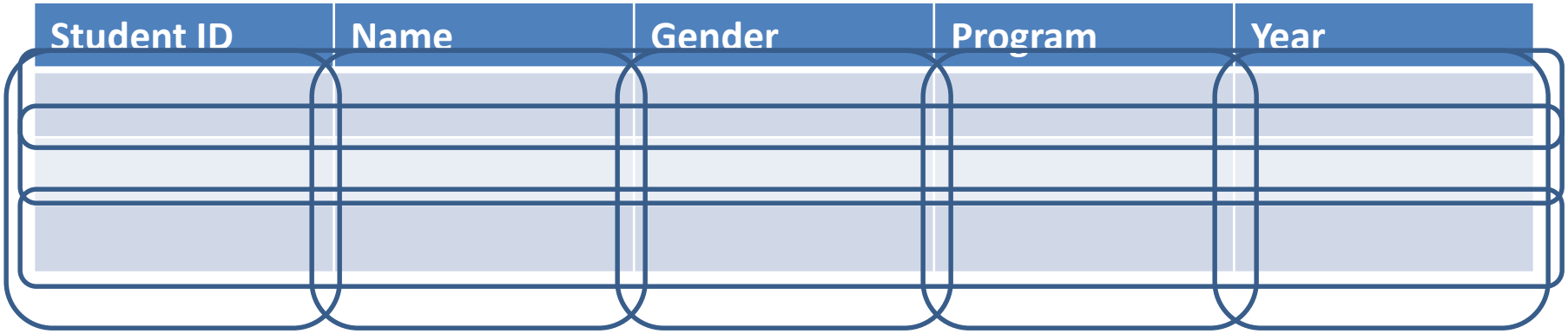
Heterogeneous Parallel Programming COMP4901D

Database Compression on the GPU

Overview

- Row stores and column stores
- Column-based database compression
- GPU-based compression for column stores

Row Stores and Column Stores

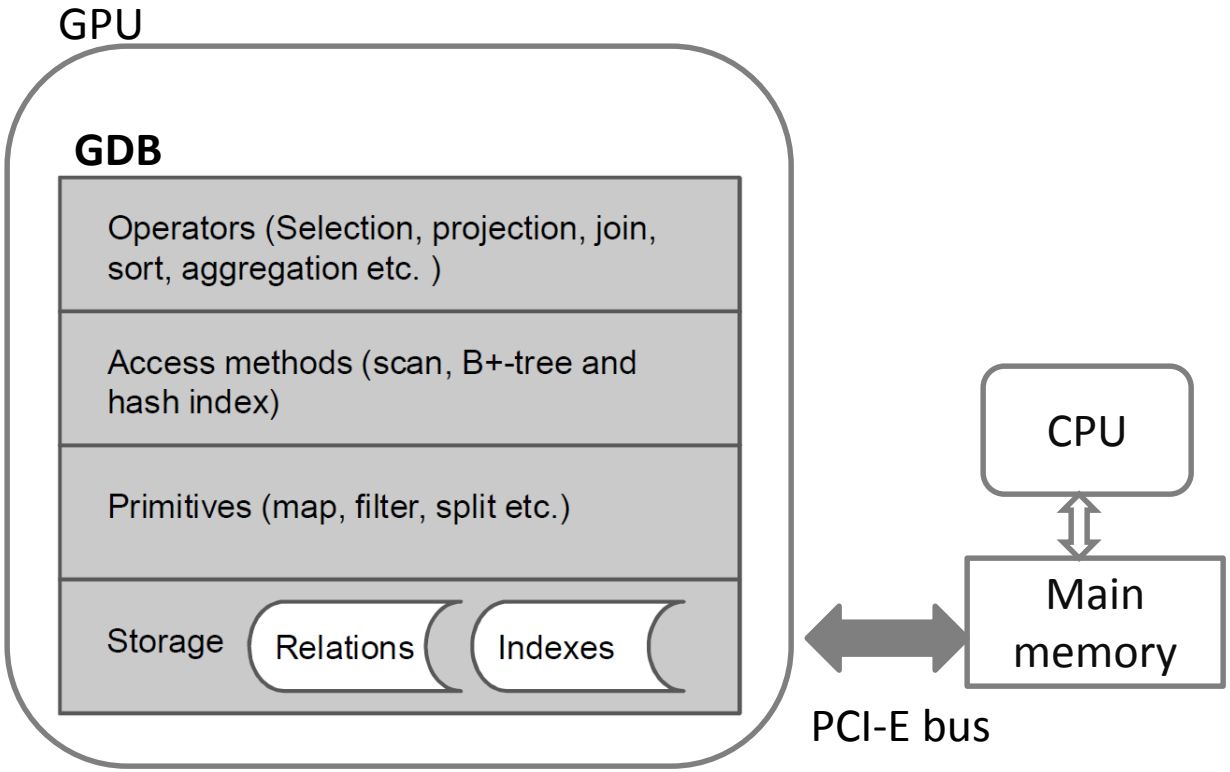


Column-Based Database Compression

- Lossless compression required
- Simple compression schemes suffice
- Fast (de)compression necessary
- Queries on compressed data desirable
- CPU and memory bandwidth intensive
- Cascaded compression not worthwhile

Database Co-Processing on the GPU

- GPU memory capacity; bandwidth; parallel processing
- PCI-e bus transfer bandwidth



Compression Schemes under Study

- Main schemes
  - Null suppression with fixed length (NS)
  - Null suppression with variable length (NSV)
  - Dictionary (DICT)
  - Bitmap (BITMAP)
  - Run-length encoding (RLE)
- Auxiliary schemes
  - Frame of reference (FOR)
  - Delta encoding (DELTA)
  - Separate components (SEP)
  - Scaling (SCALE)

Compression on the GPU

Scheme	Compression	Decompression
NS	Map	Map
NSV	Map	Prefix sum, map
DICT	Map	Map
BITMAP	Map	Map
RLE	Map, prefix sum, scatter	Prefix sum, scatter
FOR	Map	Map
DELTA	Map	Prefix sum
SEP	Map	Map
SCALE	Map	Map

RLE Compression on the GPU

Step 1, Map (find boundary of runs) --
INPUT/Uncompressed Column: A A A A B B C C C
OUTPUT/Boundary : 0 0 0 1 0 1 0 0 1

Step 2, Exclusive Prefix Sum --
INPUT/Boundary : 0 0 0 1 0 1 0 0 1
OUTPUT/Scatter Position : 0 0 0 0 1 1 2 2 2

Step 3, Scatter --
INPUT/Uncompressed Column: A A A A B B C C C
INPUT/Boundary : 0 0 0 1 0 1 0 0 1
INPUT/Scatter Position : 0 0 0 0 1 1 2 2 2
OUTPUT/Value Array : A B C
OUTPUT/Boundary Position : 4 6 9

Step 4, Map (calculate run lengths) --
INPUT/Boundary Position : 4 6 9
OUTPUT/Run Length : 4 2 3

RLE Decompression on the GPU

Step 1, Inclusive Prefix Sum --
INPUT/Run Length : 4 2 3
OUTPUT/Boundary Position : 4 6 9

Step 2, Scatter --
INPUT/Boundary Position : 4 6 (9)
OUTPUT/Boundary : 0 0 0 0 1 0 1 0 0

Step 3, Inclusive Prefix Sum --
INPUT/Boundary : 0 0 0 0 1 0 1 0 0
OUTPUT/Scatter Position : 0 0 0 0 1 1 2 2 2

Step 4, Scatter --
INPUT/Value Array : A B C
INPUT/Scatter Position : 0 0 0 0 1 1 2 2 2
OUTPUT/Uncompressed Column: A A A A B B C C C

Cascaded Compression

- Apply compression multiple times
  - Advantage: improved compression ratio
  - Disadvantages:
    - Longer compression/decompression time
    - Limitations in feasibility and effectiveness
    - Large search space for finding a good compression plan
    - Reduced direct querying on compressed data

Compression Planner

- Heuristic rules to reduce search space
  - Data properties required for an individual scheme
  - Data properties changed after applying a scheme
- Estimate execution time of individual schemes
  - Compression code
  - GPU warp scheduling and memory latency

Example Compression Plans

lineitem			
Column	**Lowest-ratio Plan**	**Ratio**	**# of candidates**
l_extendedprice	None	100%	0
l_discount	SCALE, NS	25%	42
l_shipdate	FOR, NS	50%	218
l_partkey	RLE, [[DELTA, NS] | NS]	1.67%	17
l_quantity	NS	25%	6
part			
Column	**Lowest-ratio Plan**	**Ratio**	**# of candidates**
p_partkey	DELTA, NS	25%	8
p_type	DICT, NS	9.9%	6

Query Processing on Compressed Data

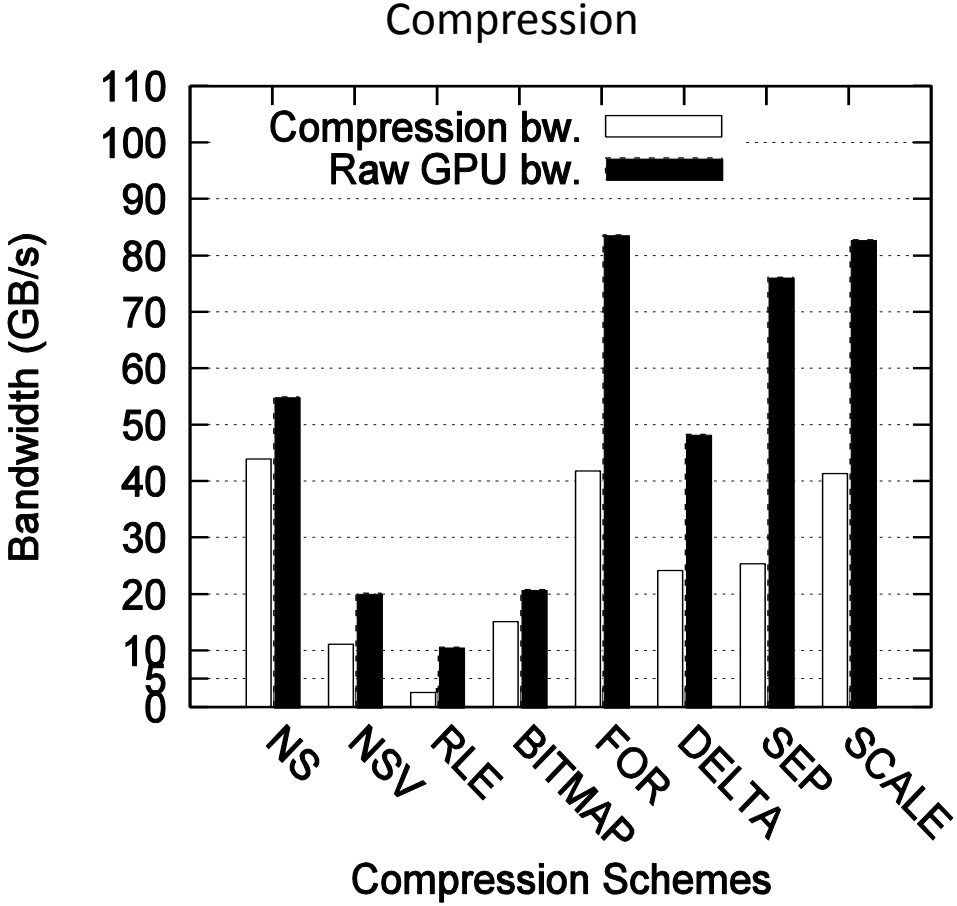
- Given a query operator, and a compression plan “S1, S2, …., SN”
  - Check the compression plan in reverse order
    - From right to left, if Si is the first compression scheme that we need to decompress with, then we decompress all schemes on the right of Si.
  - Apply the query operator on the fully or partially decompressed data

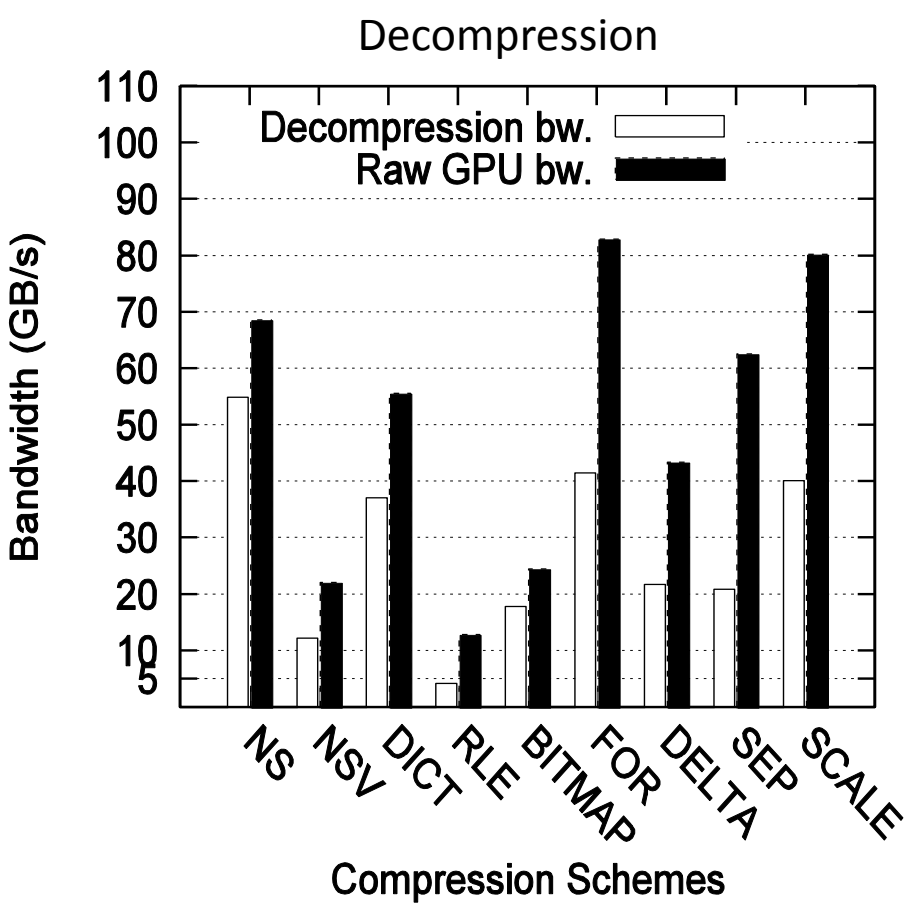
Experimental Setup

- Machine
  - Fedora 11 Linux PC
  - CPU: Intel Core2 Quad CPU 2.4 GHz
  - GPU: NVIDIA GTX 295 graphics card
  - PCI-E bus: 3.2 GB/sec
- Main memory databases: GDB/MonetDB 5
- TPC-H scaling factor 10

Compression and Decompression bandwidth

Bandwidth = Uncompressed size / processing time





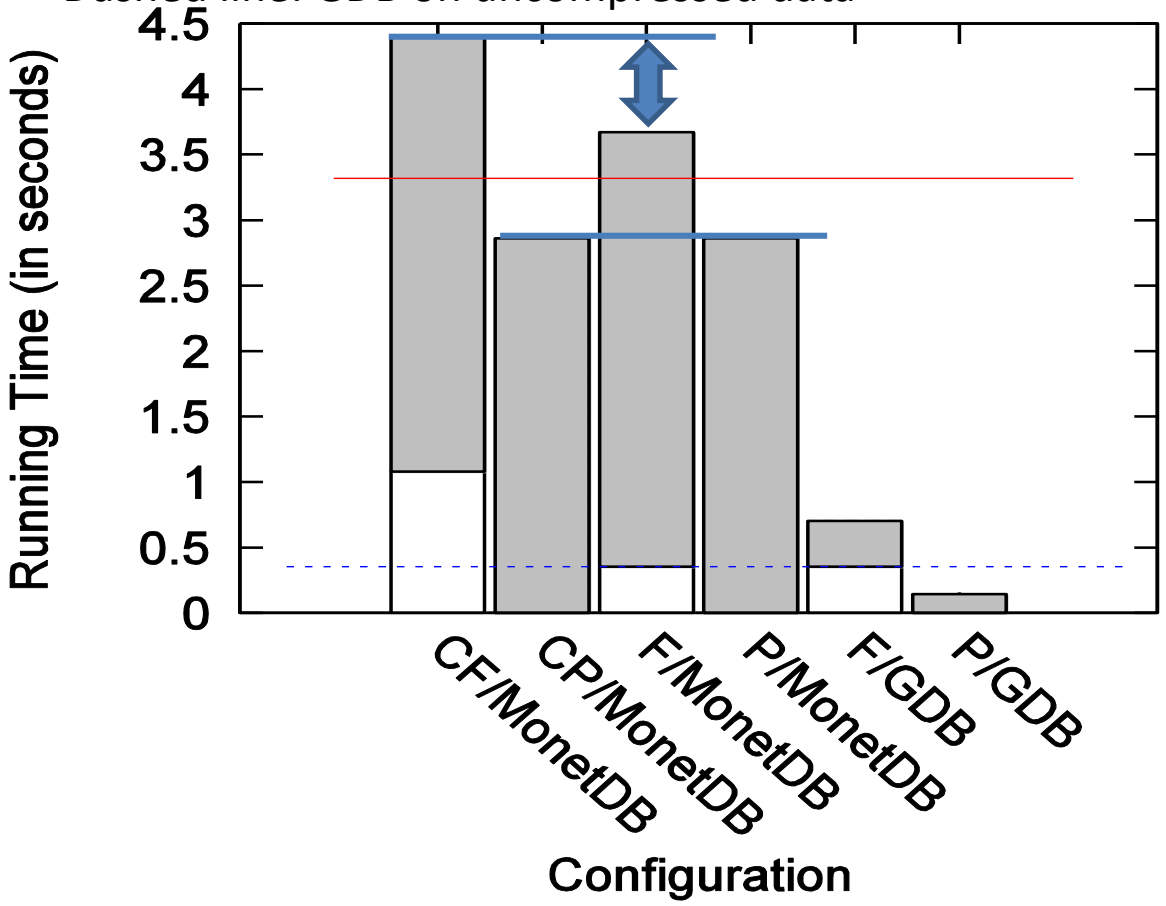
Effectiveness of Compression Planner

l_partkey in itemline table		
Plan	Comp. Ratio	By our planner?
A: RLE	6.68%	Yes
B: NS	100%	No
C: DELTA, NS	25%	Yes
D: RLE, [ε | NS]	4.16%	Yes
E: RLE, [[DELTA, NS] | NS]	1.67%	Yes
F: DELTA, RLE, [NS | NS]	3.33%	No

TPC-H Q6 (Simple Selection): Offloading Decompression to the GPU

Solid line: MonetDB on uncompressed data

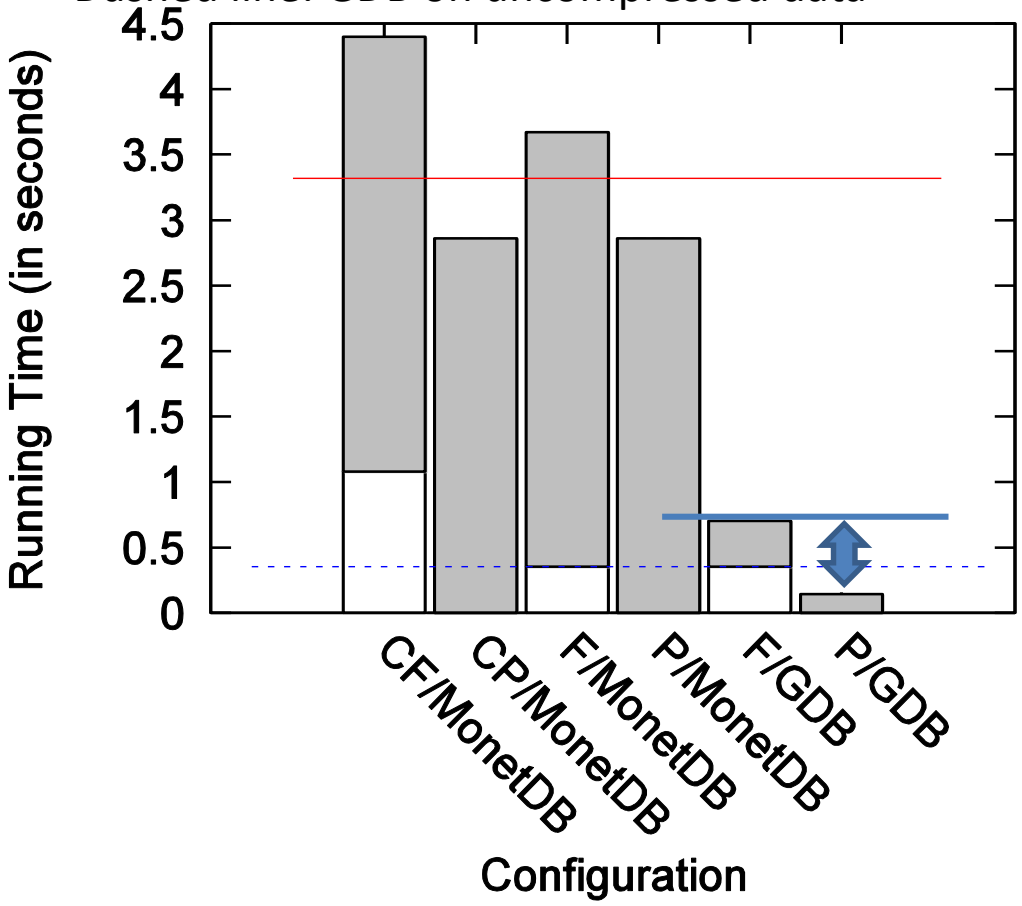
Dashed line: GDB on uncompressed data



TPC-H Q6 (Simple Selection): Partial vs. Full Decompression

Solid line: MonetDB on uncompressed data

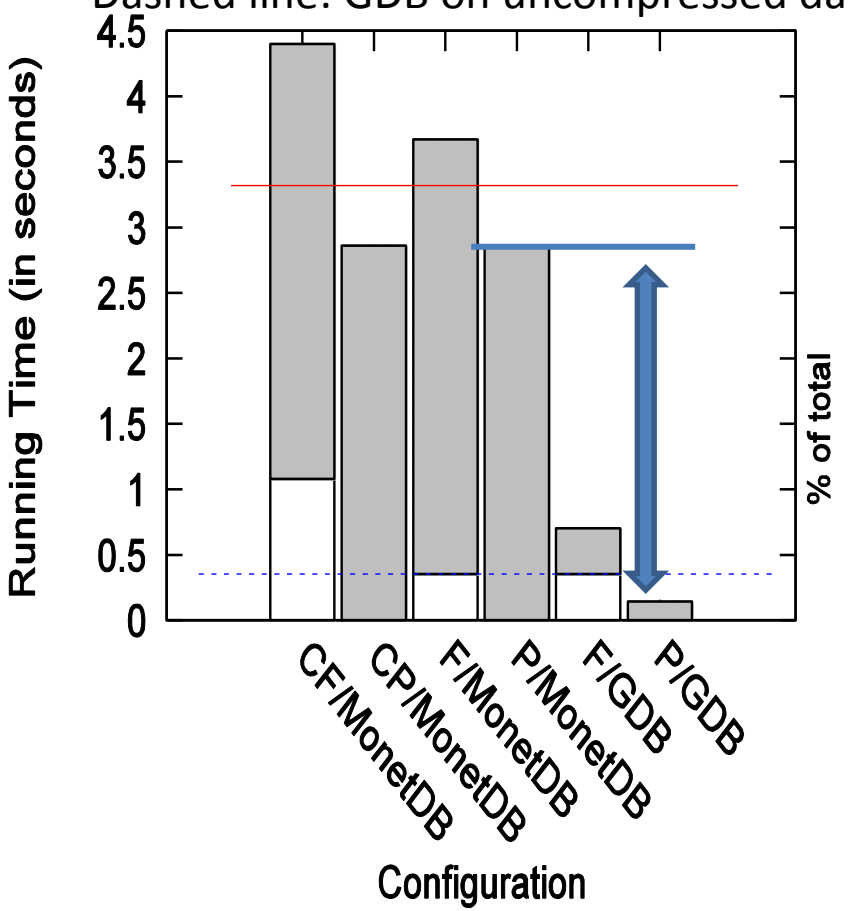
Dashed line: GDB on uncompressed data

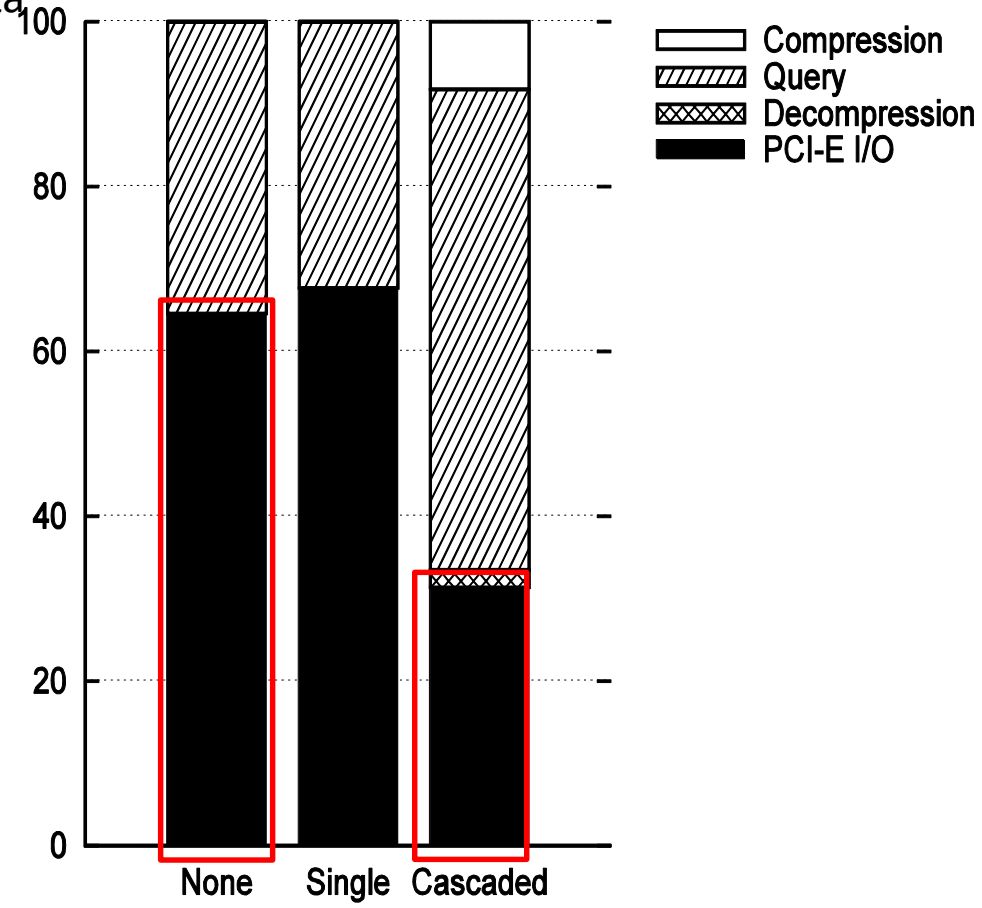


TPC-H Q6 (Simple Selection): MonetDB vs. GDB

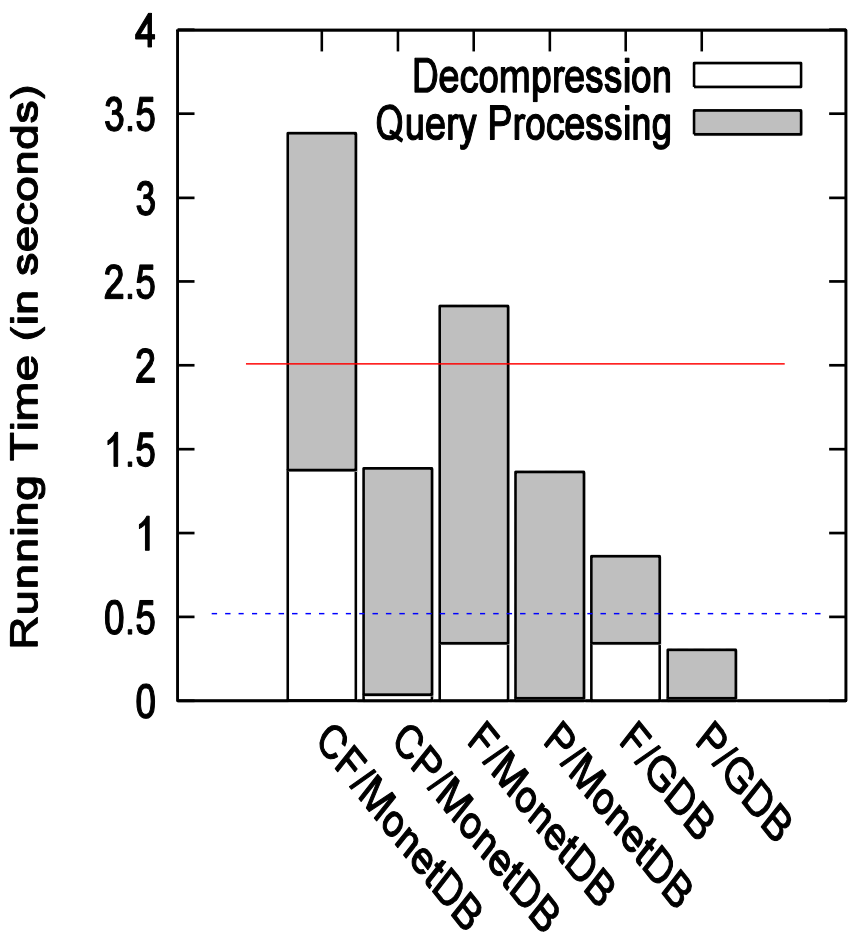
Solid line: MonetDB on uncompressed data

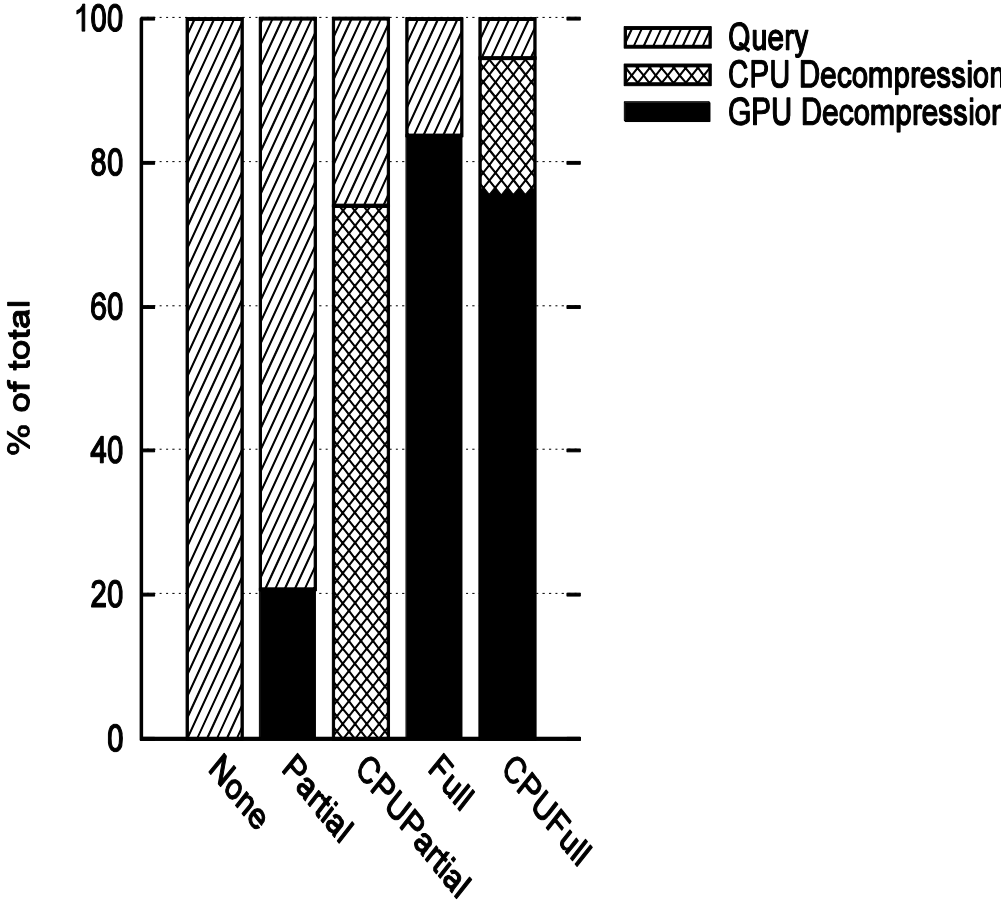
Dashed line: GDB on uncompressed data





TPC-H Q14 (A Join Query)





Comparison with Gzip

			Gzip		
Column	**Type**	**Sorted**	**Ratio**	**Compression Time**	**Decompression Time**
li_partkey	int(4 bytes)	Yes	1.87%	3.67 s	1.46 s
li_quantity	int(4 bytes)	No	25.76%	36.01 s	2.24 s
li_discount	float(4 bytes)	No	17.9%	28.58 s	2.02 s
li_shipmode	string(8 bytes)	No	14.41%	18.82 s	2.36 s
li_returnflag	char(1 byte)	No	18.77%	36.88 s	2.24 s

	GPU Compression			
Column	**Plan**	**Ratio**	**Compression Time**	**Decompression Time**
li_partkey	RLE, [[DELTA, NS] | NS]	1.67%	87.37 ms	53.96 ms
li_quantity	NS	25%	5.09 ms	4.07 ms
li_discount	SCALE, NS	25%	10.70 ms	9.82 ms
li_shipmode	DICT	50%	-	12.07 ms
li_returnflag	BITMAP	37.5%	14.78 ms	12.54 ms

Summary

- GPUs accelerate database compression, and enable cascaded compression.
  - Nine GPU-Accelerated compression schemes
  - Compression planner for finding good plans
- GPU-based database compression improves the overall performance of query processing.