Tile-based Lightweight Integer Compression in GPU

Anil Shanbhag, Bobbi W. Yogatama, Xiangyao Yu, Samuel Madden
**MOTIVATION**

- GPUs have limited memory capacity
- Typically ~80 GB of HBM (High Bandwidth Memory)

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Bandwidth</td>
<td>100 GBps</td>
<td>2 TBps</td>
</tr>
<tr>
<td>Compute</td>
<td>&lt; 1 TFLOPs</td>
<td>19.5 TFLOPs</td>
</tr>
</tbody>
</table>
COMPUTE INTENSITY

• How many operations must I do on some data to make it worth the cost of loading it?

• Compute Intensity = \frac{FLOPs}{Data Rate}

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Bandwidth</td>
<td>100 GBps</td>
<td>2 TBps</td>
</tr>
<tr>
<td>Compute</td>
<td>&lt; 1 TFLOPs</td>
<td>19.5 TFLOPs</td>
</tr>
<tr>
<td>Compute Intensity</td>
<td>~ 100</td>
<td>~ 80</td>
</tr>
</tbody>
</table>
COMPRESSION SCHEMES

1. Fit more data in GPU memory
2. Speed up data transfer between CPU and GPU
COMPRESSION SCHEMES LIMITATIONS

1. Cascading multiple compression schemes cause multiple passes over the global memory, causing high memory traffic.
2. Bit-level packing is superior, but the SIMT model has a limited instruction set for bit-level alignment operations.
COMPRESSION SCHEMES TRADE-OFF

Compression Ratio

Decompression Speed
COMPRESSION SCHEMES IN COLUMNAR STORES

- Compression Schemes
  - Light-weight
    - FOR
  - Heavy-weight
    - DELTA
    - DICT
    - RLE
    - NS
Frame-of-Reference (FOR)

Useful when the integers have similar values
Delta Encoding (DELTA)

Useful when the integers are sorted or semi-sorted.
Delta Encoding (DELTA)

Useful when the integers are sorted or semi-sorted
### Dictionary Encoding (DICT)

The **Dictionary Encoding (DICT)** is a method used in data processing where a column is encoded using a predefined dictionary of values. This method is particularly useful when the column has a low cardinality, meaning there are few unique values.

#### Example

<table>
<thead>
<tr>
<th>Color</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>0</td>
</tr>
<tr>
<td>Green</td>
<td>1</td>
</tr>
<tr>
<td>Blue</td>
<td>2</td>
</tr>
<tr>
<td>Green</td>
<td></td>
</tr>
<tr>
<td>Blue</td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td></td>
</tr>
</tbody>
</table>

#### Dictionary

<table>
<thead>
<tr>
<th>Color</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>0</td>
</tr>
<tr>
<td>Green</td>
<td>1</td>
</tr>
<tr>
<td>Blue</td>
<td>2</td>
</tr>
</tbody>
</table>

Useful when the column has low cardinality.
Run-length Encoding (RLE)

Useful when the data has high average run-length

<table>
<thead>
<tr>
<th>Values</th>
<th>Run-length</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>2</td>
</tr>
<tr>
<td>99</td>
<td>4</td>
</tr>
<tr>
<td>111</td>
<td>4</td>
</tr>
</tbody>
</table>

101 101 99 99 99 99 111 111 111 111
Null Suppression (NS)

Useful when the integers have small values
Cascade Compression

DELTA FOR NSF
OBJECTIVES

1. Decompress in single pass over global memory & inline with query execution
2. Efficient bit-packing-based compression schemes
Single Pass over Memory & Inline with Query Execution

• Consider each **Thread Block** as the basic execution unit.
• Each thread block processes over single **Tile** of data.
GPU ARCHITECTURE

Ampere A100 GPU

- SM 0: regs (256k), L1S (192k)
- SM 1: regs (256k), L1S (192k)
- SM 2: regs (256k), L1S (192k)
- SM 3: regs (256k), L1S (192k)
- SM 107: regs (256k), L1S (192k)

L2 Cache (40MB)

HBM Memory (80GB)

A100 Streaming Multiprocessor (SM)

- 64 warps/SM
- 4x concurrent warp exec
- 64k x 4-byte registers
- 192KB L1/shared memory (configurable split)

Functional Units

Shared Memory / L1 Cache (192k)

The GPU runs threads in groups of 32 - each group is known as a warp
Single Pass over Memory & Inline with Query Execution

Figure 2: Decoding cascaded compression scheme (Delta+FOR+NSF) using cascaded decompression (left) and using tile-based decompression (right)
Single Pass over Memory & Inline with Query Execution

Figure 2: Decoding cascaded compression scheme (Delta+FOR+NSF) using cascaded decompression (left) and using tile-based decompression (right)
Efficient bit-packing-based compression schemes

- GPU-FOR
- GPU-DFOR
- GPU-RFOR
GPU-FOR

- Block Size = # of integers per block
- Miniblock Count = # of miniblocks per block
- Total Count = # of integers in data array

Figure 3: GPU-FOR Data Format
GPU-FOR

Figure 4: Example encoding with GPU-FOR

- Block Size = 16
- Miniblock Count = 2
- Total Count = 16
Algorithm 1: Fast Bit Unpacking on GPU — The following code runs on each of the 128 threads within a thread block in parallel.

**Input:** int[] block_starts; int[] data; int block_id; int thread_id

**Output:** int item

1. int block_start = block_starts[block_id];
2. uint * data_block = &data[block_start];
3. int reference = data_block[0];
4. uint miniblock_id = thread_id/32;
5. uint index_into_miniblock = thread_id & (32 - 1);
6. uint bitwidth_word = data_block[1];
7. uint miniblock_offset = 0;
8. for j = 0; j < miniblock_id; j++ do
9.     miniblock_offset += (bitwidth_word & 255);
10.    bitwidth_word >>= 8;
11. end for
12. uint bitwidth = bitwidth_word & 255;
13. uint start_bitindex = (bitwidth * index_into_miniblock);
14. uint header_offset = 2;
15. uint start_intindex = header_offset + miniblock_offset + start_bitindex/32;
16. uint64 element_block = data_block[start_intindex] | (((uint64)data_block[start_intindex + 1]) << 32);
17. start_bitindex = start_bitindex & (32-1);
18. uint element = ((element_block & ((1<<bitwidth) - 1) << start_bitindex)) >> start_bitindex;
19. item = reference + element;
Algorithm 1: Fast Bit Unpacking on GPU — The following code runs on each of the 128 threads within a thread block in parallel.

**Input:** int[] block_starts; int[] data; int block_id; int thread_id

**Output:** int item

1. Identify block_start = block_starts[block_id]
2. Read bit_width word
3. Compute miniblock offset
4. Compute offset within the miniblock
5. Add the reference
6. Return decoded integer
GPU-FOR ALGORITHM OPTIMIZATIONS

- Run on synthetic dataset of 500 million 4-byte integers
- Unoptimized Decompression = 18ms
- Reading Uncompressed dataset = 2.4ms
OPTIMIZATION 1: OPERATING IN SHARED MEMORY

• All data is contained within a block
• Load entire data block from global memory to shared memory
• Reduces from 18ms to 7ms
OPTIMIZATION 2: PROCESSING MULTIPLE BLOCKS

• Read D+1 blocks from global to shared memory
• Read granularity = 128 bytes
• Block sizes may not be multiples of 128 bytes
• Leading to unaligned read by warps
OPTIMIZATION 2: PROCESSING MULTIPLE BLOCKS

- Each thread block reads $D+1$ blocks
- Results in runtime of 2.39ms ($D=4$)

Figure 5: Decompression performance with varying number of data blocks per thread block ($D$)
OPTIMIZATION 3: PRECOMPUTING MINIBLOCK OFFSETS

- Miniblock offsets are essentially prefix sum over bit-widths array
- Precompute $D \times 4$ miniblock offsets at the start
- Results in final runtime of 2.1ms ($D = 4$)
- This is better than reading uncompressed data (2.4ms)
## GPU-DFOR

**Figure 6: GPU-DFOR Data Format**

<table>
<thead>
<tr>
<th>Block Size</th>
<th>Miniblock Count</th>
<th>Total Count</th>
<th>First Value</th>
<th>Block 1</th>
<th>Block 2</th>
<th>Block 3</th>
<th>Block 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Header

Block Starts:

[Diagram showing block structure and bit allocation]
PERFORMANCE WITH VARYING BIT-WIDTHS

Figure 7: Performance with Varying Bitwidths
EVALUATION ON DIFFERENT DATA DISTRIBUTIONS

UNIFORM | NORMAL | ZIPFIAN

Figure 8: Comparison of compression schemes on different data distributions
PERFORMANCE ON STAR SCHEMA BENCHMARK

Figure 9: Compression Waterfall for Star Schema Benchmark columns

<table>
<thead>
<tr>
<th>Planner</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planner</td>
<td>Cascading Decompression</td>
</tr>
<tr>
<td>GPU-BP</td>
<td>Only Bit-packing</td>
</tr>
<tr>
<td>nvCOMP</td>
<td>No end-to-end pipelining with query execution</td>
</tr>
<tr>
<td>OmniSci</td>
<td>Only Dictionary Encoding (DICT)</td>
</tr>
</tbody>
</table>
PERFORMANCE ON STAR SCHEMA BENCHMARK

Figure 11: Performance on Star Schema Benchmark Queries
PERFORMANCE ON STAR SCHEMA BENCHMARK

Figure 10: Average Decompression Performance across SSB columns