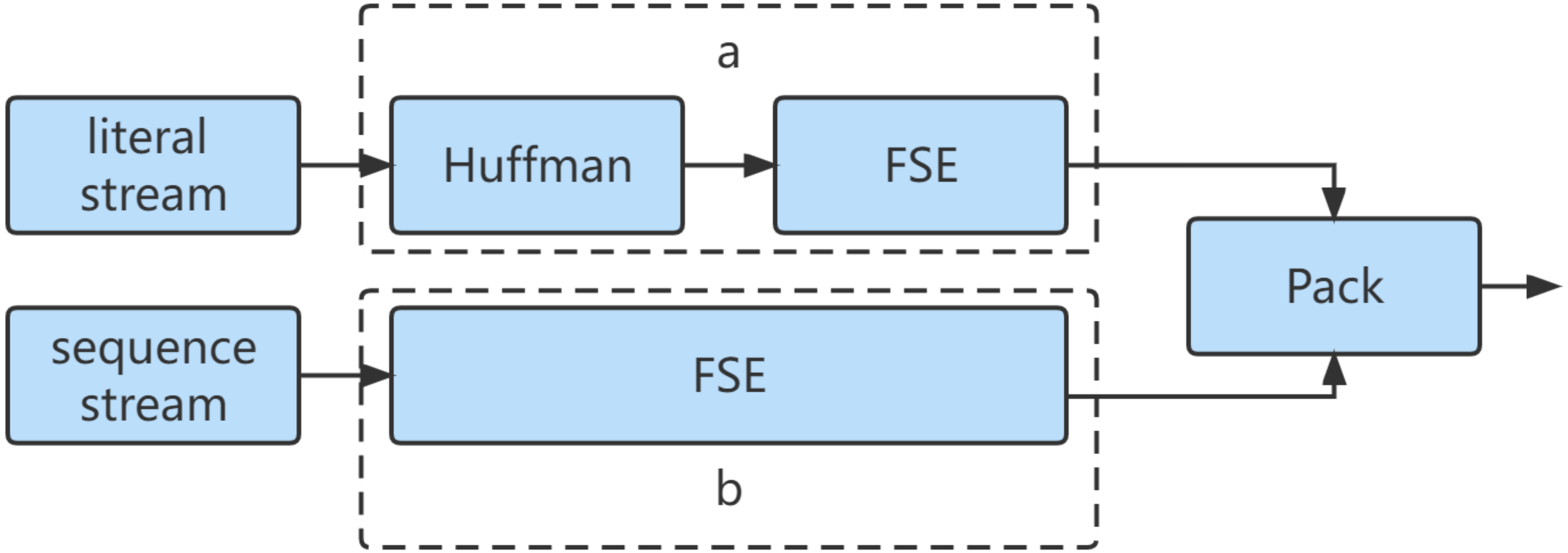
```
        hls::stream<ap_axiu<STREAM_OUT_DWIDTH, 0, 0, 0> >& destStream) {
#pragma HLS INTERFACE m_axi port = in offset = slave bundle = gmem
#pragma HLS INTERFACE m_axi port = out offset = slave bundle = gmem
#pragma HLS INTERFACE m_axi port = outputSize offset = slave bundle = gmem
#pragma HLS interface axis port = origStream
#pragma HLS interface axis port = destStream

    // Transfer Data to and from compression kernels
```

Setting the interface using HLS



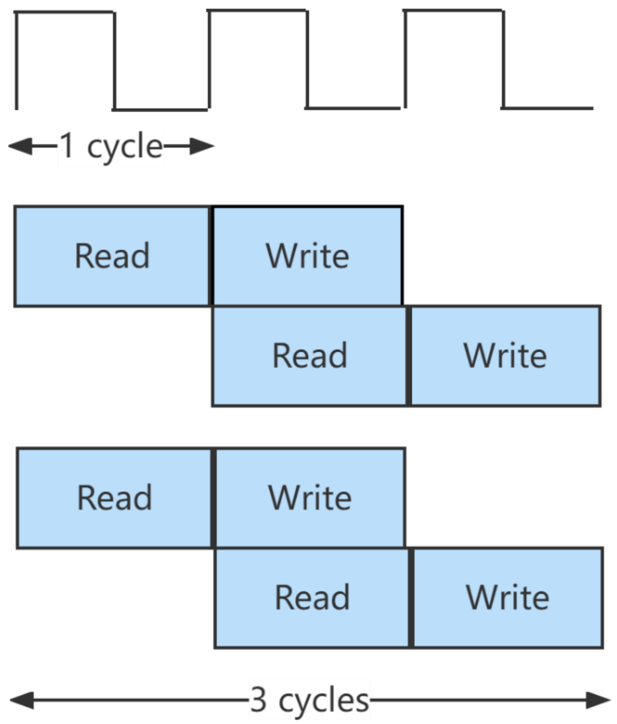
Compress Module (1/2)

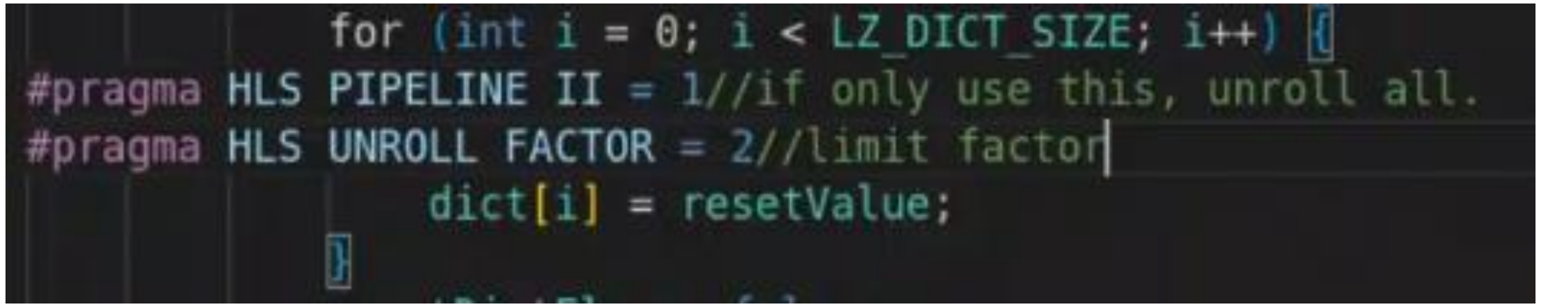
- Consist of LZ77, Huffman coding and FSE coding
  1. Perform LZ77 compression in parallel after splitting the data
  2. Huffman coding and FSE coding:
     a. Sequence stream coding
     b. Literal stream Huffman coding， then FSE coding
     c. a and b execute in parallel
  3. Pack



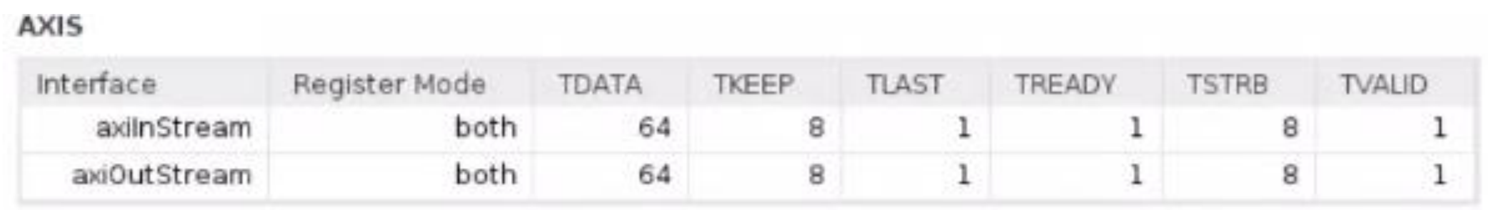
ZstdCompress Module (2/2)

- Three HLS pragma
  - pragma HLS dataflow: Task-level pipelining
  - pragma HLS pipeline: Allow the concurrent execution
  - pragma HLS unroll: unroll loops for parallelism



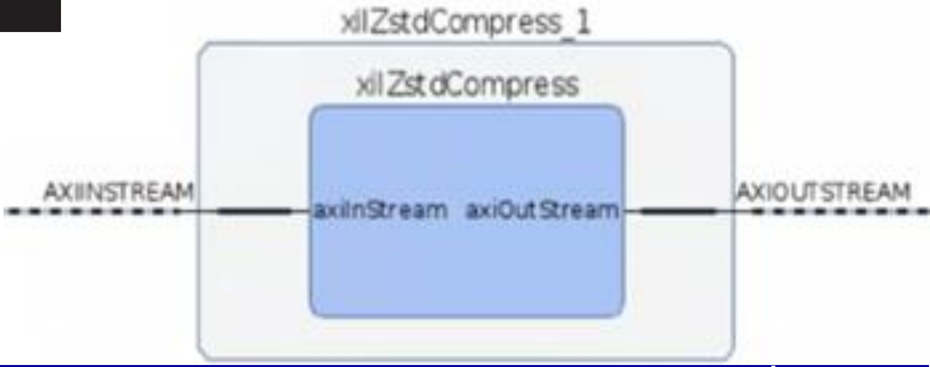


```
            for (int i = 0; i < LZ_DICT_SIZE; i++) {
#pragma HLS PIPELINE II = 1//if only use this, unroll all.
#pragma HLS UNROLL FACTOR = 2//limit factor
                dict[i] = resetValue;
            }
```



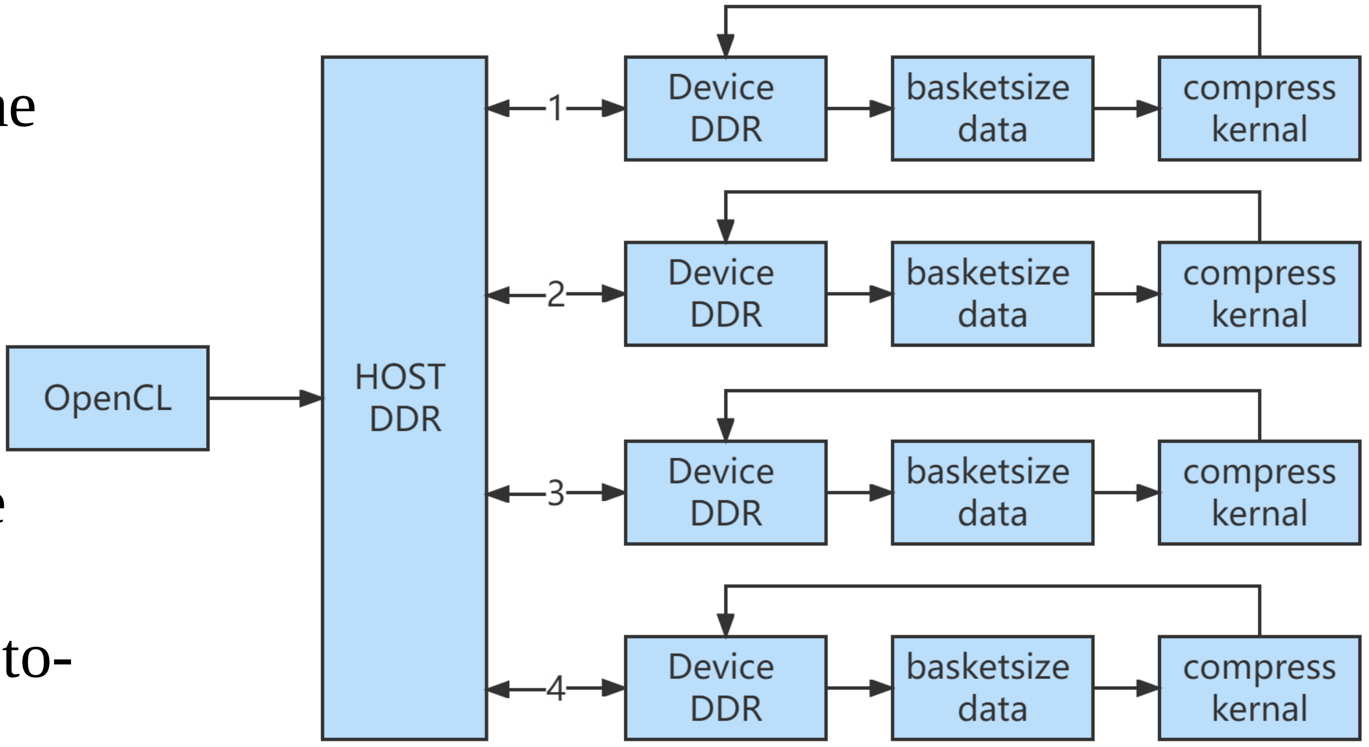
AXIS

Interface	Register Mode	TDATA	TKEEP	TLAST	TREADY	TSTRB	TVALID
axiInStream	both	64	8	1	1	8	1
axiOutStream	both	64	8	1	1	8	1



Host and FPGA data interaction

- ROOT compression
  - Compress one basket size at a time
  - A serial process by default
  - Suitable for large clusters sharing computing resources
- Our method
  - Spatial Data Parallelism: Increase Number of Compute Units
  - Temporal Data Parallelism: Host-to-Kernel Dataflow

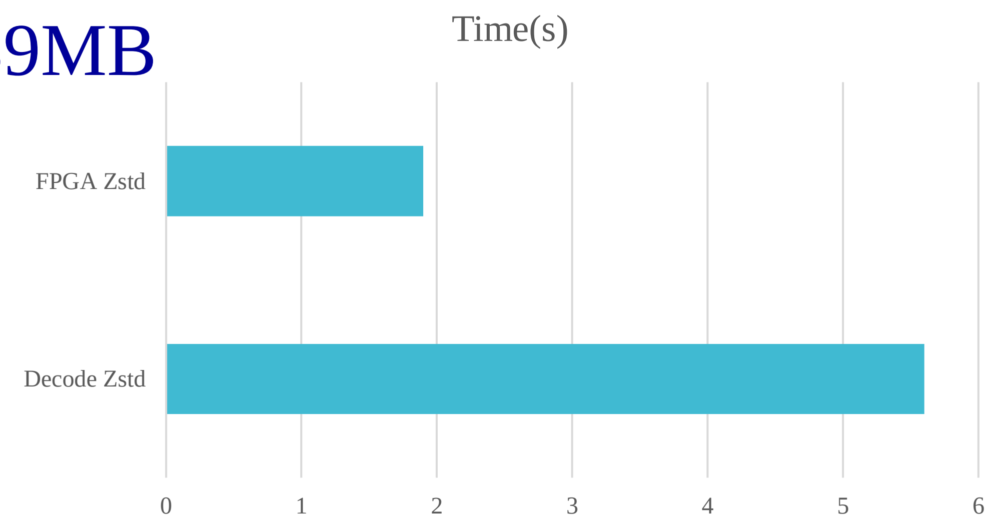


Temporal Data Parallelism

Spatial Data Parallelism

Compression Effect

- Environment for testing
  - HOST : 2 x Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz， memory： 128GB
  - FPGA : Xilinx Alveo U200
  - ROOT : 6.22/06
  - Zstd : v1.5.2
- Raw data: 960MB→ROOT data: 389MB
  - KM2A Decode :
    - Total time :11.9s
    - Compress time:5.6s
  - FPGA Zstd :
    - Time :1.9s



The compression speed of FPGA zstd is about 3 times that of software zstd

Outline

Summary

- FPGA Zstd compression speed is much faster than software Zstd
- We have preliminarily implemented the application of Zstd algorithm based on FPGA in LHAASO-KM2A detector
- Focuses more on compression rather than decompression, only compression has been completed now

Next steps

- Divide the DataMover module into two modules: DataMoverIn module and DataMoverOut module
- Try to use FSE coding only to achieve faster compression speed
  - Cannot use the general software zstd to decompress it, but it is acceptable
  - Migrate the algorithm to our own customized server for online use.

高能所計算中心
IHEP Computing Center

Thank You!

Questions?