Data Coding and Compression, FIT, BUT

Huffman and RLE Compression of Grayscale Images

David Mihola
xmihol00

12th of May, 2024

1 Problem Formulation

The overall goal of this project is to compress and decompress grayscale images with the use of Huffman coding and run length encoding (RLE) lossless compression techniques. Moreover, the compression/decompression algorithms shall support the 4 following modes:

- **static:** where the input data are perceived as 1 dimensional,
- **adaptive:** where the input data are perceived as 2 dimensional and the input is further partitioned into smaller blocks of constant size for adaptive serialization,
- **static with a model:** where a context model is applied during the regular static compression/decompression,
- **adaptive with a model:** where a context model is applied during the regular adaptive compression/decompression,

Last but not least, apart from achieving the best possible compression rates, the speed of the compression and decompression shall be greatly considered.

1.1 Huffman Coding

Huffman code is an optimal[1] variable length prefix code. The symbol codes are derived from their probability distribution using a binary tree. The tree nodes are constructed from bottom up, starting with the symbol probabilities as initial nodes, such that 2 nodes with the smallest probability are merged into a new node with the sum of theirs probabilities, until only a single node remains. Codes are then assigned by a top to bottom traversal of the tree, where at each branching the current code is extended by 0 bit for the left branch and 1 bit for the right branch or vice versa. Meaning that the length of a code for a symbol is solely determined by its depth in the constructed tree.
When symbols at each depth of the tree are sorted, the depth property allows to derive the symbol codes so that there exists just a single possible assignment of each code to a symbol, such codes are called canonical Huffman codes. Meaning that only the depth of each symbol must be transferred from the encoder to the decoder.

1.2 Run Length Encoding

Run length encoding (RLE) is a lossless compression technique, in which repeating symbols (runs) are encoded as the value of the symbol[2] followed by the number of its repetitions.
There are several methods how to encode runs of symbols. From encoding a number of repetitions of each symbol, even if it is an isolated occurrence, to using triplets with the `<triplet indicating symbol, repeated symbol, number of repetitions>` format, to encoding runs only of the

[1]for symbol probabilities equaling to negative powers of 2
[2]raw, e.g. ASCII, or encoded, e.g. with a Huffman code

most frequent symbol using a specific binary code[3], to encoding only runs longer than or equal to 3 symbols as a triplet of the 3 same symbol codes followed by the number of repetitions.
The last named approach is particularly useful for input data in which all possible symbols are exhausted and no triplet indicating symbol is available. This is in general the case, when symbols represent values of pixels in an image.

2 Implementation Details

The implemented algorithms were from the beginning designed to be multi-threaded and with a great attention to performance, i.e. compression rate may be in some cases negatively affected to enable faster computation.
The compression consists of the following higher level operations:

1. Application of a context model on the input data if compression with model is active. Pixel difference model[4] is used.
2. Adaptive serialization of the input data if adaptive compression is activate.
3. Computation of symbol frequencies, i.e. unnormalized symbol probabilities.
4. Construction of a Huffman tree, tree balancing and assignment of Huffman codes to symbols.
5. Encoding of the serialized input data with the derived Huffman codes and RLE.

Inversely, the decompression is composed of the following higher level operations:

1. Reconstruction of Huffman codes for the compressed data.
2. Decoding of the compressed input data, i.e. reversal of the Huffman coding and RLE.
3. Deserialization of the decoded data, if adaptive decompression is active.
4. Reversal of the pixel difference model, if decompression with model is active.

The following sections further expand on some of the higher level operations listed above.

2.1 Adaptive Serialization

Horizontal zig-zag, vertical zig-zag, major diagonal zig-zag and minor diagonal zig-zag traversals of 16x16 sub-patches of the input data were used. Zig-zag traversals, i.e. where paths are not cut off at the block edges but they alternate directions, should be preferred for images compression over regular traversals thanks to their ability to better preserve the locality of neighboring pixels.

[3]RLE0 technique

[4]Values of neighbouring pixels in 1 dimensionally perceived input data are subtracted from each other (`current_diff = current - previous`).

However, the code has been designed to facilitate an easy replacement of the used traversals by simply providing a different set of indices specifying the desired traversal pattern[5], may it better suite the expected inputs.

2.2 Huffman Tree Construction and Huffman Code Assignment

The computed symbol frequencies are clamped to the maximum[6] of $2^{24} - 2$ and packed into a 32-bit data structure, where the 8 least significant bits (LSBs) represent the symbol value. The Huffman tree is then constructed using an algorithm described in [2] adopted to 512-bit registers and rebalanced to have a maximal depth of 16 with an algorithm inspired by [1].
Moreover, it is ensured that there are at most 32 unique prefixes across the different depths of the tree by iteratively rounding the number of symbols at each depth. The rounding factor increases[7] after each unsuccessful iteration to ensure convergence. This allows to perform an AVX mask decoding of the assigned Huffman codes, see section 2.3.
Furthermore, symbols at their final depths are sorted in an ascending order based on their values. 256-bit registers are used for each occupied depth as bitmaps, where bits are set to 1 for each symbol appearing at the respective depth, i.e. a radix sort with a radix of 256 is used. Such populated depth bitmaps are then finally used to assign canonical Huffman codes to symbols, but also to transfer the Huffman tree to the decoder. For the latter case, the bitmaps are compressed followingly:

- A 16-bit bitmap is used to indicate which depths are occupied.
- Unseen symbols are placed to depth 0, which is otherwise unused.
- The most populated depth is not transmitted, its content can be fully recovered by xoring a fully populated depth bitmap with all the others.
- Only non 0 bytes of each depth bitmap are transferred, with a prepended further 32-bit bitmap indicating which and how many bytes of the 256-bit bitmap follow.

2.3 Huffman Codes Reconstruction and Decoding

[5] The `traversals_indices.py` script was used to generate the currently used sets of indices.
[6] Value of $2^{24} - 1$ is used to represent an unseen symbol, the 8 LSBs are all set to 1 as well.
[7] At iteration N N LSBs are cleared from a variable representing the number of symbols at a specific depth and the cleared symbols are moved to lower depth, which is occupied, with its capacity being increased accordingly.

3 Data Analysis

	Uncompressed	Compressed file size				Compression ratio				Space savings			
	file size	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
df1h.raw	262144	262145	262145	262145.00	0.00	1.00	1.00	1.00	0.00	-0.00	-0.00	-0.00	0.00
df1hvx.raw	262144	97135	97221	97163.50	32.15	2.70	2.70	2.70	0.00	0.63	0.63	0.63	0.00
df1v.raw	262144	2316	2408	2346.83	34.37	108.86	113.19	111.72	1.61	0.99	0.99	0.99	0.00
hd01.raw	262144	112295	112491	112357.17	74.33	2.33	2.33	2.33	0.00	0.57	0.57	0.57	0.00
hd02.raw	262144	106540	106726	106599.17	69.98	2.46	2.46	2.46	0.00	0.59	0.59	0.59	0.00
hd07.raw	262144	156437	156639	156501.67	76.38	1.67	1.68	1.68	0.00	0.40	0.40	0.40	0.00
hd08.raw	262144	128923	129119	128985.00	73.61	2.03	2.03	2.03	0.00	0.51	0.51	0.51	0.00
hd09.raw	262144	243510	243630	243550.67	44.23	1.08	1.08	1.08	0.00	0.07	0.07	0.07	0.00
hd12.raw	262144	199950	200138	200011.00	71.05	1.31	1.31	1.31	0.00	0.24	0.24	0.24	0.00
nk01.raw	262144	244424	244512	244455.17	32.50	1.07	1.07	1.07	0.00	0.07	0.07	0.07	0.00

Table 1: Static compression without a model

	Uncompressed	Compressed file size				Compression ratio				Space savings			
	file size	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
df1h.raw	262144	65804	65916	65841.33	41.87	3.98	3.98	3.98	0.00	0.75	0.75	0.75	0.00
df1hvx.raw	262144	76391	76473	76419.33	30.79	3.43	3.43	3.43	0.00	0.71	0.71	0.71	0.00
df1v.raw	262144	65804	65916	65841.33	41.87	3.98	3.98	3.98	0.00	0.75	0.75	0.75	0.00
hd01.raw	262144	110505	110711	110572.17	78.04	2.37	2.37	2.37	0.00	0.58	0.58	0.58	0.00
hd02.raw	262144	104746	104944	104809.83	74.67	2.50	2.50	2.50	0.00	0.60	0.60	0.60	0.00
hd07.raw	262144	154477	154689	154545.67	79.96	1.69	1.70	1.70	0.00	0.41	0.41	0.41	0.00
hd08.raw	262144	125789	125991	125854.33	76.99	2.08	2.08	2.08	0.00	0.52	0.52	0.52	0.00
hd09.raw	262144	240826	240934	240860.67	40.21	1.09	1.09	1.09	0.00	0.08	0.08	0.08	0.00
hd12.raw	262144	196382	196614	196456.33	87.83	1.33	1.33	1.33	0.00	0.25	0.25	0.25	0.00
nk01.raw	262144	244296	244388	244326.17	34.48	1.07	1.07	1.07	0.00	0.07	0.07	0.07	0.00

Table 2: Adaptive compression without a model

	Uncompressed file size	Compressed file size				Compression ratio				Space savings			
		min	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
df1h.raw	262144	19	157	62.67	52.19	1669.71	13797.05	6848.90	4559.57	1.00	1.00	1.00	0.00
df1hvx.raw	262144	36490	36558	36513.67	25.32	7.17	7.18	7.18	0.00	0.86	0.86	0.86	0.00
df1v.raw	262144	1295	1473	1377.33	69.84	177.97	202.43	190.74	9.70	0.99	1.00	0.99	0.00
hd01.raw	262144	102040	102244	102105.50	76.54	2.56	2.57	2.57	0.00	0.61	0.61	0.61	0.00
hd02.raw	262144	85755	85919	85808.83	62.42	3.05	3.06	3.05	0.00	0.67	0.67	0.67	0.00
hd07.raw	262144	148610	148810	148675.33	75.78	1.76	1.76	1.76	0.00	0.43	0.43	0.43	0.00
hd08.raw	262144	97558	97734	97613.67	66.28	2.68	2.69	2.69	0.00	0.63	0.63	0.63	0.00
hd09.raw	262144	173643	173757	173682.33	42.07	1.51	1.51	1.51	0.00	0.34	0.34	0.34	0.00
hd12.raw	262144	159052	159236	159112.00	69.28	1.65	1.65	1.65	0.00	0.39	0.39	0.39	0.00
nk01.raw	262144	198172	198276	198206.00	38.69	1.32	1.32	1.32	0.00	0.24	0.24	0.24	0.00

Table 3: Static compression with a difference model

	Uncompressed file size	Compressed file size				Compression ratio				Space savings			
		min	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
df1h.raw	262144	275	413	318.67	52.19	634.73	953.25	838.97	120.45	1.00	1.00	1.00	0.00
df1hvx.raw	262144	29160	29224	29183.00	24.02	8.97	8.99	8.98	0.01	0.89	0.89	0.89	0.00
df1v.raw	262144	949	1085	1021.00	58.15	241.61	276.23	257.46	14.96	1.00	1.00	1.00	0.00
hd01.raw	262144	100644	100860	100713.50	81.69	2.60	2.60	2.60	0.00	0.62	0.62	0.62	0.00
hd02.raw	262144	84917	85097	84975.50	67.60	3.08	3.09	3.08	0.00	0.68	0.68	0.68	0.00
hd07.raw	262144	146528	146744	146597.00	82.09	1.79	1.79	1.79	0.00	0.44	0.44	0.44	0.00
hd08.raw	262144	95918	96096	95975.33	66.94	2.73	2.73	2.73	0.00	0.63	0.63	0.63	0.00
hd09.raw	262144	171235	171359	171276.00	46.74	1.53	1.53	1.53	0.00	0.35	0.35	0.35	0.00
hd12.raw	262144	156268	156486	156338.33	82.31	1.68	1.68	1.68	0.00	0.40	0.40	0.40	0.00
nk01.raw	262144	198330	198430	198364.33	37.31	1.32	1.32	1.32	0.00	0.24	0.24	0.24	0.00

Table 4: Adaptive compression with a difference model

	Without a model				With a difference model			
	Horizontal	**Vertical**	**Minor diagonal**	**Major diagonal**	**Horizontal**	**Vertical**	**Minor diagonal**	**Major diagonal**
df1h.raw	0	1024	0	0	1024	0	0	0
df1hvx.raw	540	484	0	0	484	332	0	208
df1v.raw	1024	0	0	0	1022	0	0	2
hd01.raw	695	214	36	79	741	177	35	71
hd02.raw	679	216	49	80	716	192	51	65
hd07.raw	381	127	182	334	447	158	152	267
hd08.raw	499	209	123	193	560	222	106	136
hd09.raw	200	246	239	339	263	244	228	289
hd12.raw	369	182	172	301	441	170	162	251
nk01.raw	780	146	22	76	891	90	11	32

Table 5: Applied zig-zag block traversals for adaptive compression without and with a difference model

	Static compression without a model	**Adaptive compression without a model**	**Static compression with a difference model**	**Adaptive compression with a difference model**
df1h.raw	–	3	8	8
df1hvx.raw	54	54	93	93
df1v.raw	3	3	20	20
hd01.raw	80	80	85	85
hd02.raw	71	71	90	90
hd07.raw	86	86	81	81
hd08.raw	60	60	143	143
hd09.raw	83	83	94	94
hd12.raw	53	53	77	77
nk01.raw	93	93	87	87

Table 6: Compressed Huffman tree sizes

4 Performance Analysis

	1 thread		2 threads		4 threads		8 threads		16 threads		32 threads	
	mean	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**
df1h.raw	4063.66	183.44	3053.22	547.61	2615.22	200.48	2477.06	212.09	2651.40	219.48	13961.96	11473.40
df1hvx.raw	3005.82	137.20	2512.10	975.12	2046.40	154.87	1962.40	181.17	2100.52	162.77	11815.76	10517.39
df1v.raw	2202.20	132.18	1823.42	139.02	1661.02	144.02	1713.36	138.47	1934.66	172.79	11162.30	10579.77
hd01.raw	2949.50	148.29	2348.86	186.09	2128.62	140.06	2027.68	138.91	2152.98	157.87	11511.98	11094.16
hd02.raw	2929.92	175.79	2293.06	140.24	2145.44	162.37	2060.00	171.85	2200.00	174.62	14578.90	11713.34
hd07.raw	3411.50	153.97	2668.62	180.63	2278.68	187.11	2174.16	186.48	2284.70	184.68	13214.12	11494.07
hd08.raw	3145.90	168.67	2574.44	182.94	2203.20	163.22	2111.48	162.85	2217.52	162.09	12803.02	11300.56
hd09.raw	3952.78	245.45	2932.64	171.54	2504.34	154.45	2367.56	175.22	2437.26	172.54	10370.74	10076.15
hd12.raw	3478.12	200.76	2744.64	169.64	2355.86	166.73	2246.08	177.54	2342.92	170.50	15494.04	12469.59
nk01.raw	3747.36	199.44	2873.94	164.02	2536.40	435.64	2362.14	190.79	2458.32	214.06	16009.74	13376.36

Table 7: Performance of full static compression without a model in microseconds

	1 thread		2 threads		4 threads		8 threads		16 threads		32 threads	
	mean	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**
df1h.raw	4281.88	156.52	2964.32	134.35	2399.54	152.59	2171.28	170.70	2252.26	203.57	14203.72	11250.28
df1hvx.raw	4518.30	140.53	3113.54	137.55	2555.22	631.27	2159.52	166.14	2284.64	206.69	13161.34	11185.83
df1v.raw	4423.54	141.47	3001.76	156.62	2424.48	221.66	2185.74	179.27	2225.70	181.01	11477.02	10398.39
hd01.raw	4734.32	169.14	3237.40	166.83	2627.02	220.56	2347.22	155.28	2410.00	212.54	10801.14	10341.85
hd02.raw	4684.04	167.54	3242.12	164.84	2626.26	214.99	2301.26	169.20	2309.54	185.43	13630.88	11150.64
hd07.raw	5170.56	171.57	3553.64	173.61	2706.60	146.34	2444.62	196.55	2459.78	193.58	14122.86	11666.26
hd08.raw	5030.32	512.61	3500.68	164.61	2668.48	204.18	2362.56	216.34	2447.44	284.87	10161.82	9593.68
hd09.raw	5656.56	186.44	3834.78	205.19	2983.30	188.07	2592.58	184.19	3099.42	2845.51	15238.36	11792.53
hd12.raw	5249.28	188.52	3603.88	168.36	2832.66	181.16	2531.94	202.35	2555.46	214.32	11920.02	10451.98
nk01.raw	5679.84	543.28	3819.80	336.91	2969.56	204.47	2609.54	189.74	2619.60	210.40	12953.74	11935.36

Table 8: Performance of full adaptive compression without a model in microseconds

	1 thread		2 threads		4 threads		8 threads		16 threads		32 threads	
	mean	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**
df1h.raw	2332.02	121.38	1917.66	231.87	1800.42	166.89	1778.20	186.79	2002.94	215.18	12465.30	12377.45
df1hvx.raw	2719.34	147.74	2150.54	132.63	1995.54	168.65	1922.74	164.30	2080.66	172.10	13320.42	10996.21
df1v.raw	2312.96	132.18	1885.12	155.48	1748.82	159.82	1937.64	1089.92	1921.54	180.77	12225.24	11105.32
hd01.raw	3030.22	151.48	2382.28	167.78	2204.40	174.51	2114.96	179.31	2273.68	194.54	11242.12	10227.06
hd02.raw	2955.64	125.82	2367.38	164.57	2169.02	164.20	2101.48	171.75	2284.32	185.73	8950.98	8689.87
hd07.raw	3660.96	152.67	2803.52	150.07	2403.96	170.32	2296.02	213.64	2346.96	190.31	13764.04	11355.15
hd08.raw	3275.30	152.43	2673.96	149.64	2287.06	171.76	2216.24	405.33	2304.62	187.42	11429.18	10205.85
hd09.raw	4110.78	181.88	3027.14	158.44	2612.46	434.16	2399.18	207.70	2418.70	194.14	10728.06	10172.71
hd12.raw	3756.90	147.37	2895.96	165.68	2541.62	507.14	2281.94	187.10	2404.24	273.07	12482.74	10940.00
nk01.raw	3705.18	211.52	2824.96	193.68	2418.74	173.87	2357.72	233.29	2506.36	232.27	15084.36	11506.03

Table 9: Performance of full static compression with a difference model in microseconds

	1 thread		2 threads		4 threads		8 threads		16 threads		32 threads	
	mean	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**	**mean**	**std**
df1h.raw	4163.08	155.64	2944.54	121.19	2304.54	147.51	2040.54	153.65	2093.90	144.16	14088.82	11182.34
df1hvx.raw	4423.50	127.66	3080.64	165.13	2441.42	140.64	2161.74	163.57	2186.36	170.28	13178.86	10509.04
df1v.raw	4134.66	163.52	2853.38	148.17	2268.92	173.05	2017.08	138.99	2082.46	166.28	12891.58	11083.63
hd01.raw	4877.02	181.16	3373.12	137.60	2760.26	161.73	2426.94	169.44	2399.98	171.87	12995.14	12394.14
hd02.raw	4782.24	176.90	3365.40	164.77	2691.48	167.66	2443.68	379.58	2381.94	226.85	15815.10	11558.13
hd07.raw	5389.88	180.24	3735.14	167.45	2900.82	167.92	2493.66	175.09	2556.84	210.97	12696.66	11235.75
hd08.raw	5099.32	146.10	3655.52	160.59	2787.28	186.15	2420.96	193.81	2433.06	201.25	10781.98	10369.51
hd09.raw	5794.54	183.28	4014.66	191.59	3121.28	182.50	2608.02	186.28	2604.50	337.23	12132.34	11160.99
hd12.raw	5401.40	138.60	3779.34	179.41	2947.02	179.02	2575.86	199.91	2518.92	184.99	13172.76	10799.70
nk01.raw	5564.14	186.14	3828.68	192.94	3046.20	182.80	2610.24	177.62	2706.14	195.70	16678.52	12506.43

Table 10: Performance of full adaptive compression with a difference model in microseconds

References

[1] Cyan. Huffman revisited - part 3 - depth limited tree. http://fastcompression.blogspot.com/2015/07/huffman-revisited-part-3-depth-limited.html, 2015. Accessed: 2024-05-06.

[2] Guilford, J. D., and Nguyen, K. T. Fast computation of huffman codes. https://www.intel.com/content/www/us/en/developer/articles/technical/fast-computation-of-huffman-codes.html, 2016. Accessed: 2024-05-06.