Tile-based Lightweight Integer Compression in GPU

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

Presented By – Vaibhav Nitnaware

MOTIVATION

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

	CPU	GPU
Memory Bandwidth	100 GBps	2 TBps
Compute	< 1 TFLOPs	19.5 TFLOPs

COMPUTE INTENSITY

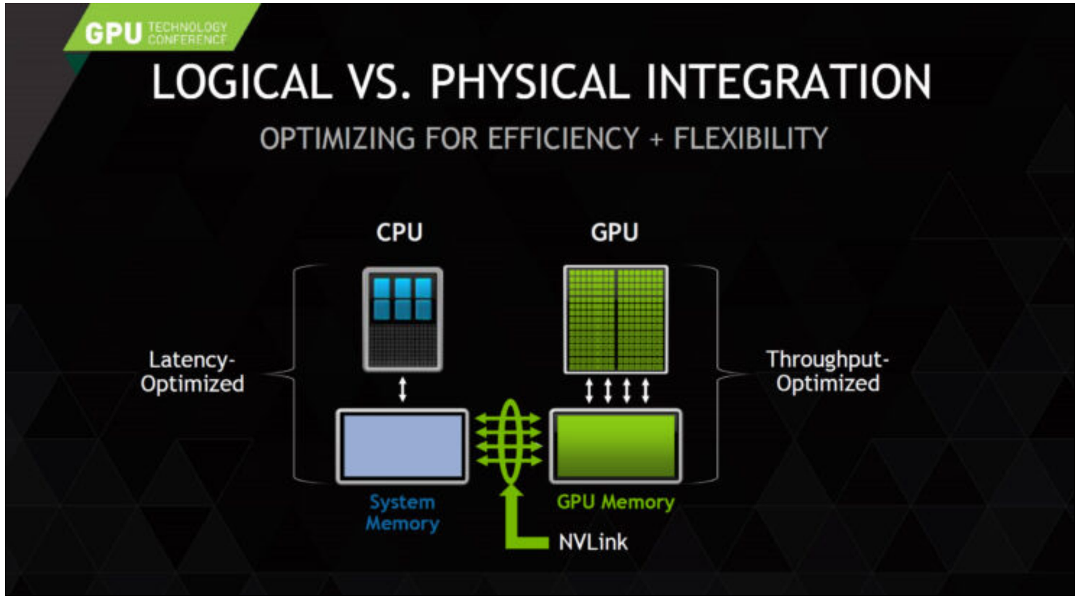
- How many operations must I do on some data to make it worth the cost of loading it?

- $Compute\ Intensity = \frac{FLOPs}{Data\ Rate}$

	CPU	GPU
Memory Bandwidth	100 GBps	2 TBps
Compute	< 1 TFLOPs	19.5 TFLOPs
Compute Intensity	~ 100	~ 80

COMPRESSION SCHEMES

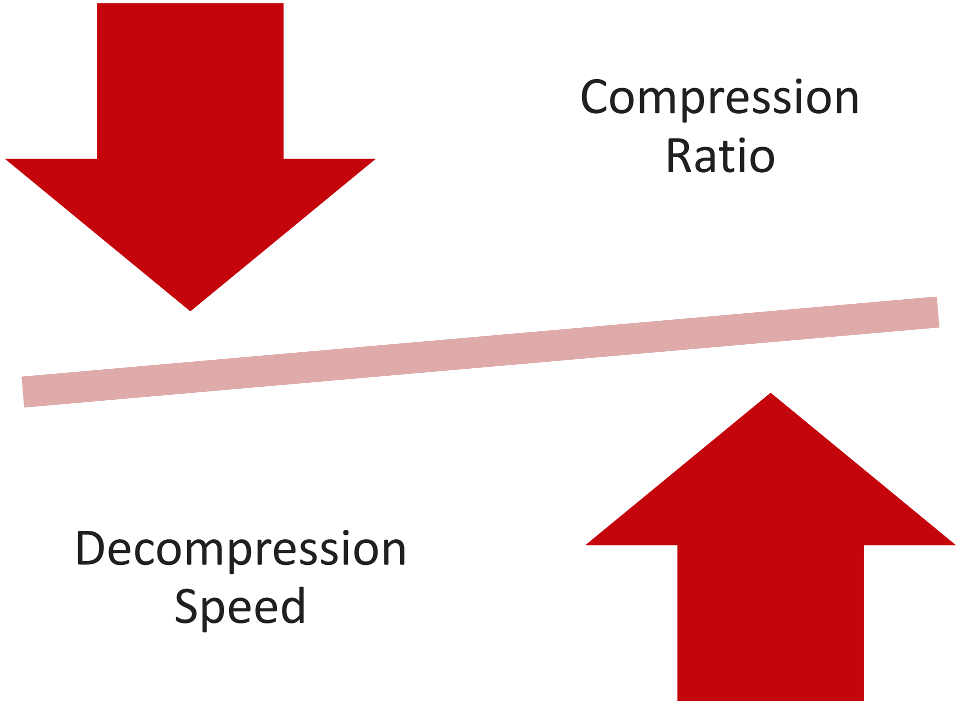
1. Fit more data in GPU memory
2. Speed up data transfer between CPU and GPU



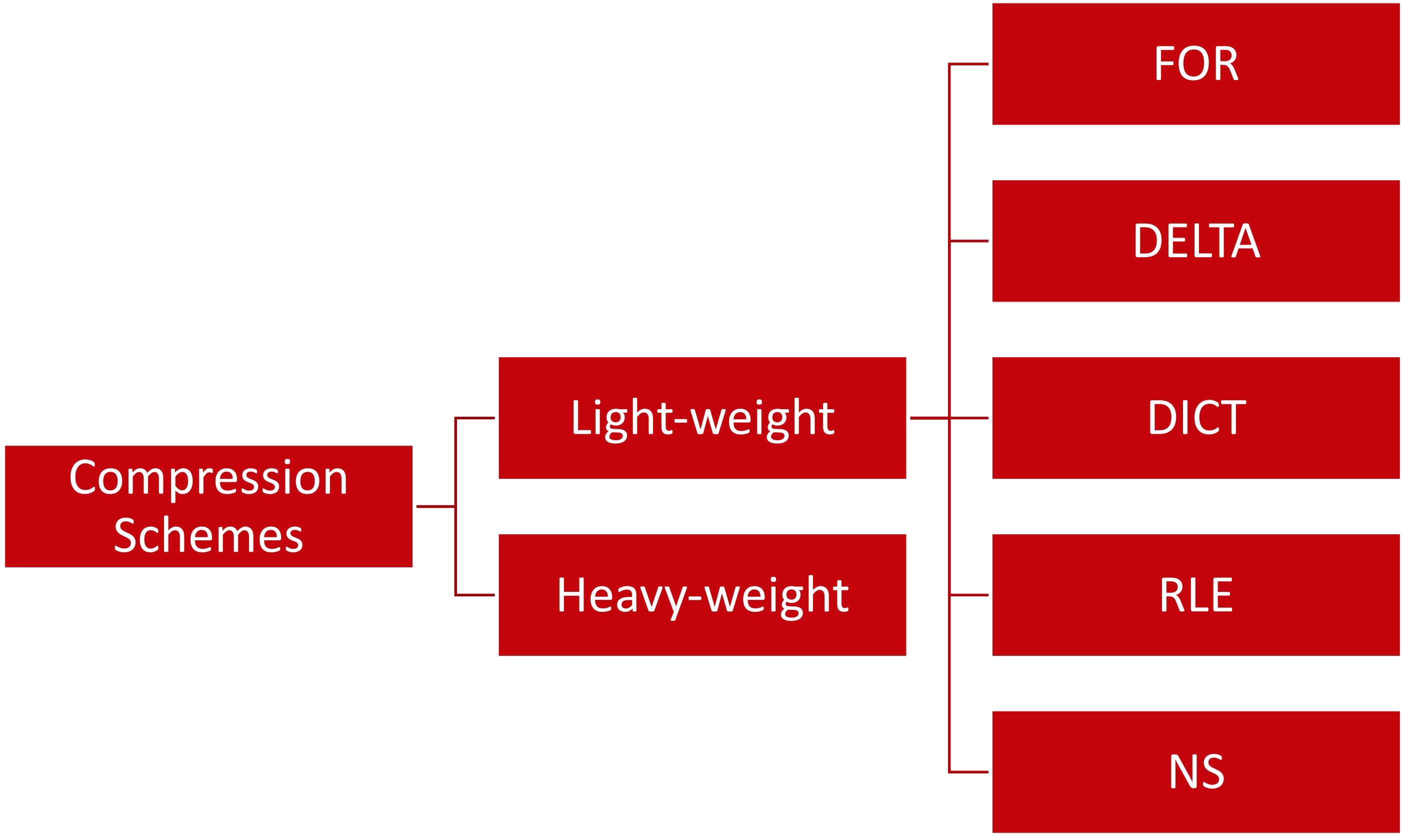
COMPRESSION SCHEMES LIMITATIONS

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 SCHEMES IN COLUMNAR STORES



Frame-of-Reference (FOR)

100	101	102	100	102	101	104	102	103	105

100	0	1	2	0	2	1	4	2	3	5

Useful when the integers have similar values

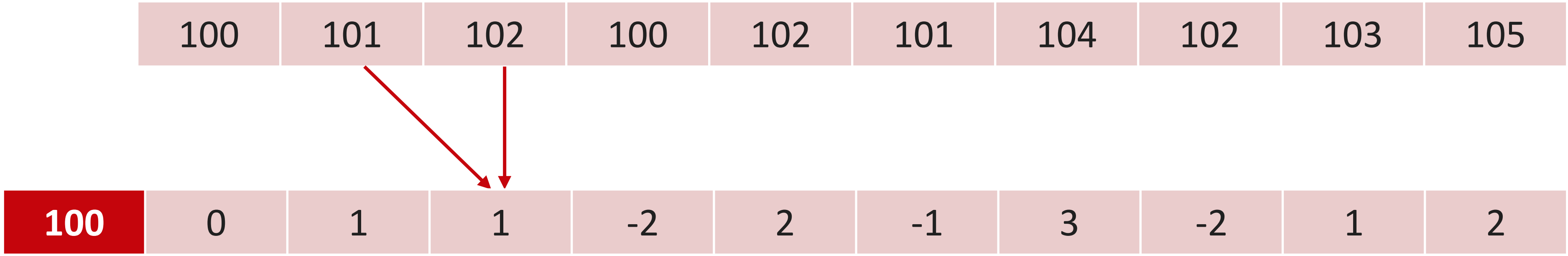
Delta Encoding (DELTA)

100	101	102	100	102	101	104	102	103	105

100	0	1	1	-2	2	-1	3	-2	1	2

Useful when the integers are sorted or semi-sorted

Delta Encoding (DELTA)



Useful when the integers are sorted or semi-sorted

Dictionary Encoding (DICT)

Column

Color
Red
Green
Blue
Green
Blue
Red

Dictionary

Color	Value
Red	0
Green	1
Blue	2

Color
0
1
2
1
2
0

Useful when the column has low cardinality

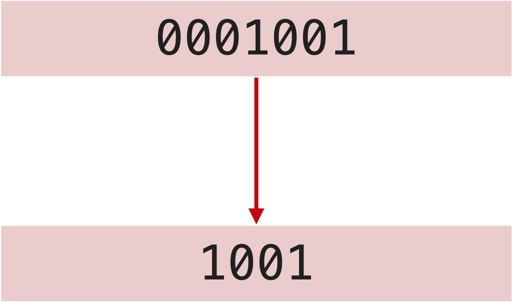
Run-length Encoding (RLE)

101	101	99	99	99	99	111	111	111	111

Values	101	99	111
Run-length	2	4	4

Useful when the data has high average run-length

Null Suppression (NS)



Useful when the integers have small values

Cascade Compression

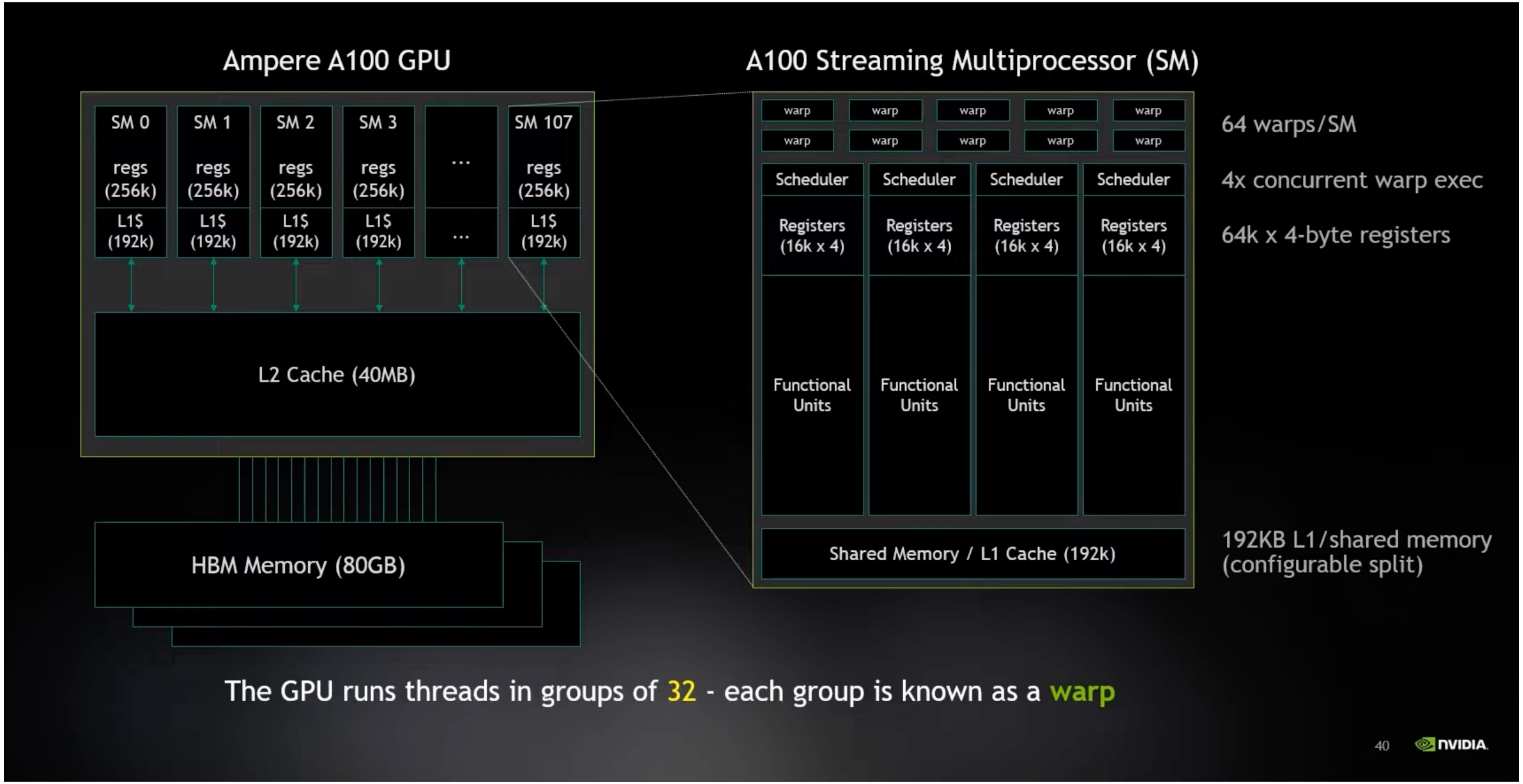
OBJECTIVES

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



Single Pass over Memory & Inline with Query Execution

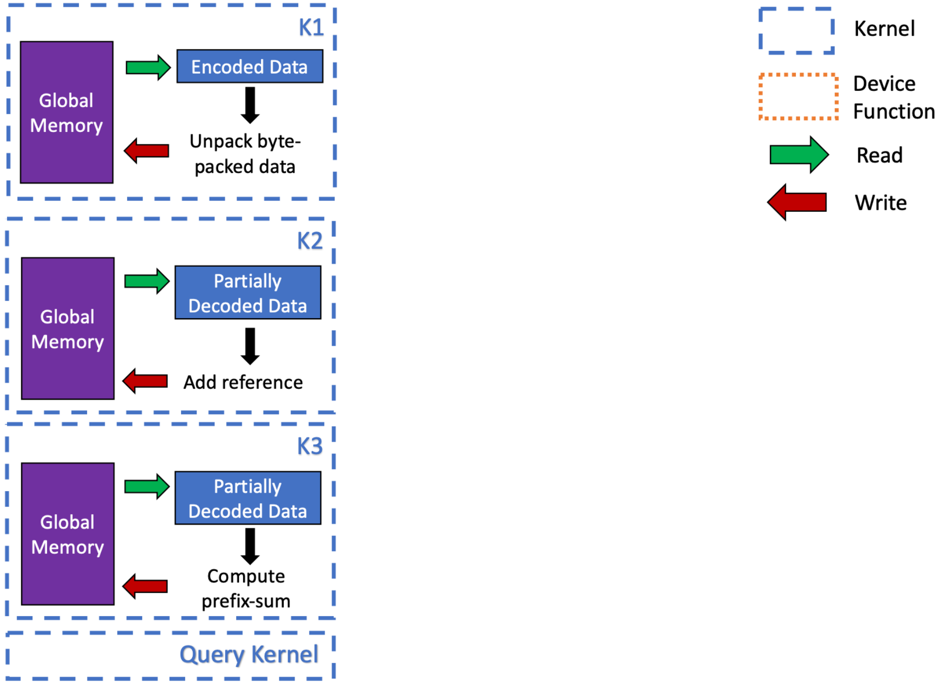


Figure 2: Decoding cascaded compression scheme (Delta+FOR+NSF) using *cascaded decompression* (left) and using *tile-based decompression* (right)

Single Pass over Memory & Inline with Query Execution

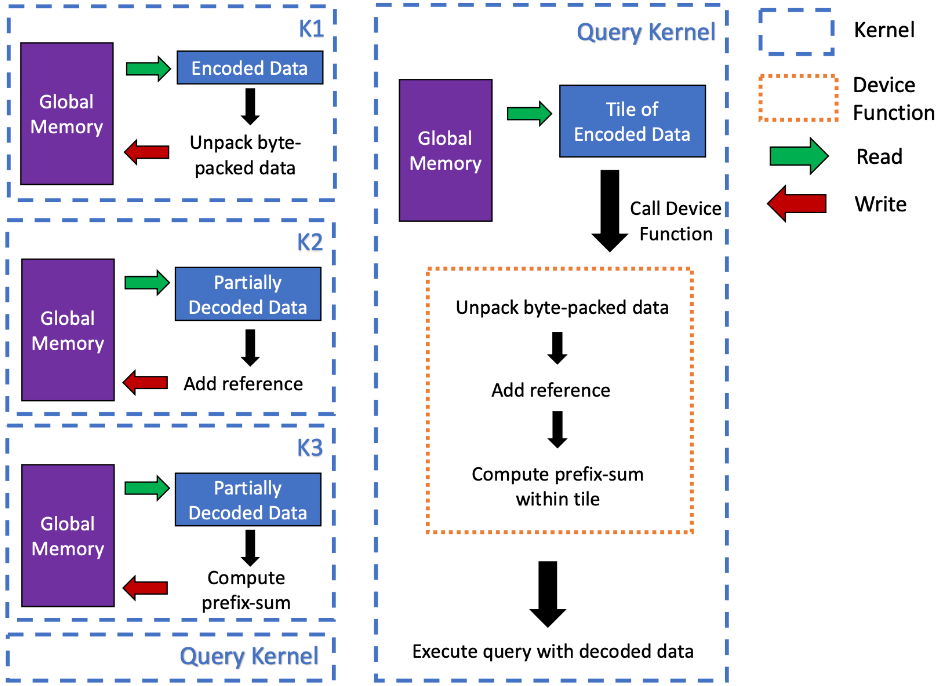


Figure 2: Decoding cascaded compression scheme (Delta+FOR+NSF) using *cascaded decompression* (left) and using *tile-based decompression* (right)

Efficient bit-packing-based compression schemes

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

GPU-FOR

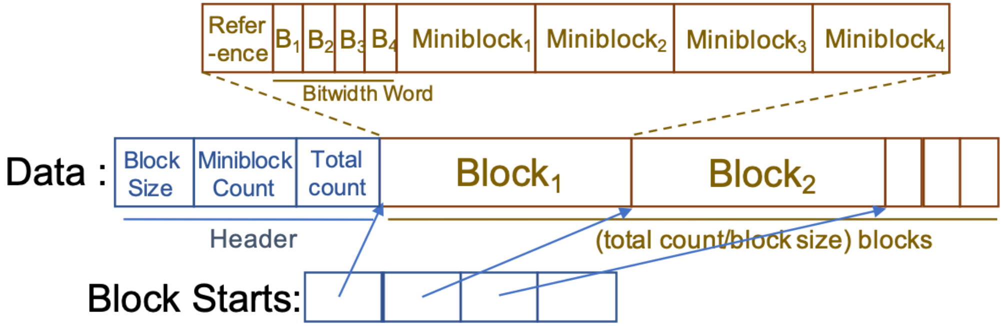


Figure 3: GPU-FOR Data Format

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

GPU-FOR

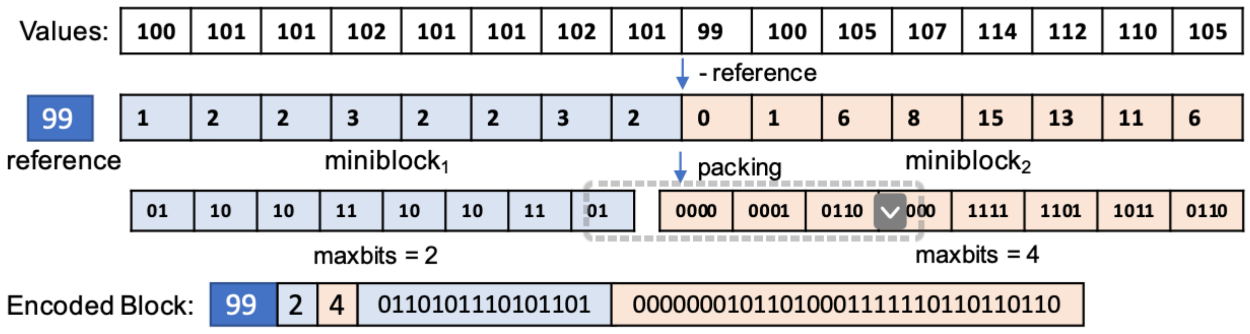


Figure 4: Example encoding with GPU-FOR

- Block Size = 16
- Miniblock Count = 2
- Total Count = 16

GPU-FOR ALGORITHM

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
int block_start = block_starts[block_id];
uint * data_block = &data[block_start];
int reference = data_block[0];
uint miniblock_id = thread_id/32;
uint index_into_miniblock = thread_id & (32 - 1);
uint bitwidth_word = data_block[1];
uint miniblock_offset = 0;
for j = 0; j < miniblock_id; j++ do
    miniblock_offset += (bitwidth_word & 255);
    bitwidth_word >>= 8;
uint bitwidth = bitwidth_word & 255;
uint start_bitindex = (bitwidth * index_into_miniblock);
uint header_offset = 2;
uint start_intindex = header_offset + miniblock_offset +
   start_bitindex/32;
uint64 element_block = data_block[start_intindex] |
   (((uint64)data_block[start_intindex + 1]) << 32);
start_bitindex = start_bitindex & (32-1);
uint element = (element_block & (((1<<bitwidth) - 1) <<
   start_bitindex)) >> start_bitindex;
item = reference + element;
```

https://github.com/anilshanbhag/gpu-compression

GPU-FOR ALGORITHM

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

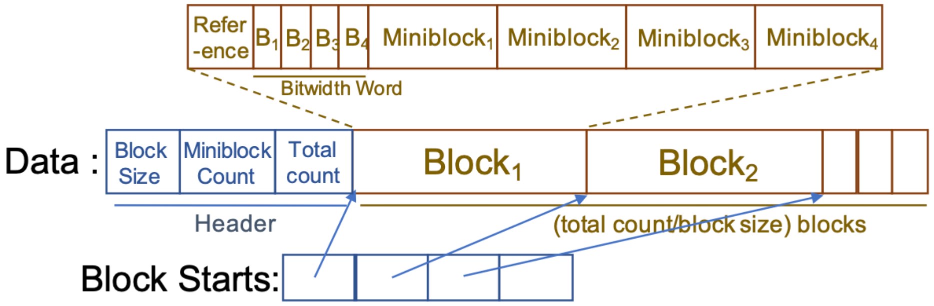


Figure 3: GPU-FOR Data Format

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)

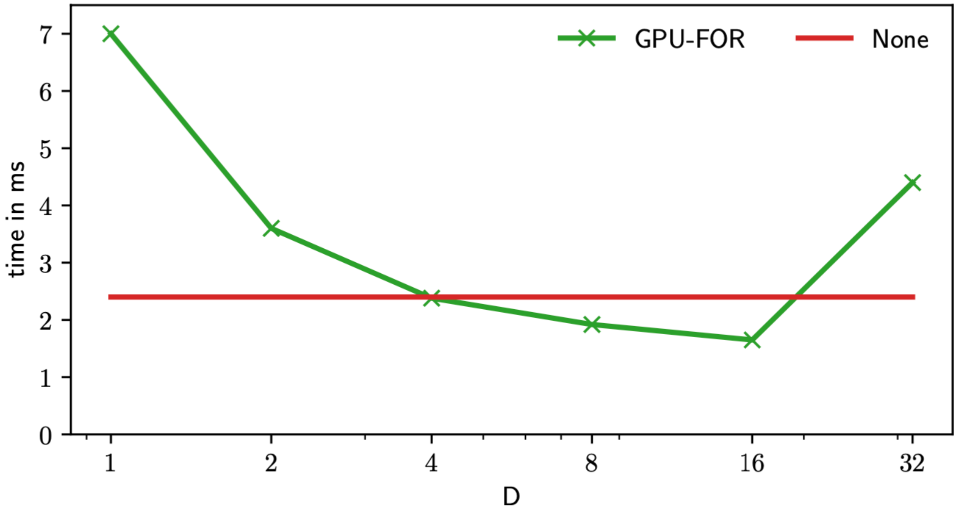


Figure 5: Decompression performance with varying number of data blocks per thread block (D)

OPTIMIZATION 3: PRECOMPUTING MINIBLOCK OFFSETS

- Miniblock offsets are essentially prefix sum over bit-widths array
- Precompute D*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

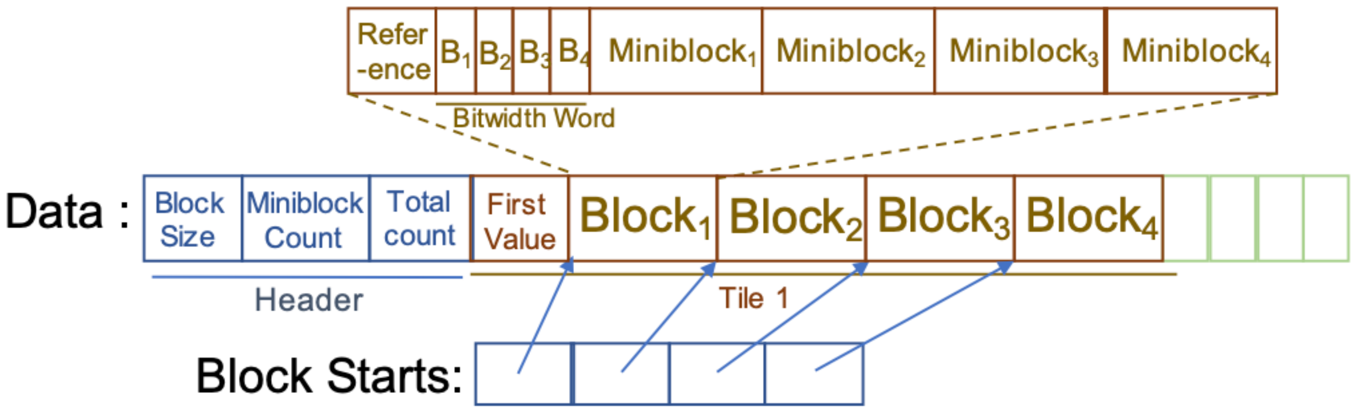


Figure 6: GPU-DFOR Data Format

PERFORMANCE WITH VARYING BIT-WIDTHS

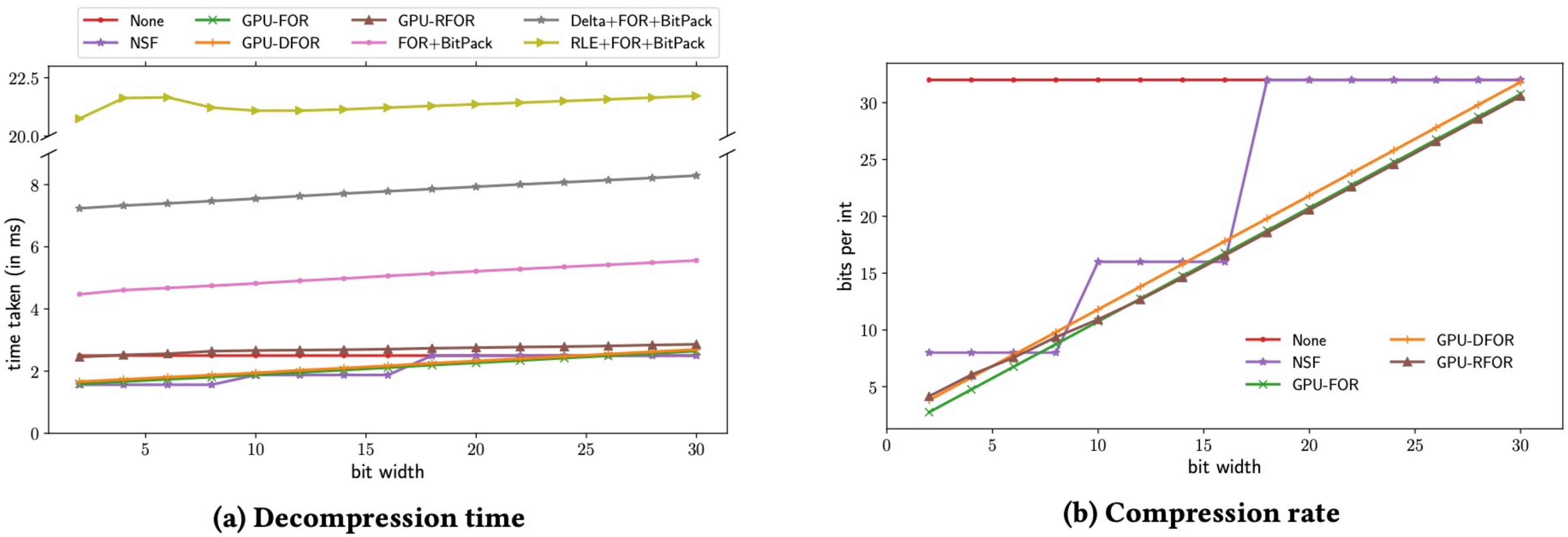


Figure 7: Performance with Varying Bitwidths

EVALUATION ON DIFFERENT DATA DISTRIBUTIONS

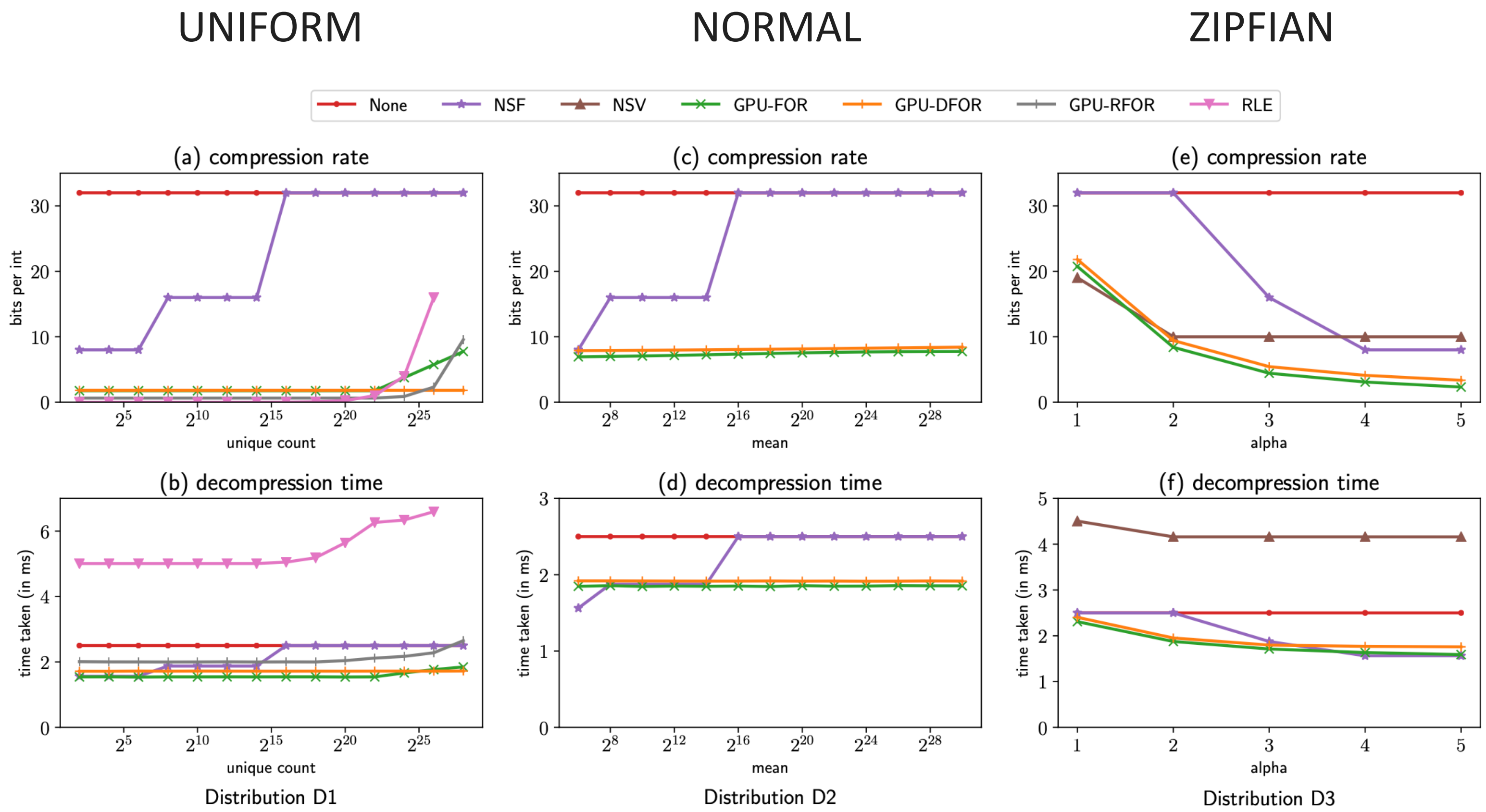


Figure 8: Comparison of compression schemes on different data distributions

PERFORMANCE ON STAR SCHEMA BENCHMARK

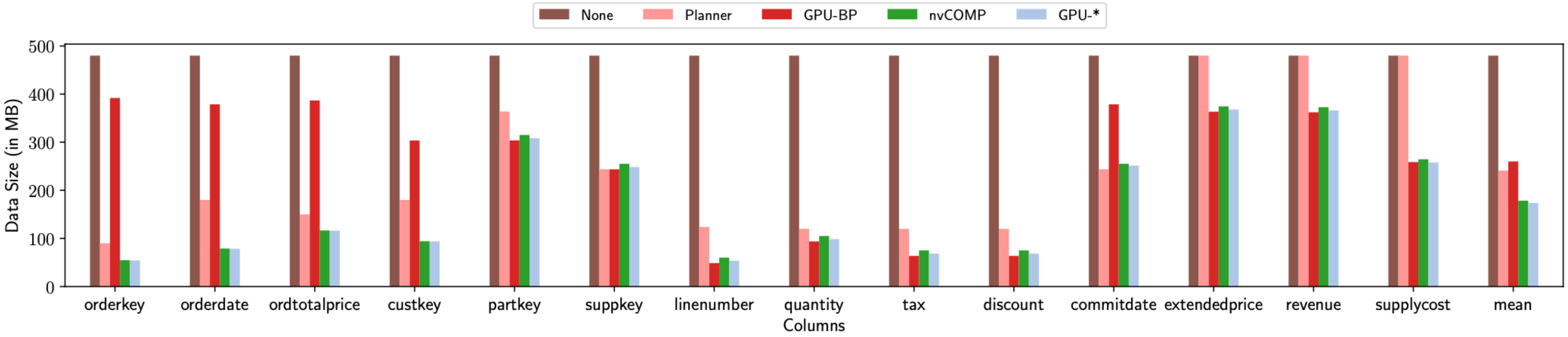


Figure 9: Compression Waterfall for Star Schema Benchmark columns

Planner	Cascading Decompression
GPU-BP	Only Bit-packing
nvCOMP	No end-to-end pipelining with query execution
OmniSci	Only Dictionary Encoding (DICT)

PERFORMANCE ON STAR SCHEMA BENCHMARK

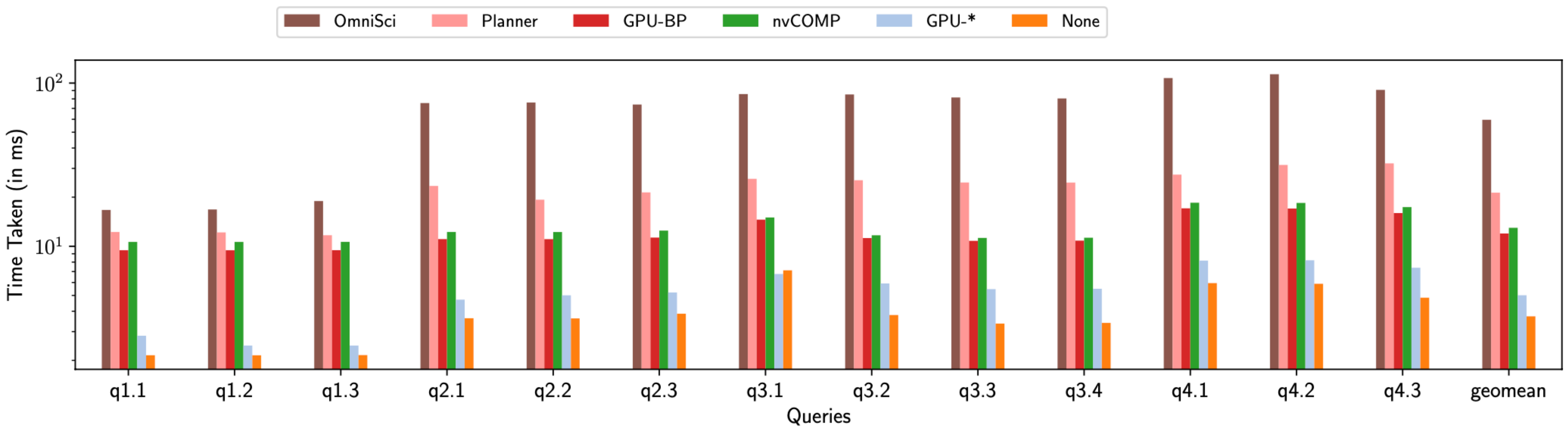


Figure 11: Performance on Star Schema Benchmark Queries

PERFORMANCE ON STAR SCHEMA BENCHMARK

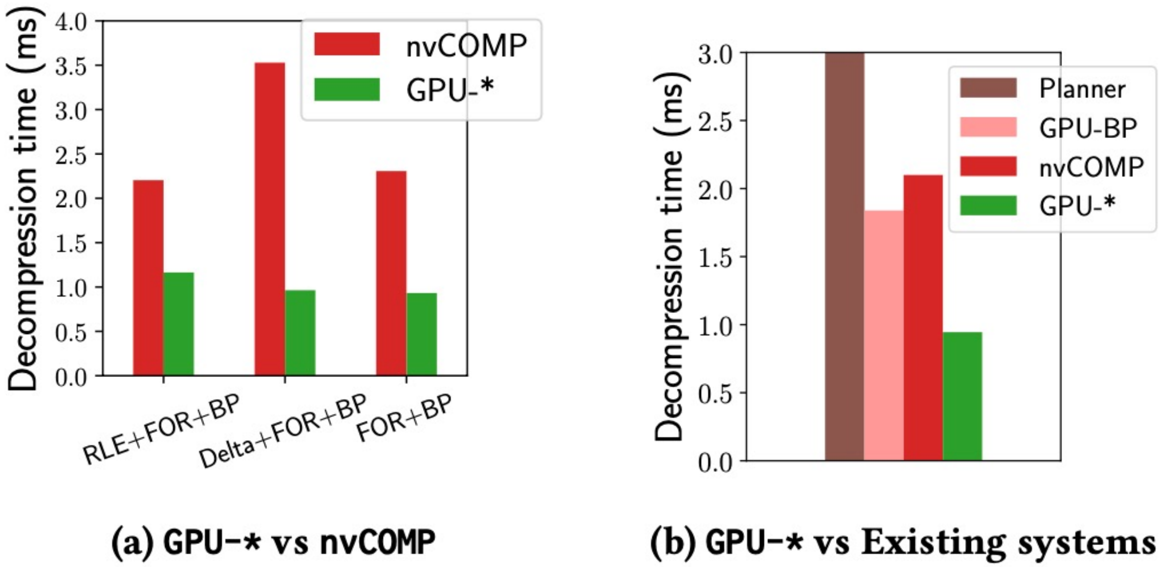


Figure 10: Average Decompression Performance across SSB columns

QUESTIONS