Accelerated Entropy Coding for ALICE O²

Michael Lettrich (CERN, Technische Universität München) for the ALICE collaboration

28.06.2022
Recap: Requirements

- Efficient, lossless compression of CTF payload at the end of Synchronous Reconstruction step
- Achieve high compression under firm real-time conditions:
  - Compression factor $\sim 3$ (TDR)
  - Time $\sim 35$ s (TDR)
- Data: Structures of Integer Arrays:
  - Different ranges of data (4 Bit/int – 32 Bit/int )
  - Signed/Unsigned
  - Known at compile time
- Per detector compression + merging
Recap: Classes of Compression Algorithms

• **Packing** (Google Varint,...)
  - Pack + shift, Varint

• **Universal Codes** (Elias $\gamma$, $\delta$, $\omega$, Golomb,...)
  - Longer Codes for longer Integers

• **Dictionary Coders** (Lempel-Ziv-Welch,...)
  - Replace sequence of symbols by reference to dictionary

• **Entropy Coders** (Huffman, Arithmetic, ANS,...)
  - Transform source data using distribution of symbols (information content)
  - Length of encoded symbol depends on symbol probability (shorter codes for more frequent symbols)
Recap: Classes of Compression Algorithms

• General Purpose coders (gzip, zstd, lzma) combine algorithms
  • First pass dictionary coding second pass entropy coding
  • Goal: Compress by finding structure in arbitrary data, then store it efficiently

• ALICE: Structure of data is well known
  • Use entropy coders for best possible compression where appropriate
  • Provide a flexible framework that allows to exploit the structure of our data
  • Fast implementation
  • Entropy coding using rANS for us provides best tradeoff between performance and flexibility
rANS Encoding

Reuse of dictionaries is possible (tradeoff compression quality/ metadata size)

Build Frequency Table → Build Dictionary

While Input

N

O

Store Metadata

Yes

No

Lookup Symbol in Dictionary

Renorm/ Stream Out

Encode
Accelerating rANS: Challenges

- Slow, difficult to vectorize
- Reuse of dictionaries is possible (tradeoff compression quality/ metadata size)
- Compression rate is highly sensitive towards bad approximation
- Compression Quality
- Performance
- Memory Footprint
- Inefficient Storage of Metadata
- parallel Streams in rANS via SIMD
- Heavy on arithmetics (integer division)
- Improve Lookup speed/ memory footprint
- Maintain Compatibility across platforms

Flowchart:

1. Build Frequency Table
2. Build Dictionary
3. While Input
   - Yes: Lookup Symbol in Dictionary
   - No: Store Metadata
4. Encode
   - Renorm/Stream Out
5. Optional: Maintain Compatibility across platforms
6. Optional: Improve Lookup speed/ memory footprint
7. Optional: Reduce Inefficient Storage of Metadata
8. Optional: Simplify Compression Quality

05.04.2022
M Lettrich for ALICE - Entropy coding for ALICE Run 3
Multistream rANS Encoders
serial, SSE4.1, AVX2

• Up to 16 interleaved streams writing into same datastream
• No changes to encoding efficiency
• Encoded Data remains decodable on any platform
• STL like interface: Random Access Iterators, faster with pointers
• Shared, modular codebase to ensure compatibility, maintainability and facilitate testing
Multistream rANS Encoders
serial, SSE4.1, AVX2

• Serial based on ryg_rans, precomputed divides
• SSE/AVX coders: internal fp64 arithmetic
• speedup 20% for SSE, 70% microarchitecture utilization
• speedup up to 2x using AVX2, 70% microarchitecture utilization
• SSE/AVX reduce RAM usage by 3x
Dictionary Improvements

• Reimplement frequency table creation (Speedup up to 8x)
• Additional Hash table based dictionaries for large, sparse ranges (32 Bit/symbol)
• Improved renorming:
  • More accurate estimation of renorming and cutoff thresholds
  • Incompressible dataset detection
• Drastically improved dictionary serialization (factor 1700x compression over current naïve solution)
rANS Encoding

**Variant A: External Dictionary**

1. Load Dictionary
2. Build Encoder
3. Encode

**Variant B: TF specific dictionary**

1. Build Frequency Table
2. Compressible?
3. Yes: Build Dictionary, Encode, Store Metadata
4. No: PackData
Synthetic Benchmarks

- Binomial distribution $\mathcal{B}()$
- 8 Bit, 16 Bit, 32 Bit source data
- 128 MB of raw data
- 1000 Iterations
- Google Benchmark for timing
- Pure Encode Timing
- System: Core i7 9700K, 4.8 GHz, 32GB RAM, Ubuntu 20.04, GCC 10.2
- Flags: std=c++17 -O2 -march=native
TPC Pb-Pb Benchmarks

- Full Simulated Timeframe TPC Pb-Pb
- System: Core i7 9700K, 32GB RAM, Ubuntu 20.04, GCC 10.2 alidist
- Flags: std=c++17 -O2 -march=native
- Literal Encoders, exclude output buffer allocation in timings
- Excluded timeA as it is incompressible
# TPC Pb-Pb Compression Metrics

<table>
<thead>
<tr>
<th></th>
<th>Size</th>
<th>Entropy</th>
<th>Buffer</th>
<th>Literals</th>
<th>Total</th>
<th>Compression</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>3459.89</td>
<td>1161.96</td>
<td>1160.70</td>
<td>2.80</td>
<td>1163.50</td>
<td>2.97</td>
<td>1.0013</td>
</tr>
<tr>
<td>Dictionary</td>
<td>64.83</td>
<td>-</td>
<td>0.038</td>
<td>-</td>
<td>0.038</td>
<td>1696.01</td>
<td>-</td>
</tr>
</tbody>
</table>
TPC Pb-Pb Compression Performance

Time [ms]

- Current: Encoding 4686, Dictionary 183, FrequencyTable 5124
- SSE: Encoding 3749, Dictionary 145, FrequencyTable 834
- AVX: Encoding 2515, Dictionary 146, FrequencyTable 834
Recommended Use

• Header-only template library
• Compile time configuration of encoders
  • tag based interface
  • Simplified configuration via factories and smart presets

Compatibility
• API not fully compatible, but requires only minor adjustments
• Legacy Encoders/Decoders maintained to ensure full compatibility
• Integration path:
  • Bump rANS Version in CTF
  • Minor, cosmetic changes to CTFCoder and EncodedBlocks

05.04.2022
M Lettrich for ALICE - Entropy coding for ALICE Run 3
Compatibility / Integration

• Header-only template library
• Compile time configuration of encoders
  • tag based interface
  • Simplified configuration via factories and smart presets

Compatibility
• API not fully compatible, requires only minor adjustments
• Legacy Encoders/Decoders maintained to ensure full compatibility
• Integration path:
  • Bump rANS Version in CTF
  • Minor, cosmetic changes to CTF Coders and EncodedBlocks
Conclusion

- New rANS implementation
  - Unified codebase
  - Flexible Iterator-based API
  - Improved Performance
  - Maintain close-to-entropy compression
- Performance improvements allow on-the-fly generation of dictionaries
- Minimal API changes allow easy integration