Lightweight Compression Methods Achieving 120GBps and More

Piotr Przymus

Laboratoire d'Informatique Fondamentale de Marseille
Aix-Marseille University, France

GPU Technology Conference
Silicon Valley May 2017

FeatherGPU

K. Kaczmarski and P. Przymus, *Fixed Length Lightweight Compression for GPU Revised*, Journal of Parallel and Distributed Computing, 2017.

- A lightweight compression library for GPU.
- **github.com/mis-wut/feathergpu**
- **MIT**-licenesed.
- This project was partly funded by National Science Centre, decision DEC-2012/07/D/ST6/02483.

Team


- Krzysztof Kaczmarski
  - Warsaw University of Technology, Poland
- Piotr Przymus
  - Aix-Marseille University, France
  - Nicolaus Copernicus University in Toruń, Poland.

Lightweight compression on GPU – motivation

Lightweight compression algorithms favours compression and decompression speed over compression ratio.

- Improved data transfer:
  - Disk ↔ RAM ↔ GPU.
  - GPU ↔ GPU:
    - exchange of already compressed data,
    - **compress → transfer → decompress.**
- Lower memory footprint:
  - Less disk space used.
  - Less RAM used.
  - Less GPU memory used.
- Improved internal memory access:
  - In some cases improved internal GPU memory access.

Lightweight compression on GPU – motivation

Lightweight compression algorithms favours compression and decompression speed over compression ratio.

- Improved data transfer:
  - Disk $\leftrightarrow$ RAM $\leftrightarrow$ GPU.
  - GPU $\leftrightarrow$ GPU:
    - exchange of already compressed data,
    - **compress $\rightarrow$ transfer $\rightarrow$ decompress.**
- Lower memory footprint:
  - Less disk space used.
  - Less RAM used.
  - Less GPU memory used.
- Improved internal memory access:
  - In some cases improved internal GPU memory access.

Lightweight compression on GPU – motivation

Lightweight compression algorithms favours compression and decompression speed over compression ratio.

- Improved data transfer:
  - Disk $\leftrightarrow$ RAM $\leftrightarrow$ GPU.
  - GPU $\leftrightarrow$ GPU:
    - exchange of already compressed data,
    - **compress $\rightarrow$ transfer $\rightarrow$ decompress.**
- Lower memory footprint:
  - Less disk space used.
  - Less RAM used.
  - Less GPU memory used.
- Improved internal memory access:
  - In some cases improved internal GPU memory access.

Lightweight compression on GPU – motivation

Lightweight compression algorithms favours compression and decompression speed over compression ratio.

- Improved data transfer:
  - Disk $\leftrightarrow$ RAM $\leftrightarrow$ GPU.
  - GPU $\leftrightarrow$ GPU:
    - exchange of already compressed data,
    - **compress $\rightarrow$ transfer $\rightarrow$ decompress.**
- Lower memory footprint:
  - Less disk space used.
  - Less RAM used.
  - Less GPU memory used.
- Improved internal memory access:
  - In some cases improved internal GPU memory access.

Fixed length compression

Fixed length (**FL**) – is a simple well known compression scheme where fixed number of bits is suppressed. Suppressed bits should be equal to 0.

0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	1	0	0	0

Figure: Original data, only 4 bits are used in each byte.

Fixed length compression

Fixed length (**FL**) – is a simple well known compression scheme where fixed number of bits is suppressed. Suppressed bits should be equal to 0.

0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	1	0	0	0

Figure: Original data, only 4 bits are used in each byte.

Fixed length compression

Fixed length (**FL**) – is a simple well known compression scheme where fixed number of bits is suppressed. Suppressed bits should be equal to 0.

0 0 0 0	0 0 0 1	0 0 0 0	0 0 1 0	0 0 0 0	0 0 1 1	0 0 0 0	0 1 0 0
0 0 0 0	0 1 0 1	0 0 0 0	0 1 1 0	0 0 0 0	0 1 1 1	0 0 0 0	1 0 0 0

Figure: Original data, only 4 bits are used in each byte.

0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0

Figure: Compressed data (each byte encodes two words of length 4 bits.)

Fixed length compression

Fixed length (**FL**) – is a simple well known compression scheme where fixed number of bits is suppressed. Suppressed bits should be equal to 0.

0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	1	0	0	0

Figure: Original data, only 4 bits are used in each byte.

0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0	0	1	0	1	0	1	1	0	0	1	1	1	1	0	0	0

Figure: Compressed data (each byte encodes two words of length 4 bits.)

$$\text{compression ratio (CR)} = \frac{\textit{Uncompressed size}}{\textit{Compressed size}} = 2,$$

Fixed length compression

Fixed length (FL) compression:

- easy to implement,
- easy to achieve high data throughput.

Many applications:

- Database compression: Columns, Indexes,
- Timeseries compression,
- Graph compression,
- etc.

Many variants:

- Patched FL, Adaptive FL, DELTA-*

Fixed length compression on GPU

Performance over flexibility (**Fang et al. 2010**)

- High performance but highly simplified version of algorithm.
  - Words are mapped to full bytes e.g. 4 bits word will be mapped to 1 byte.
- Uses map primitive.
- Coalesced reads and writes: **YES**.
- Direct memory access: **YES**.

Flexibility over performance
(**Nvbio** and **Kaczmarski, Przymus 2012-2017**)

- No simplifications at the cost of lower performance.
- Supports all possible bit encodings.
- Uses allgather or gather primitive.
- Coalesced reads and writes: **NO**.
- Direct memory access: **YES**.

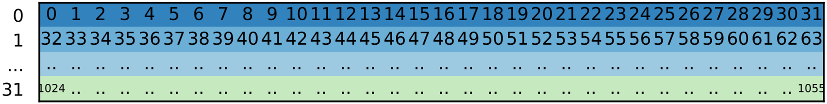


Figure: Read pattern: GPU version of FL algorithm

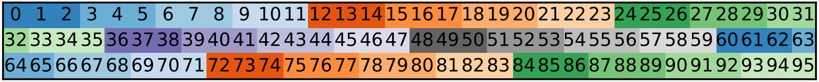
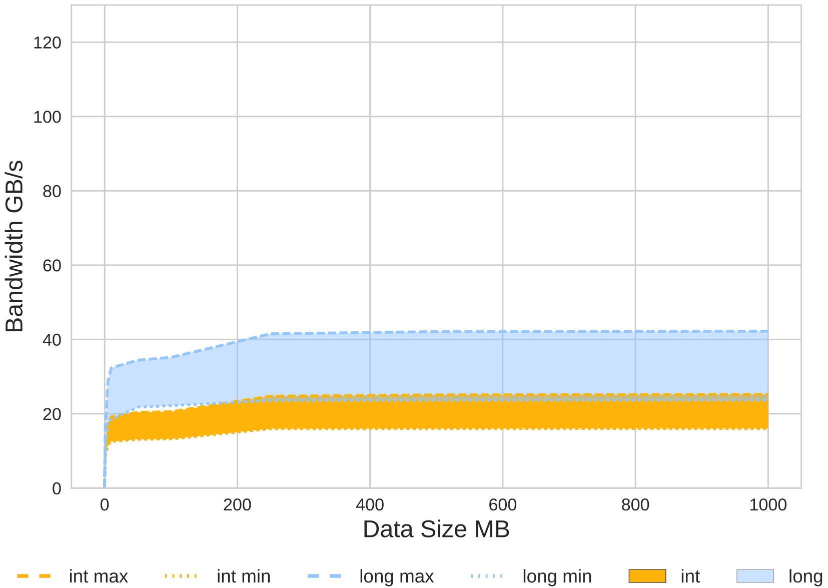
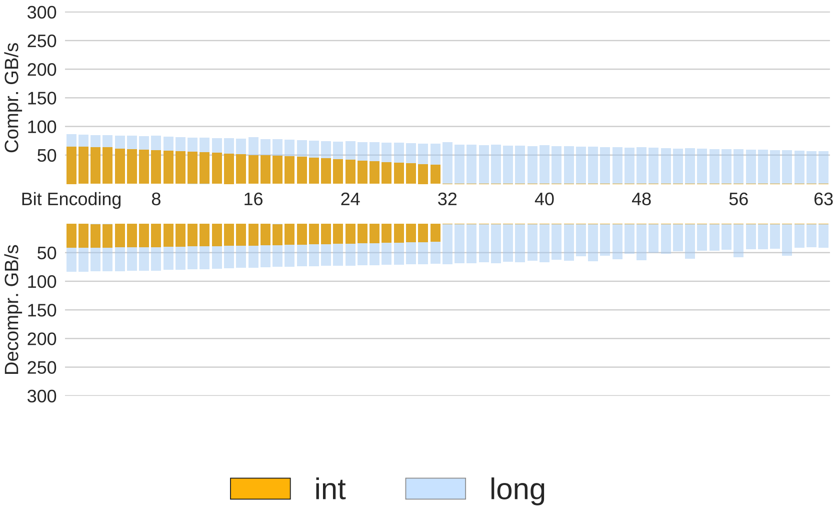


Figure: Write pattern: GPU version of FL algorithm

Fixed length on GPU (C+D, GTX Titan Black)



Fixed length on GPU (1 GB of data, GTX Titan Black)



Can we do better?

Aligned Fixed Length (**AFL**) algorithm.

- The **FL** algorithm is optimized for **CPU** memory access scheme.
- We can do better with **GPU** friendly memory organisation scheme.

Features

- No simplifications, high performance on **GPU**.
- Still works quite well on **CPU**, but loses some cache hits benefits.
- Supports all possible bit encodings.
- Uses allgather or gather primitive.
- Coalesced reads and writes: **YES**.
- Direct memory access: **YES**.

Aligned FL on GPU

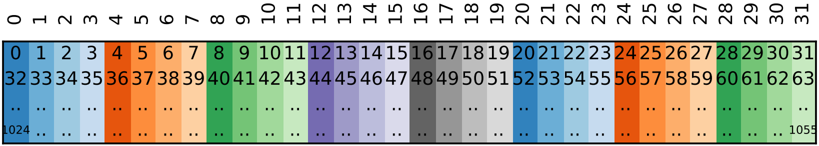


Figure: Read pattern: GPU version of Aligned FL algorithm

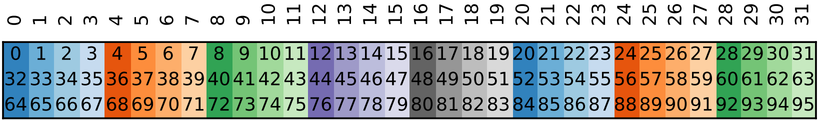
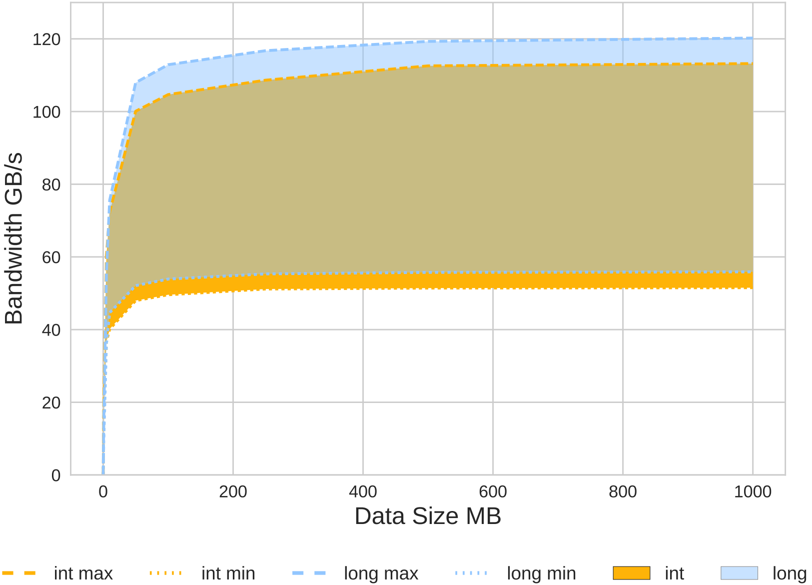
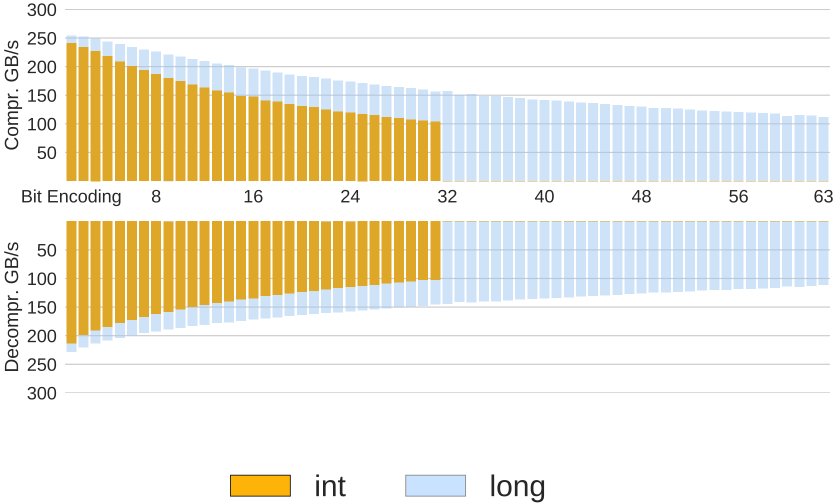


Figure: Write pattern: GPU version of Aligned FL algorithm

Aligned FL on GPU (C+D, GTX Titan Black)



Aligned FL on GPU (1 GB of data, GTX Titan Black)



Direct memory access

Direct memory access (Random access):

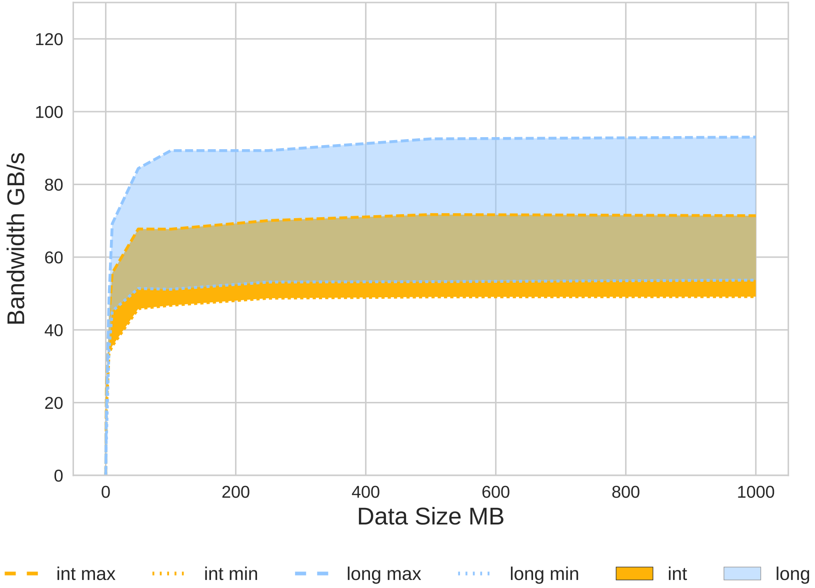
- single value access with decompression on-the-fly,
- no need for explicite decompression step,
- simple integration with existing algorithms.

Example applications of direct access FL:

- Bioinformatics – **NVBIO**,
- Databases – **Fang et al. 2010**,
- Timeseries, Graph – **Kaczmarski and Przymus 2012-2017**.

Aligned FL supports direct access!

Direct access AFL on GPU (C+D, GTX Titan Black)



Initial transformations

A initial data preparation is often used in order to minimize bits usage.

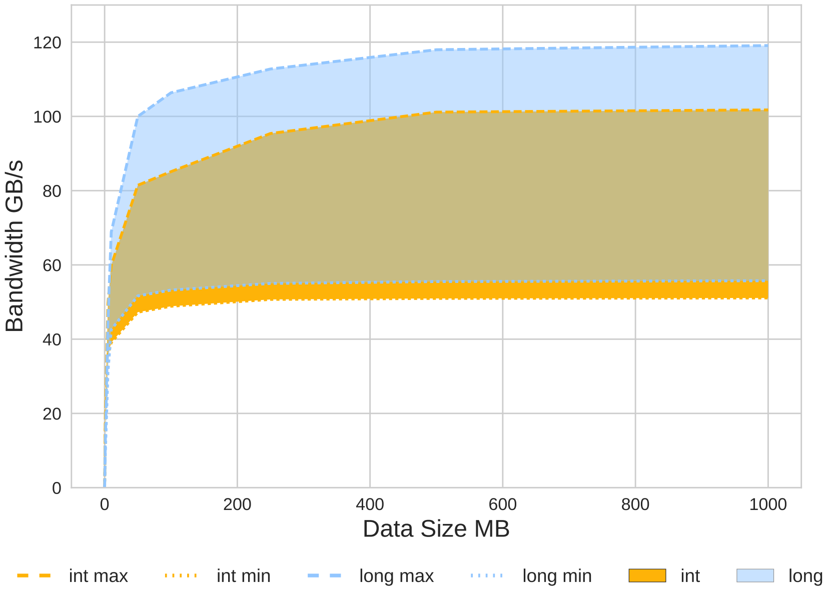
Frame of reference (FOR):

- Subtracts a given value from all values $\{v_0, v_1, \ldots, v_n\} \rightarrow \{v_0 - f, v_1 - f, \ldots, v_n - f\}$.
- Straightforward and simple integration with **AFL**.

DELTA:

- Transforms data into the differences between successive values $\{v_0, v_1, \ldots, v_n\} \rightarrow \{v_1 - v_0, v_2 - v_1, \ldots, v_n - v_{n-1}\}$.
- Integration with **AFL** algorithm is not that simple.
  - Coalesced memory reads and writes require different scheme of data reads.
  - Threads operate on data subsequence $\{a_k, a_{k+32}, a_{k+64}, \ldots\}$.
  - **Solution**: Interthread communication within warp using **shuffle** instruction available starting from the **Kepler**.

DELTA Aligned FL on GPU (C+D, GTX Titan Black)



Dealing with the data outliers

FL and **AFL** algorithms are prone to data outliers.

Example:

- Input data: $\{1, 2, 3, 2, 2, 3, 1, 1, 3, 2, 3, 1, 1\}$
  - 2 bits **FL** (or **AFL**) encoding may be used.
- Input data: $\{1, 2, 3, 2, 2, 3, 1, 1, 64, 2, 3, 1, 1\}$
  - 6 bits **FL** (or **AFL**) encoding may be used.

Solution: Outlier aware algorithms

- Patched FL and Patched Aligned FL
- Adaptive FL and Adaptive Aligned FL

Patched Aligned FL

- Patched FL – **Zukowski et al. 2006** for CPU.
- Patched FL (new memory organisation) – **Yan et al. 2009** for CPU.
- Patched FL on GPU – **Kaczmarski and Przymus 2012, 2013**.

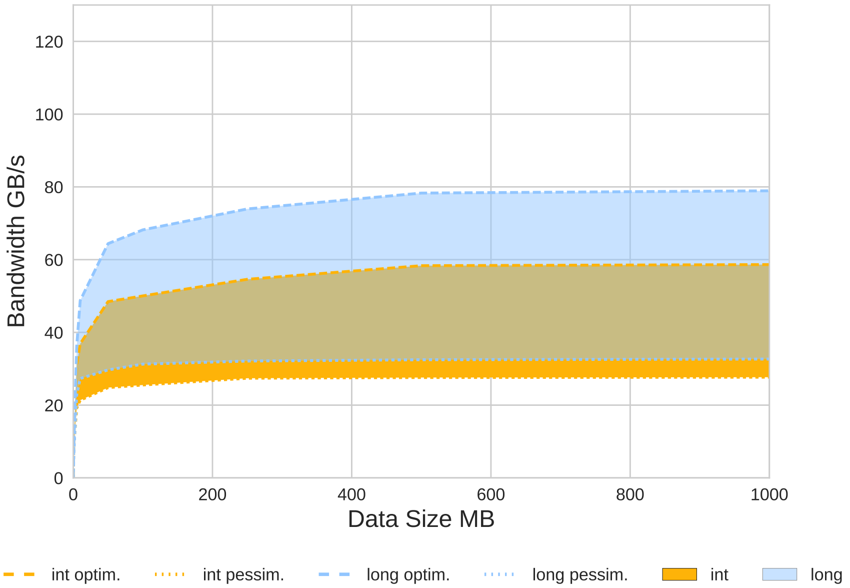
Structure:

- compressed data, exceptions positions, exceptions values.

Patched Aligned FL:

- One step compression:
  - threads gather outliers in local memory buffers,
  - buffer overflow is managed per warp (voting of threads).
- Two step decompression:
  - step 1: AFL decompression,
  - step 2: exceptions extraction.

Patched Aligned FL on GPU (C+D, GTX Titan Black)



Adaptive FL

First introduced by **Delbru et al. 2010** for CPU:

- processes data in chunks,
- sets different bit length encoding for each chunk.

Adaptive FL does not fit GPU memory model very well.

Adaptive Aligned FL:

- Aligned FL + new organisation of compressed chunks.
- Bit length encoding is established per warp.
- Compressed chunks are order according to warps completion time.

Note that warps may finish in different order then data order.

Adaptive Aligned FL on GPU

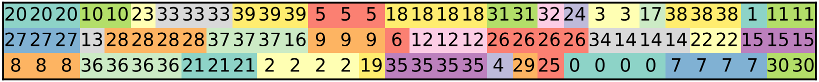


Figure: Adaptive aligned memory organisation: main compression array.

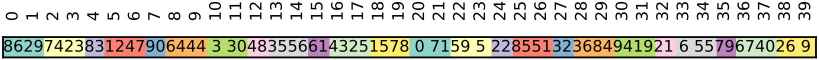
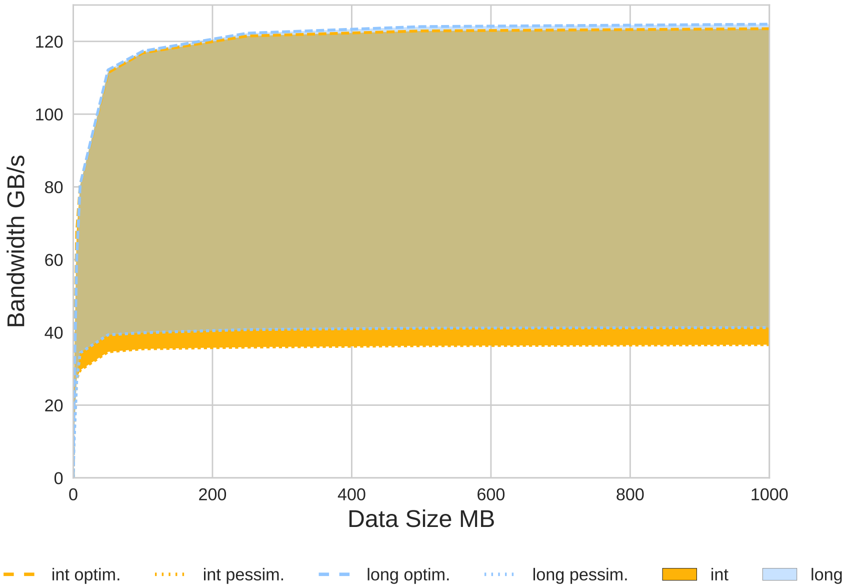


Figure: Adaptive aligned memory organisation: warp offset index.

Adaptive Aligned FL on GPU (C+D, GTX Titan Black)



TPC-H Data Benchmark

Column	Org. / used	Sort.	Size MB	Alg.	CR	Bandwidth [GB/s] Comp.	Bandwidth [GB/s] Decomp.
l_discount	dec. / int(4)	N	239	FL	8.00	52.7137	34.0756
				AFL	8.00	211.8846	181.8800
				PAFL	8.00	200.6869	72.8897
				AAFL	30.79	101.4848	125.6707
l_quantity	int(4) / int(4)	N	239	FL	2.46	45.4952	32.0122
				AFL	2.46	154.2067	141.1335
				PAFL	2.46	153.2897	65.5559
				AAFL	9.72	81.5276	105.6029
l_partkey	id / int(4)	N	239	FL	1.52	37.7608	29.4157
				AFL	1.52	125.7666	120.2389
				PAFL	1.52	125.0484	60.6303
				AAFL	6.05	72.4169	93.3693
l_shipdate	date / int(4)	Y	239	FL	1.06	28.2662	26.2429
				DELTA-AFL	1.88	136.6975	127.8505
				DELTA-PAFL	31.91	205.3076	55.5019
				DELTA-AAFL	126.31	210.5169	209.3501

FeatherGPU

- A lightweight compression library for GPU.
- **github.com/mis-wut/feathergpu**
- **MIT**-licenesed.
- **Supported algorithms**:
  - FL, FOR, Adaptive FL, Patched AFL
  - Aligned FL, Aligned FOR, Adaptive Aligned FL, Patched Aligned AFL

GTX Titan Black:

- Compression **up to** 250GB/s
- Decompression **up to** 250GB/s

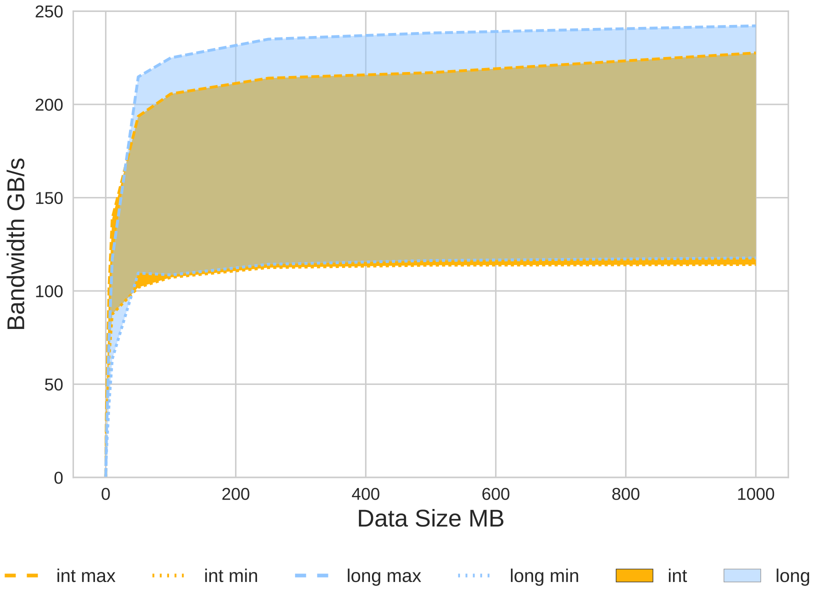
Tesla P100 16 GB (**New!**)

- Compression **up to** 550GB/s
- Decompression **up to** 450GB/s

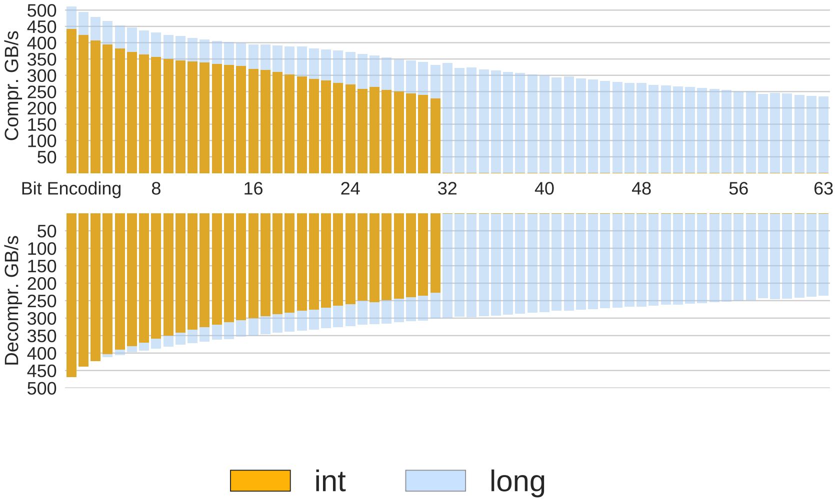
K. Kaczmarski and P. Przymus, *Fixed Length Lightweight Compression for GPU Revised*, Journal of Parallel and Distributed Computing, 2017.

Appendix

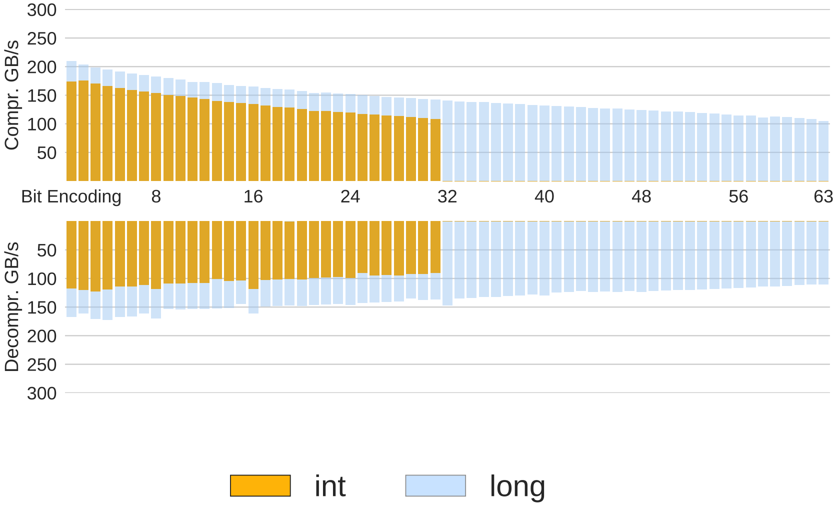
Aligned FL on GPU (C+D, Tesla P100 16 GB)



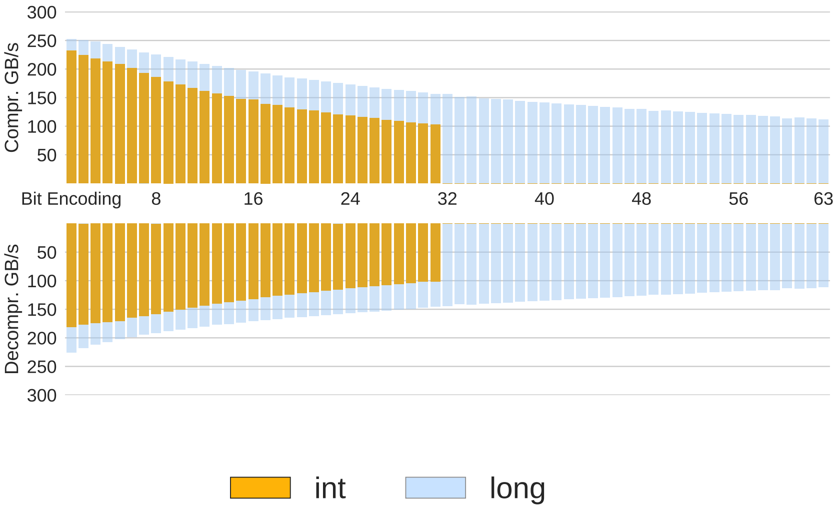
Aligned FL on GPU (1 GB of data, Tesla P100 16 GB)



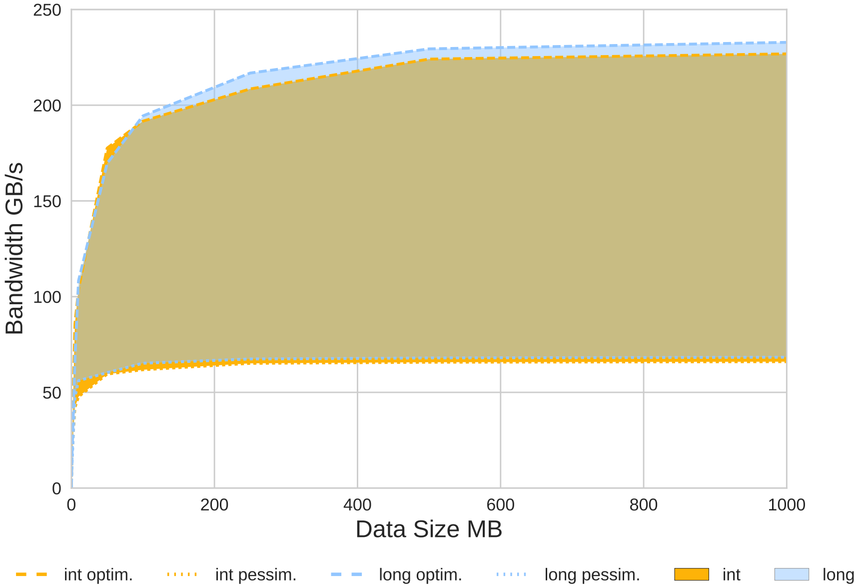
Direct access Aligned FL on GPU



DELTA Aligned FL on GPU

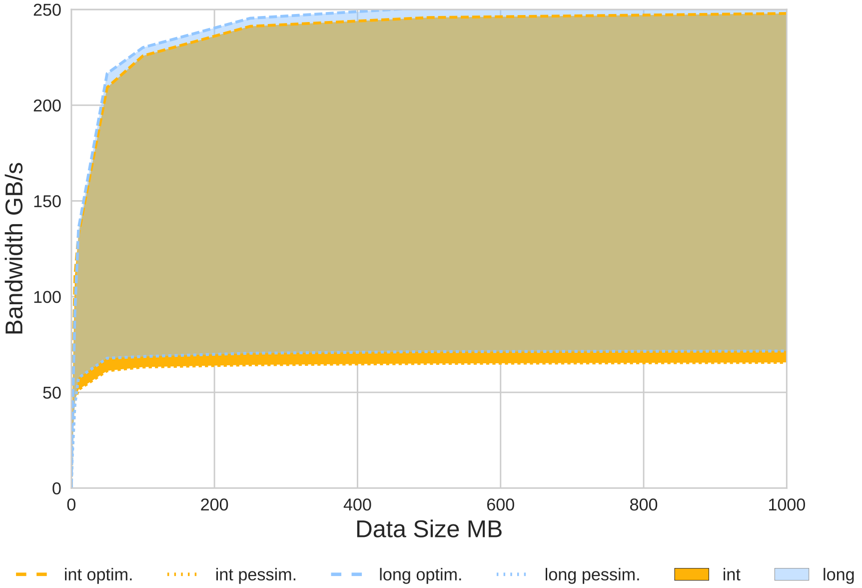


Patched Aligned FL on GPU (compression)



Patched Aligned FL on GPU (decompression)

Adaptive Aligned FL on GPU (compression)



Adaptive Aligned FL on GPU (decompression)