Huffman Coding with Gap Arrays for GPU Acceleration

Naoya Yamamoto, Koji Nakano, Yasuaki Ito, Daisuke Takafuji

Hiroshima University

Akihiko Kasagi, Tsuguchika Tabaru

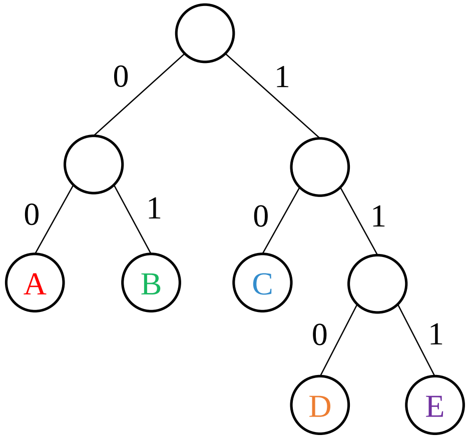
Fujitsu Laboratories

Huffman coding

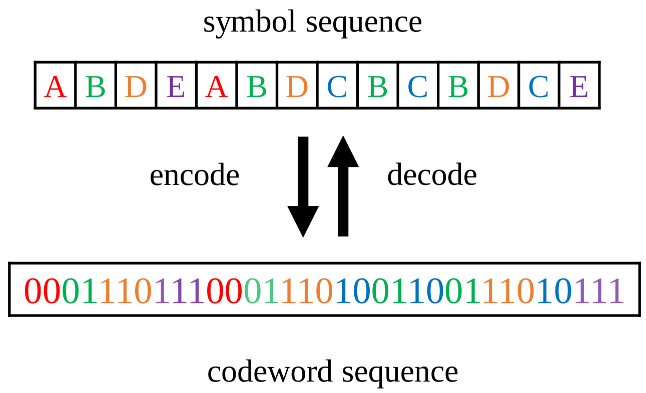
- **Lossless** data compression scheme
- Used in many data compression formats:
  - gzip, zip, png, jpg, etc.
- Uses a **codebook**: mapping of fixed-length (usually 8-bit) symbols into codewords bits.
- **Entropy coding**: Symbols appear more frequently are assigned codewords with fewer bits.
- **Prefix code**: Every codeword is not a prefix of the other codewords.

codebook

symbols	A	B	C	D	E
codeword bits	00	01	10	110	111

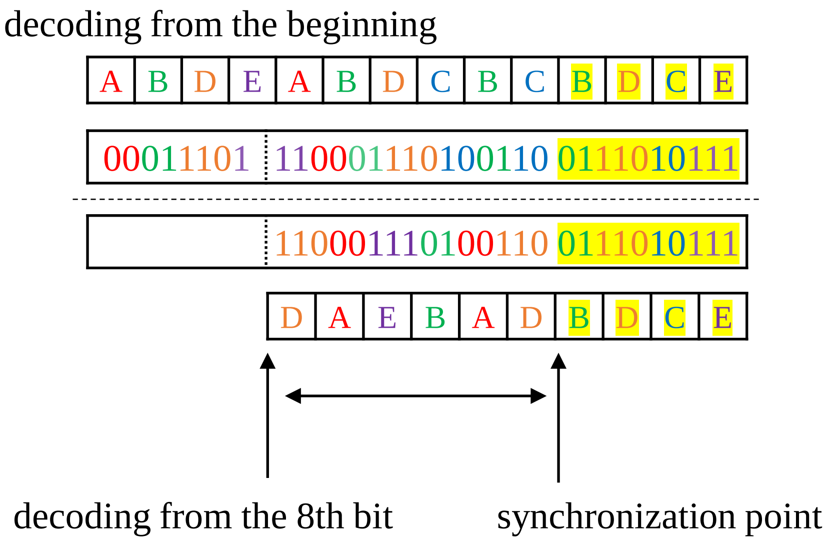


- **Huffman Encoding** can be done by converting each symbol to the corresponding codeword: parallel encoding is easy.
- **Huffman Decoding** can be done by reading the codeword sequence from the beginning
  1. identifying each codeword
  2. converting it into the corresponding codeword
- **Parallel Huffman decoding is hard**:
  - codeword sequence has no separator to identify codewords
  - It is not possible to start decoding from the middle of the codeword sequence.
  - Parallel divide-and-conquer approaches that perform decoding for every equal-sized partitioned segment do not decode correctly: a codeword may be **incomplete** and separated into two segments

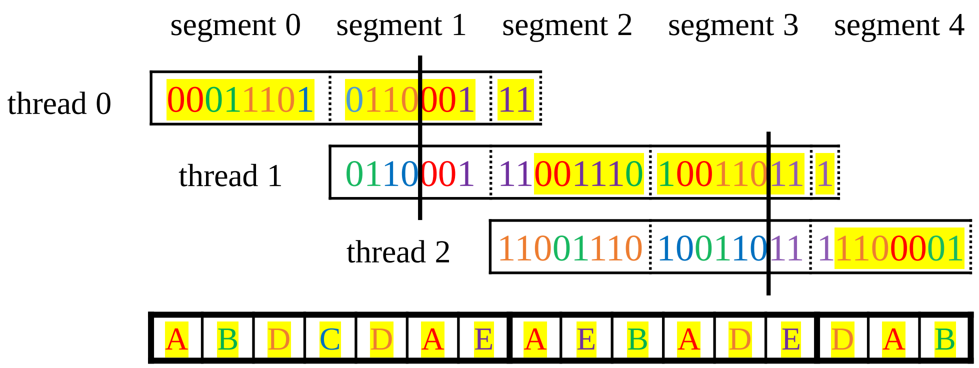


Parallel GPU decoding by self-synchronization

- **Self-synchronization** of Huffman decoding **[3]**
  - Decoding from a middle bit will synchronize.
  - Decoding is correct after synchronization.
  - The expected length for self-synchronization is 73 **[16]**
  - Decoding may never synchronize in the worst case.



- **Parallel GPU decoding by self-synchronization [29,30]**
- The codeword sequence is partitioned into equal-sized segments.
- Each thread is assigned to a segment and starts decoding from it.
- It continues decoding of following segments until it finds synchronization.
- **Drawbacks**
  - Every segment is decoded by two times or more.
  - In the worst case, thread 0 must decode all segments.

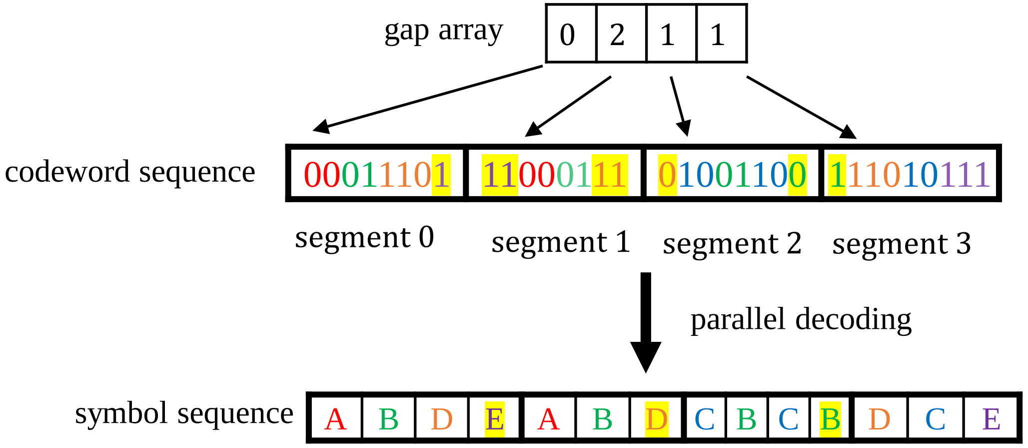


[3] T. Ferguson and J. Rabinowitz. 1984. Self-synchronizing Huffman codes. IEEE Trans. on Information Theory 30, 4 (July 1984), 687 – 693.
[16] S. T. Klein and Y. Wiseman. 2003. Parallel Huffman Decoding with Applications to JPEG Files. Comput. J. 46, 5 (Jan. 2003), 487 – 497.

[29] André Weissenberger. 2018. CUHD - A Massively Parallel Huffman Decoder. https://github.com/weissenberger/gpuhd.
[30] André Weissenberger and Bertil Schmidt. 2018. Massively Parallel Huffman Decoding on GPUs. In Proc. of International Conference on Parallel Processing. 1–10.

Our contribution

- **First contribution:** Present a **gap array,** a new data structure for accelerating parallel decoding
  - the bit position of the first complete codeword in each segment
  - Computed and attached to a codeword sequence when encoding is performed
- **Gap array is very small:** array of 4 bits
  - the size overhead is less than 1.5% for 256-bit segments
  - the time overhead for GPU encoding is less than 20%.
- **Gap array accelerate GPU decoding**
  - 1.67x – 6450x faster



- **Second contribution:** Develop several acceleration techniques for Huffman encoding/decoding

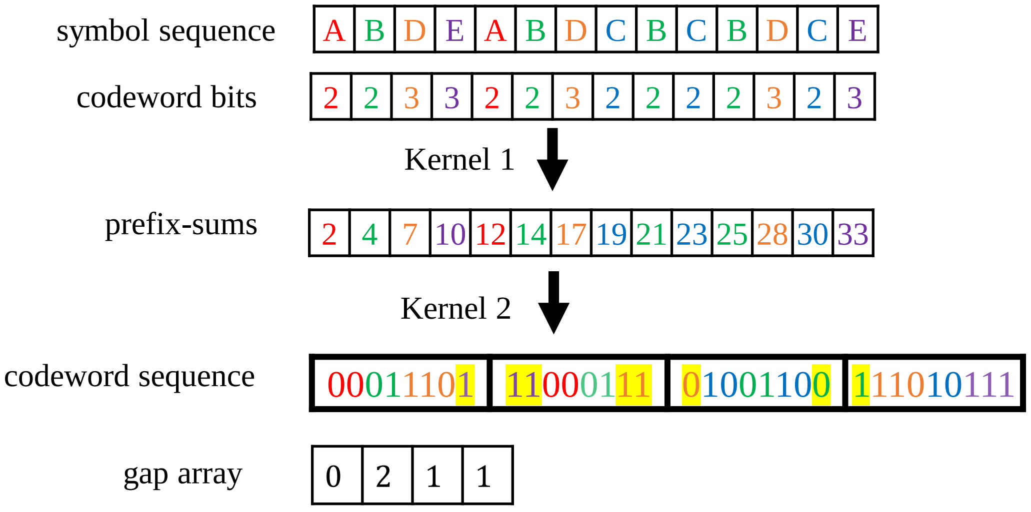
1. **Single Kernel Soft Synchronization(SKSS)** technique **[9]**
   - Only one kernel call is performed.
   - Kernel call and global memory access overhead can be reduced
2. **Wordwise global memory access**
   - four 8-bit symbols (32 bits) are read/write by one instruction.
3. **Compact codebook:** new data structure for codebooks of Huffman coding
   - Codebook size can be 64Kbytes : too large to store it in the GPU shared memory
   - The size is reduced to less than 3 Kbytes: enough small to store it in the GPU shared memory

- **Experimental results for a data set of 10 files**
  - Our GPU encoding/decoding is 2.87x-7.70x and 1.26-2.63x faster than previous presented GPU implementations.
  - If a gap array is available, our GPU decoding is 1.67x-6450x times faster.

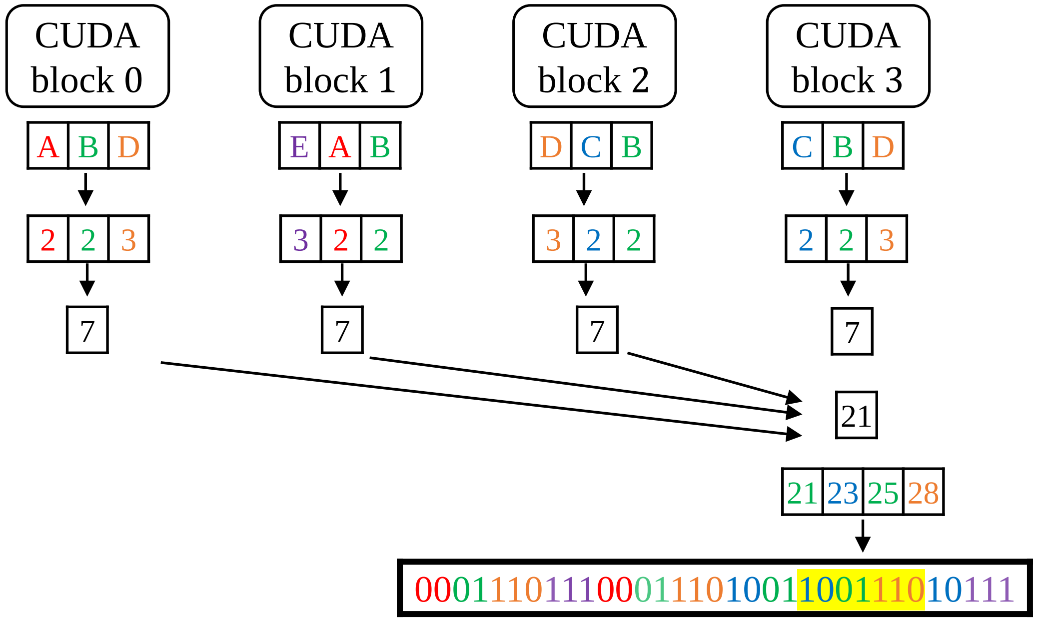
[9] Shunji Funasaka, Koji Nakano, and Yasuaki Ito. 2017. Single Kernel Soft Synchronization Technique for Task Arrays on CUDA-enabled GPUs, with Applications. In Proc. International Symposium on Networking and Computing. pp.11–20.

GPU Huffman encoding with a gap array

- **Naive Parallel GPU encoding**
- Kernel 1: The prefix-sums of codeword bits are computed.
  - The bit position of the codeword corresponding to each symbol can be determined from the prefix-sums.
- Kernel 2: The codeword of corresponding to each symbol is written.
  - Gap arrays can be written if necessary.
- Both Kernels 1 and 2 perform global memory access.

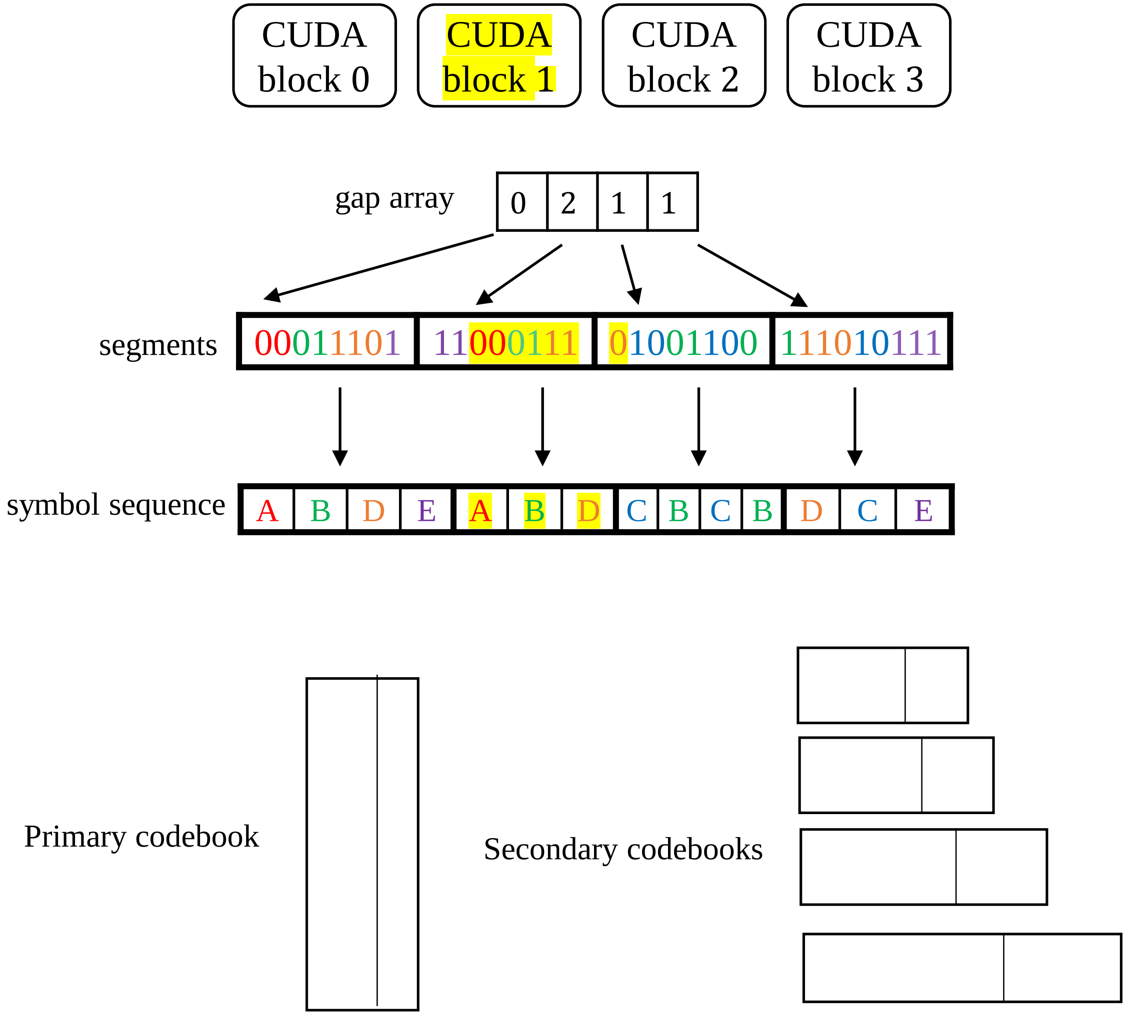


- GPU encoding by **the Single Kernel Soft Synchronization** (**SKSS**)
  - Only one kernel call is performed.
  - Reduce global memory access
- The codeword sequence are partitioned into equal-sized segments.
- Each CUDA block *i* (this number is assigned by a global counter) works for encoding segment *i*
- The Prefix-sums for each segment *i* are computed by looking back previous CUDA blocks



GPU Huffman decoding with a gap array

- **SKSS technique**:
  - The codeword sequence is partitioned into equal-sized segments and the gap value of each segment is available.
  - Each CUDA block *i* (this number is assigned by a global counter) works for decoding a segment *i*
  - Since the gap value is available, each CUDA block can start decoding from the first complete codeword.
  - Similarly to GPU Huffman decoding, the prefix-sums of the number of symbols corresponding to segments are computed by the SKSS.
  - From the prefix-sums, each CUDA block can determine the position in the symbol sequence where it writes the decoded symbols.
- **Compact codebook:**
  - A 64Kbyte codebook is separated into several small codebooks.
  - Primary codebook: stores codewords with no more than 11 bits
  - Secondary codebooks: store codewords with 11 bits or more
  - The total size is less than 3 Kbytes.
- **wordwise memory access**
  - 4 symbols are written as a 32-bit word.
  - Global memory access throughput can be improved.



Experimental results: Data set of 10 files

Compression ratio

NOGAP: Original Huffman code with no gap array

GAP: Huffman code with gap array for 256-bit segment

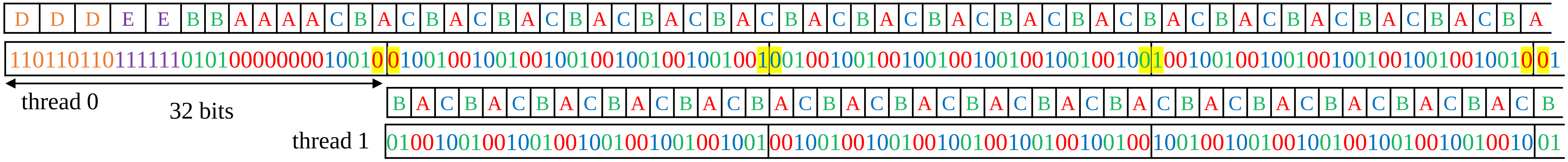
file	type	contents	size(Mbyte)	NOGAP	GAP	GAP Overhead
bible	text	Collection of sacred texts or scriptures	4.047	54.82%	55.67%	**+0.86%**
enwiki	xml	Wikipedia dump file	1095.488	68.30%	69.37%	**+1.07%**
mozilla	exe	Tarred executables of Mozilla	51.220	78.05%	79.27%	**+1.22%**
mr	image	Medical magnetic resonance image	9.971	46.37%	47.10%	**+0.72%**
nci	database	Chemical database of structures	33.553	30.47%	30.95%	**+0.48%**
prime	text	50th Mersenne number	23.714	44.12%	44.80%	**+0.69%**
sao	bin	The SAO star catalog	7.252	94.37%	95.85%	**+1.47%**
webster	html	The 1913 Webster Unabridged Dictionary	41.459	62.54%	63.52%	**+0.98%**
linux	src	Linux kernel 5.2.4	871.352	70.23%	71.32%	**+1.10%**
malicious	text	Never self-synchronizes until the end	1073.742	25.00%	25.39%	**+0.39%**

size overhead
+0.39% – +1.47%

NOGAP → GAP

$$\text{Compression ratio} = \frac{\text{compressed size}}{\text{uncompressed size}}$$

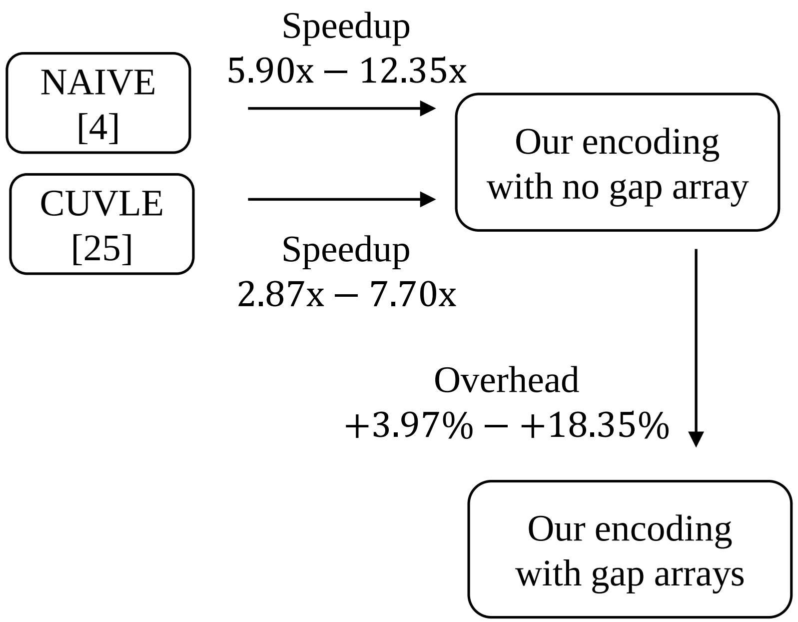
malicious: text that never self-synchronizes



Experimental results: GPU Huffman encoding

Running time : Nvidia Tesla V100

	with no gap array					with gap arrays	
file	NAIVE [25]	CUVLE [4]	Our encoding with no gap array	Speedup over NAIVE[25]	Speedup over CUVLE[4]	Our encoding with gap arrays	Gap array overhead
bible	0.747ms	0.180ms	**0.0605ms**	12.35x	2.98x	**0.0716ms**	**+18.35%**
enwiki	70.8ms	37.7ms	**6.53ms**	10.84x	5.77x	**7.05ms**	**+7.96%**
mozilla	4.55ms	1.97ms	**0.451ms**	10.09x	4.37x	**0.495ms**	**+9.76%**
mr	1.11ms	0.407ms	**0.119ms**	9.33x	3.42x	**0.134ms**	**+12.61%**
nci	2.00ms	1.31ms	**0.339ms**	5.90x	3.86x	**0.365ms**	**+7.67%**
prime	1.52ms	0.926ms	**0.175ms**	8.69x	5.29x	**0.193ms**	**+10.29%**
sao	1.21ms	0.307ms	**0.107ms**	11.31x	2.87x	**0.123ms**	**+14.95%**
webster	3.27ms	1.62ms	**0.303ms**	10.79x	5.35x	**0.332ms**	**+9.57%**
linux	55.0ms	30.0ms	**5.59ms**	9.84x	5.37x	**6.05ms**	**+8.23%**
malicious	36.0ms	36.9ms	**4.79ms**	7.52x	7.70x	**4.98ms**	**+3.97%**



[4] Antonio Fuentes-Alventosa, Juan Gomez-Luna ; JoseM Gonzalez-Linares, and Nicolas Guil. 2014. CUVLE: Variable-Length Encoding on CUDA. In *Proc. Con- ference on Design and Architectures for Signal and Image Processing*. 1–6.

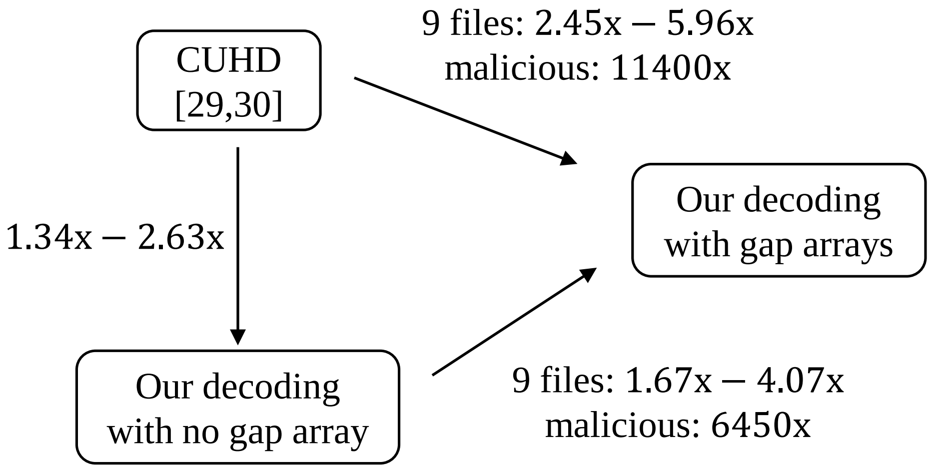
[25] Habibelahi Rahmani, Cihan Topal, and Cuneyt Akinlar. 2014. A parallel Huffman coder on the CUDA architecture. In *Proc. of IEEE Visual Communications and Image Processing Conference*. 311–314.

Experimental results: GPU Huffman decoding

Running time : Nvidia Tesla V100

	with no gap array by self-synchronization			with gap arrays		
file	CUHD [29,30]	Our decoding with no gap array	Speedup over CUHD [29,30]	Our decoding with gap arrays	Speedup over CUHD [29,30]	Speedup over our decoding with no gap array
bible	0.331ms	0.205ms	**1.61x**	0.0682ms	**4.85x**	**3.01x**
enwiki	40.3ms	22.3ms	**1.81x**	10.5ms	**3.84x**	**2.12x**
mozilla	3.67ms	2.74ms	**1.34x**	0.674ms	**5.45x**	**4.07x**
mr	0.64ms	0.461ms	**1.39x**	0.261ms	**2.45x**	**1.77x**
nci	1.90ms	0.923ms	**2.06x**	0.552ms	**3.44x**	**1.67x**
prime	1.67ms	0.636ms	**2.63x**	0.280ms	**5.96x**	**2.27x**
sao	0.472ms	0.278ms	**1.70x**	0.120ms	**3.93x**	**2.32x**
webster	1.76ms	0.906ms	**1.94x**	0.488ms	**3.61x**	**1.86x**
linux	34.6ms	21.3ms	**1.62x**	9.04ms	**3.83x**	**2.36x**
malicious	106000ms	60000ms	**1.77x**	9.30ms	**11400x**	**6450x**

Speedup

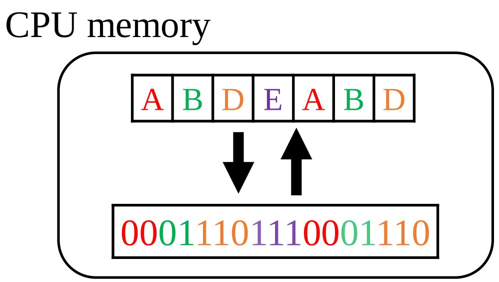


wordwise global memory access
compact codebook

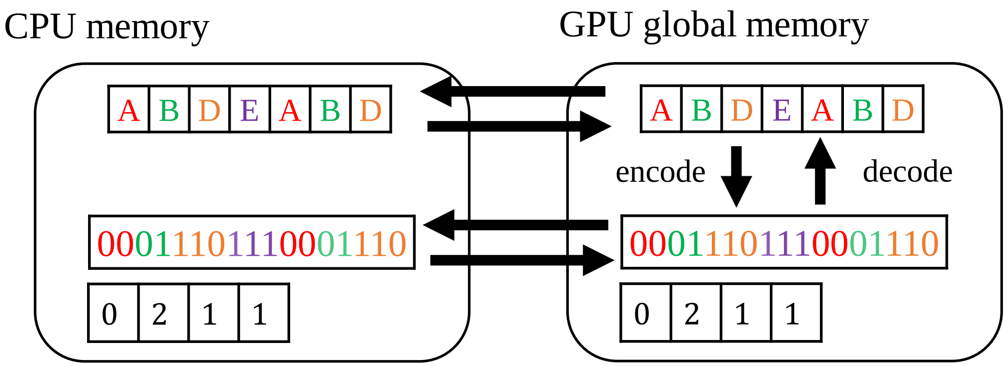
[29] André Weissenberger. 2018. CUHD - A Massively Parallel Huffman Decoder. https://github.com/weissenberger/gpuhd.
[30] André Weissenberger and Bertil Schmidt. 2018. Massively Parallel Huffman Decoding on GPUs. In Proc. of International Conference on Parallel Processing. 1–10.

Huffman coding with gap arrays: CPU vs. GPU

CPU encoding/decoding with no gap array



GPU encoding/decoding with gap arrays



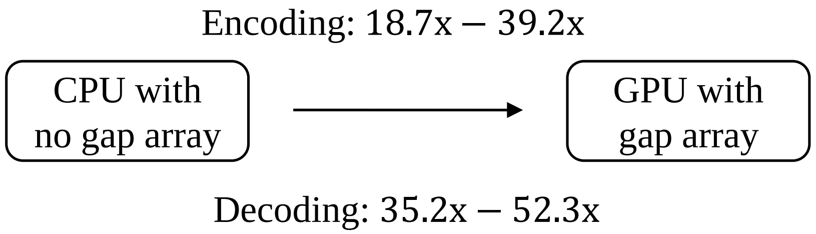
The time for all necessary operations are included:

- Computing symbol frequency by histogramming
- Codebook generation
- Data transfer time between CPU/GPU

Running time

CPU:Intel Xeon Silver 4112 (2.60GHz)
GPU:Nvidia Telsa V100

	Huffman encoding			**Huffman decoding**		
file	CPU	GPU	Speedup	CPU	GPU	Speedup
bible	47.0ms	**1.20ms**	**39.2x**	25.9ms	**0.598ms**	**43.3x**
enwiki	3500ms	**158ms**	**22.2x**	5930ms	**159ms**	**37.2x**
mozilla	313ms	**8.67ms**	**36.1x**	308ms	**7.95ms**	**38.7x**
mr	67.0ms	**2.05ms**	**32.7x**	52.9ms	**1.50ms**	**35.2x**
nci	177ms	**5.50ms**	**32.2x**	170ms	**4.48ms**	**37.9x**
prime	80.0ms	**4.27ms**	**18.7x**	160ms	**3.06ms**	**52.2x**
sao	75.2ms	**3.15ms**	**23.9x**	49.3ms	**1.28ms**	**38.4x**
webster	174ms	**7.31ms**	**23.8x**	248ms	**5.94ms**	**41.7x**
linux	3130ms	**128ms**	**24.5x**	4890ms	**128ms**	**38.3x**
malicious	2250ms	**117ms**	**19.2x**	4500ms	**119ms**	**37.8x**



Conclusion

- We have presented new data structure **gap array** for accelerating Huffman decoding on GPUs.
- We have also presented several acceleration techniques for Huffman encoding/decoding on GPUs.
- The size overhead of gap arrays is small: $+0.39\% - +1.47\%$
- The time overhead of gap arrays in GPU Huffman encoding is small: $+3.97\% - +18.35\%$
- GPU Huffman decoding is much faster if gap arrays are available:
  - 9 files: 1.67x–4.07x
  - malicious file : 6450x
- Including all operations for Huffman encoding/decoding and CPU-GPU data transfer, GPU can accelerate Huffman encoding/decoding
  - Encoding: $18.7x - 39.2x$
  - Decoding: $35.2x - 52.3x$
- Gap arrays should be attached if Huffman encoding/decoding are performed using GPUs.