Decompressing Snappy Compressed Files at the Speed of OpenCAPI

Speaker: Jian Fang
TU Delft
Current Project

SHADE

Scalable
Heterogeneous
Accelerated
Database

NVMe

OpenCAPI

SNAP

FPGA

Decompression

Fletcher

OpenCAPI

ARROW

Memory

Tables

CPU

POWER9

Sort

Join

Skyline

Spark

DB

DNA

Seq

GPU

HBM

HBM

Actions

OpenCAPI

Scalable Heterogeneous Accelerated Database
Big Data Processing

Database Analytics

Data Movement

Sort  Join  Group

Filter  Learning

Data reformat  Aggregation
• Moving Data from/to Storage
  • Slow, Bottleneck
  • 1st step in data processing and database analytics

• Decompress-filter Solutions

[Diagram showing decompression and filtering solutions including GZIP, LZ4, RLE, LZW, Compressed Data, Decompressed Data, Filtered Data, and Netezza]
Snappy compression Algorithm

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

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

  ABCDEFGHABCDEFGH
  offset 8, length 4

  Use (8, 4) to replace “ABCD”
Snappy Compression

Input →

Generate a literal token →

Find next match →

Generate a copy token →

copy : offset 8 length 4

Output →

... (Literal, “ABCDEFGH”) (Copy, 8, 4)

No Match!

Match!
Snappy Decompression

... ABCDEFGH

(Copy, 8, 4)

(Literal, “ABCDEFGH”)
Snappy compression Algorithm

- An independent 64KB history for every 64KB data
- Token: copy or literal, copy length <= 64B
- Token size: 2 Byte minimum, average ≈< 4 Byte (assumption for design)
- Highly data dependent

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Compression Ratio</th>
<th>Average Token Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNA1</td>
<td>3.31</td>
<td>5.79</td>
</tr>
<tr>
<td>DNA2</td>
<td>2.36</td>
<td>7.38</td>
</tr>
<tr>
<td>DNA3</td>
<td>2.57</td>
<td>7.03</td>
</tr>
<tr>
<td>TPC-H</td>
<td>2.53</td>
<td>2.72</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Design in FPGA

- An independent 64KB history for every 64KB data
- Token: copy or literal, copy length <= 64B
- Token size: 2 Byte minimum, average ≈< 4 Byte (assumption for design)
- Highly data dependent

Design Challenge

- Deal with different types of tokens
- Handle various size of tokens
- Parallelize token translation
- Keep up with the interface bandwidth
Design Challenge

- Deal with different types of tokens
- Handle various size of tokens
- Parallelize token translation
- Keep up with the interface bandwidth
Design Challenge

• Deal with different types of tokens

• Handle various size of tokens

• Parallelize token translation

• Keep up with the interface bandwidth
Token Structure

**Literal Token**

<table>
<thead>
<tr>
<th>len-1</th>
<th>00</th>
<th>literal content</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag byte</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>60</th>
<th>00</th>
<th>len-1 (8 bits)</th>
<th>literal content</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag byte</td>
<td>extra byte</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>61</th>
<th>00</th>
<th>lower 8 bits of len-1</th>
<th>higher 8 bits of len-1</th>
<th>literal content</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag byte</td>
<td>extra bytes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Token Structure

<table>
<thead>
<tr>
<th>higher 3 bits of offset</th>
<th>cpy_len-4 (3 bits)</th>
<th>01</th>
<th>lower 8 bits of offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag byte</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>extra byte</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cpy_len-1 (6 bits)</th>
<th>10</th>
<th>lower 8 bits of offset</th>
<th>higher 8 bits of offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag byte</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>extra bytes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tokens are in various length

Prior Solutions: Decode all possibilities

Design Challenge

- Deal with different types of tokens
- Handle various size of tokens
- Parallelize token translation
- Keep up with the interface bandwidth
Token Dependency

- Read – Read
  - No dependency
- Write – Write
  - Never write same bytes
  - No dependency
- Write – Read
  - Forwarding
Token Dependency – Address Conflict

- Write – Write
  - Same block, same line
Token Dependency – Address Conflict

- Write – Write
  - Same block, different lines

![Diagram showing BRAMs with write-write conflict]

BRAMs
Token Dependency – Address Conflict

- Read – Read
Design Challenge

- Deal with different types of tokens
- Handle various size of tokens
- Parallelize token translation
- Keep up with the interface bandwidth

Token size: 4B
2 tokens/cycle
250MHz
--> 13 instances

token size * token/cycle * frequency * instances
= OpenCAPI BW
IDEA – from token-level to BRAM-level
Architecture Overview
Architecture Overview

- Two-level Parser
Architecture Overview

- Two-level Parser
Architecture Overview

- Two-level Parser
- Parallel BRAM Operations
Architecture Overview

- Two-level Parser
- Parallel BRAM Operations
- Relax Execution Model
Parallel BRAM Operations & Relax Execution Model

BRAM

<table>
<thead>
<tr>
<th>Byte0</th>
<th>Byte1</th>
<th>Byte2</th>
<th>Byte3</th>
<th>Byte4</th>
<th>Byte5</th>
<th>Byte6</th>
<th>Byte7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Data</td>
<td>Data</td>
<td>Data</td>
<td>invalid</td>
<td>invalid</td>
<td>invalid</td>
<td>invalid</td>
</tr>
</tbody>
</table>

Write command

Copy command

MUX

Write port

Read port

Recycle FIFO

Write command

Write command

Copy command
Results

- Implementation Result (on XCKU15P)

<table>
<thead>
<tr>
<th>Flip-Flop</th>
<th>LUTs</th>
<th>BRAMs</th>
<th>Frequency</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>32K(3.32%)</td>
<td>46K(8.95%)</td>
<td>29(2.95%)</td>
<td>250MHz</td>
<td>2.9W</td>
</tr>
</tbody>
</table>

- Benchmark

  - Alice’s Adventures in Wonderland: 85KB (149KB) for compressed (uncompressed) file.

- Throughput

<table>
<thead>
<tr>
<th></th>
<th>FPGA (single decompressor)</th>
<th>CPU (Core i7 with one thread)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>3GB/s</td>
<td>300MB/s</td>
</tr>
<tr>
<td>Output</td>
<td>5GB/s</td>
<td>500MB/s</td>
</tr>
</tbody>
</table>
Thank You

Open Source

Publish Paper:
J. Fang, J. Chen, P. Hofstee, Z. Al-Ars, J. Hidders,
A High-Bandwidth Snappy Decompressor in Reconfigurable Logic (CODES+ISSS 2018)

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

Contact Me: j.fang-1@tudelft.nl (Jian Fang)