UDC 519.683.4 10.23947/2587-8999-2018-2-2-144-152

# Modified approach to arithmetical decoder performance optimization for aerial photography images compression

**R. V. Arzumanyan****

Institute of computer technology and information security of Southern federal university, Taganrog, Russia

This article is devoted to the development of fast arithmetical encoding algorithm for compressing digital images. Subject of this paper is the research of arithmetical encoder complexity dependency on set of criteria beside the problem input size. The task of this article is to research those encoder compartments, which are the most computationally expensive and optimize their implementations. Aim of this research is the development of fast arithmetical encoder as a part of still image codec for compressing the images coming during distant aquatory objects scanning. New mean algorithm complexity estimation method is proposed as well as optimized arithmetical encoder algorithm based on mentioned method. Theoretical research is conducted, results of which are proved with numerical experiment. Actual set of satellite images of Azov sea aquatory was used. Performance of arithmetical encoder is improved by 7%. New method for mean algorithm complexity assessment is proposed which is based on partitioning of inputs array into equivalence classes. Obtained practical results allows increasing newest image compression techniques performance and using them on mobile computational platforms including those, which are installed on UAVs. Theoretical results of this article expands set of methods for assessing the mean algorithm complexity for those cases when number of steps doesn't depend on problem's input size but rather on non-measureable criteria such as memory access pattern to RAM from multiple ALUs.

**Keywords:** arithmetical coding, performance optimization, image compression, mean algorithm complexity, video codec

**Introduction.** One of aquatory conditions monitoring methods is aerial photography in visible- and infra-red ranges using UAV. Photo- and video- capturing by mobile camera means impose a set of limitations on image processing and storage equipment:

1. Energy efficiency of video coding equipment as it directly influences the UAV battery lifetime.

2. Video coding performance as high-resolution media requires high storage capacity. It influences the amount of experimental data UAV can capture & store during the flight.

** E-mail: roman.arzum@gmail.com.

The most widespread still image codec JPEG has hardware support from wast majority of image capture equipment yet it does not yield sufficient compression efficiency in comparison to modern motion- and still- image codecs such as HEVC and VP9, which support both motion and still profiles. At similar visual quality determined by SSIM (structure similarity) metrics VP9 codec yield 25-34% better compression [1]. HEVC codec shows 10-44% better compression level at same visual quality determined by PSNR (peak signal to noise ratio) [2]. Due to superior compression both VP9 and HEVC are significantly more computationally complex [3]. JPEG codec architecture overview is shown by fig. 1. Input frame is diced up by set of 8x8 fixed size blocks known as macro-blocks, each of which undergo discrete cosine transform [4], coefficients quantization and Huffman variable-length lossless encoding [5]. Discrete transform is integer [6], variety of fast algorithms were developed for it since the standard adoption in 1992, which allows for in-register execution. Huffman variable-length coding is also of low computational complexity so that even mobile CPUs can run JPEG codec in software mode.

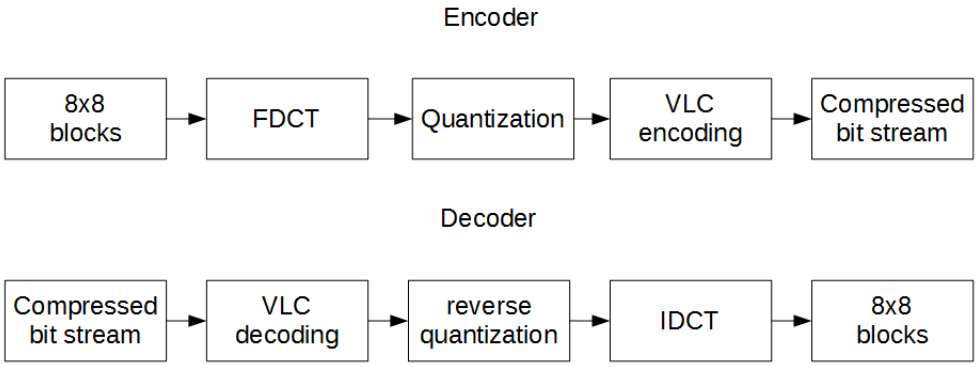


**Fig. 1.** JPEG codec flowchart

HEVC [7] and VP9 have similar high-level design. Both are hybrid block-based codecs with adaptive frame partitioning, intra-frame prediction, discrete transform and in-loop filtering for blocky artifacts removal. HEVC codec high-level design is shown on fig. 2.

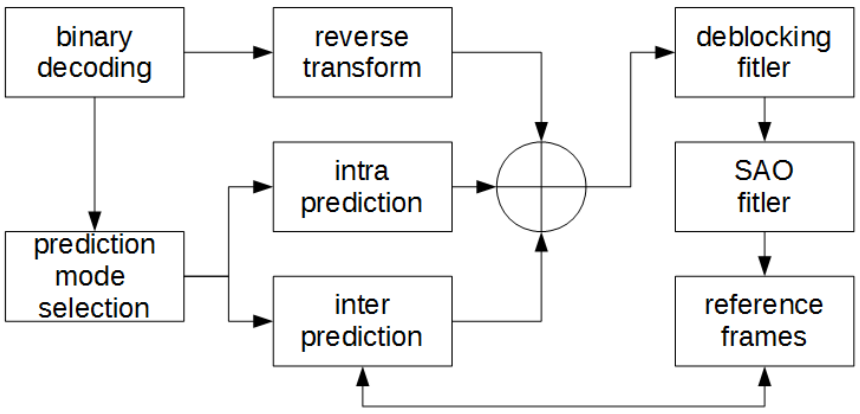


**Fig. 2.** HEVC codec flowchart

Beside mentioned frame reconstruction algorithms, both VP9 and HEVC utilize context-adaptive binary arithmetical coding which is much more complex in comparison to Huffman variable-length coding. At high visual quality, namely the arithmetical coding is the most time-consuming stage. Its algorithm is shown on fig. 3.

To allow modern high-efficiency image compression for images captured by UAV, one shall solve task of performance optimization for hybrid block-based video codecs. Because arithmetical coding is the most time-consuming stage, extra attention should be spent on it.

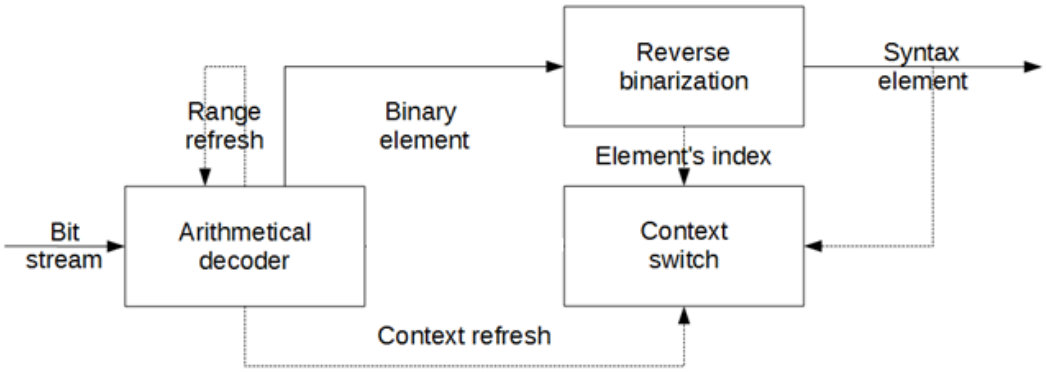


**Fig. 3.** Arithmetical encoder flowchart

**Aim of this paper** is to increase arithmetical decoder performance for mobile CPUs such as ARM processors. VP9 arithmetical encoder is shown for example; yet proposed approach can be used for any similar algorithm. High-efficiency codecs adoption for compression of aerial photography images allows for bigger amount of data stored during the UAV flight and increase both visual quality and resolution of photographs.

**The main part**. One of the main stages of modern video codec is the binarization of bit stream syntax elements. This stage is characterized by the fact that it cannot be vectorized or parallelized; yet statistical optimization approaches can be leveraged. The main approach to algorithm complexity estimation is the estimation of mean input complexity. Best- worst- and mean- case algorithm complexities are distinguished. $\sqsupset$ $x$- be the input of algorithm $A$, and $y$ be the output. Let us denote algorithm's time and space and space complexity functions as

$$T_A(n) = \max_{\|x\|=n} C_A^T(x)$$

$$S_A(n) = \max_{\|x\|=n} C_A^S(x)$$

Where $C_A^T(x)$ is time cost functions and $C_A^S(x)$ is the space cost function. Let us define finite array of algorithm input of size $n$:

$$X_n = \{x: \|x\| = n\}$$

For every $\forall x \in X_n$ there is a probability:

$$P_n(x) \in [0,1]: \sum_{x \in X_n} P_n(x) = 1$$

The mean complexity is determined as expected value:

$$T_A = \sum_{x \in X_n} P_n(x) C_A^T(x)$$
$$S_A = \sum_{x \in X_n} P_n(x) C_A^S(x)$$

This approach is classical and it is well described in prior art [8, 9]. One shall notice that there are situations when mentioned approach does not show good results on practice due to following reasons:

1. Big difference in number of algorithm steps and number of processor instructions required to do those steps. E. g. on commodity CPUs multiplication and addition is done within single instruction while floating-point division is done in tens of instruction
2. Peculiarities of memory system implementation in hardware. Multi-level memory hierarchy is used in modern CPUs and GPUs. Calls to different memory levels take an order of magnitude different time.
3. Optimizing compilers and hardware schedulers. During the binary files compilation, compilers are significantly changing the binary code and schedulers are re-ordering the instruction while keeping the program's finite automata the same. Memory cache controllers are performing reads in batches to speed-up IO operations.
4. For software implementations of algorithms, different programs and components are influencing each other. Therefore, task scheduler divide processor's time between tasks and parallel processes composed of multiple threads may end up being executed on variable number of physical processor cores.

Proposed modification of existing algorithm complexity analysis serves as theoretical addition to practical performance measurement means such as profiling and instrumentation. Method for algorithm input partitioning depending on complexity criteria was first covered in [10]. Suppose algorithm $A$ variety of all possible input data

$$G: \{g_1, g_2 \dots\}$$

All possible samples from $G$, distinct by size and content

$$g_i: \{g_i^1, g_i^2 \dots\}$$

In addition, array of complexity criteria for algorithm implementation (such as number of CPU instructions, runtime, etc.).

$$\propto_i : g_i \to \mathbb{R}$$

Suppose array of different complexity criteria

$$\mathrm{A}: \{\propto_1, \propto_2 \dots\}$$

This array has particular properties:

1. $\forall \propto_1, \propto_2 \in \mathrm{A}: \propto_1 \neq \propto_2$ – all elements of A are distinct.
2. $\forall \propto_i \in \mathrm{A}$ dice up $G$ onto multiple classes of complexity criteria equivalence:
$$G(\alpha_i) = \{g_i^1 \cap g_i^2 \dots\}$$
3. All samples of $G(\alpha_i)$ have the same complexity:
$$\sqsupset \alpha_i \in \mathrm{A}, g_i^k \in G(\alpha_i), \alpha_i : g_i^k \to r_i, k \in [0, \|G(\alpha_i)\|], r_i \in \mathbb{R}$$

Since all the elements of A are distinct we can always reorder it in a way that complexity function is non-decreasing for all criteria array elements. Expected complexity is similar to algorithm

expected mean complexity for discrete and continuous complexity probability. Below is the formula for discrete case:

$$R(\mathrm{A}) = \sum_{\alpha_i \in \mathrm{A}} r_i \, p_i$$

For continuous case:

$$R(\mathrm{A}) = \int_{\mathrm{A}} r \, dF(r)$$

Now let us apply proposed method to arithmetical coder complexity analysis. The process of entropy compression [11, 12] can be roughly divided by following steps:

– Binarization or transformation of symbol being coded (syntax element) into binary string composed of zeros and ones.

– Compression context modeling for syntax elements compression. For those elements, which statistical distribution is close to normal, this stage is skipped, and they are coded in bypass mode.

– Binary string arithmetical coding.

Delving into VP9 arithmetical sub-exponential coding of syntax elements [13] its algorithm may be represented in two steps. On first step, variables $b$ and $u$ are calculated as follows:

$$b = \begin{cases} k : n < 2^k \\ \lfloor \log_2 n \rfloor : n \geq 2^k \end{cases}$$

$$u = \begin{cases} 0 : n < 2^k \\ b - k + 1 : n \geq 2^k \end{cases}$$

Where k is sub-exponential parametric value, for VP9 video codec it is always equal to 4. On second strep, unary code of $u(u+1)$ bit is complemented with lower bits $n$. Length of code is equal to:

$$u + 1 + n = \begin{cases} k + 1 : n < 2^k \\ 2\lfloor \log_2 n \rfloor - k + 2 : n \geq 2^k \end{cases}$$

Therefore, literal decoding comes down to decoding its bits in a loop. For performance optimization of this algorithm, it is important to know probability distribution for literals length. Since those literals of the biggest length (such as discrete transform coefficients) are coded in series, there is a big chance that probability distribution for literal length will be biased with big number of elements having the same length. To check this hypothesis, experimental data was collected for literal lengths for aquatory images of Azov Sea (table. 1).

Table 1

| Literal length, bits | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Probability,% | 0.94 | 0 | 67.35 | 18.25 | 0 | 13.46 |

3, 4 and 6-bit long literals have the highest probabilities. Another important fact is that maximal possible literal length is 6 bits only. This is important for software implementation scenarios. During the real decoder optimization, we are most interested in run-time complexity criteria. In order to obtain array of complexities $R : \{r_0, \ldots, r_4\}$ we will do the program profiling. Array of unique

elements *R* will make up the array of run-time complexity criteria array A. Based upon an obtained data, following approaches were used for optimization:

1. Keeping the results of literal length calculation for series of literals of same length.
2. Sub-exponential literal decoding loop unwinding.
3. More efficient algorithm for calculation of number of bits for a literal.
4. More efficient CPU registers usage within an arithmetical coding function.

Implementation of points 1 and 4 is obvious so we will look at points 2 and 3 closer. Within the arithmetical decoder function, literal bits are decoded from compressed bit-stream. In our case, loop with variable number of iterations is the bottleneck. It can be replaced with so-called "Duff's Device" which is the switch-case construction without the break clauses. It allows for substitution of multiple iterations without the necessity for conditional execution. In addition, bit-shift is done for constant amount of bits, which does not depend on loop counter.

Listing 1: Original literal decoding function

```
static int vp9_read_literal(vp9_reader *br, int bits)
{
    int z = 0, bit;
    for (bit = bits - 1; bit >= 0; bit --)
        z |= vp9_read_bit(br) << bit;
    return z;
}
```

Listing 2: Modified literal decoding function

```
static int vp9_read_literal(vp9_reader *br, int bits) {
    register int z = 0;
    switch(bits -1){
        case 6: z |= vp9_read(br, 128) << 6;
        case 5: z |= vp9_read(br, 128) << 5;
        case 4: z |= vp9_read(br, 128) << 4;
        case 3: z |= vp9_read(br, 128) << 3;
        case 2: z |= vp9_read(br, 128) << 2;
        case 1: z |= vp9_read(br, 128) << 1;
        case 0: z |= vp9_read(br, 128);
        break;
    }
    return z;
}
```

One more bottleneck is the calculation of amount of bits for a literal done within the while loop [14]. This approach has a drawback of lack of knowledge of loop iteration in the run-time. Fast bits number calculation algorithm was use instead [15, 16] which does the calculation in the constant time without conditional expressions.

Listing 3: Fast calculation for amount of bits in literal

```
unsigned int v; // 32 bit argument
register unsigned int r; // variable number of bits
register unsigned int shift;

r = (v > 0xFFFF) << 4;
v >>= r;
shift = (v > 0xFF) << 3;
v >>= shift;
r |= shift;
shift = (v > 0xF) << 2;
v >>= shift;
r |= shift;
shift = (v > 0x3) << 1;
v >>= shift;
r |= shift;
r |= (v >> 1);
```

For the run-time measurement, a series of launches was performed for both reference and modified codecs. Modified codec shows 5.21% over-all lower run-time, which means arithmetical decoder alone, is 7.33% faster.

**Conclusion**. In present paper, arithmetical decoder optimization was done as a part of VP9 video codec. To solve a given problem, a modification to existing algorithm complexity analysis method was proposed which is based on partitioning of algorithm's input into set of equivalence classes for complexity. Proposed method allows predicting the number of steps for algorithms which complexity does not depend on input size or which is difficult to measure, that is typical for context-adaptive arithmetical coding. Results obtained in current paper allows to adopt novel high-performance image codecs yielding 25-34% better images compression according to SSIM metric and speed up the arithmetical coder by the 7%.

**References**

1. WebP Compression Study // https://developers.google.com/speed/webp/docs/webp_study.
2. T. Nguyen and D. Marpe, "Objective Performance Evaluation of the HEVC Main Still Picture Profile," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 25, no. 5, pp. 790-797, May 2015.
3. Arzumanyan R. V. Sukhinov A. I. Research on high-performance Google VP9 codec software implementation // Software systems and computation methods. — 2016. — vol. 1, № 2. — pp. 184–200.
4. Blahut, R. (2010). Fast Algorithms for Signal Processing. Cambridge: Cambridge University Press.

5. G. K. Wallace, "The JPEG still picture compression standard," in IEEE Transactions on Consumer Electronics, vol. 38, no. 1, pp. xviii-xxxiv, Feb. 1992.

6. Dvorkovich A.V., Dvorkovich V.P. Digital video-informational systems (theory and practice) // Technosphera. – 2012. -1009p.

7. Asaduzzaman, A.; Suryanarayana, V.R.; Rahman, M. Performance-power analysis of H.265/HEVC and H.264/AVC running on multicore cache systems // Intelligent Signal Processing and Communications Systems. 2013. - C. 174-179.

8. Sedgewick Robert, Wayne Kevin. Algorithms (Fourth edition). — Addison-Wesley, 2016.

9. Introduction to Algorithms / Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, Charles E. Leiserson. — 2nd edition. — McGraw-Hill Higher Education, 2001.

10. Welch William J. Algorithmic complexity: three NP-hard problems in computational statistics // Journal of Statistical Computation and Simulation. — 1982. — Vol. 15, no. 1. — P. 17–25. — URL: http://www.tandfonline.com/ doi/abs/10.1080/00949658208810560.

11. High efficiency video coding[Электронный ресурс] // sist.sysu.edu.cn.

12. 121. Sze V., Budagavi M. Parallelization of CABAC transform coefficient coding for HEVC // Picture Coding Symposium (PCS), 2012. — 2012. — May. — Pp. 509– 512.

13. Handbook of data compression / D. Grois, D. Marpe, A. Mulayoff et al. — Springer, 2010. — C. 1198.

14. Bit Twiddling Hacks [Электронный ресурс] // Stanford University.

15. Gervich L. R., Steinberg B. Y. Exaflops systems programming // Otktrytie systemi. - 2013. - №8.

16. Waren G. S.-jr. Algorithmic tricks for programmers. – 2nd ed. – M.: Williams, 2013. - 512 p.

**Author:**

**Arzumanyan Roman Vadimovich,** PhD student of department of Intellectual and Multiprocessor Systems, Institute of Computer Technology and Information Security of Southern Federal University (Checkhova str. 22. Taganrog, Russian Federation)

УДК 519.683.4 10.23947/2587-8999-2018-2-2-133-143

# Модифицированный подход к оптимизации производительности арифметического декодера для сжатия аэрофотоснимков

**Р. В. Арзуманян****

Южный Федеральный Университет, Таганрог, Российская Федерация

Статья посвящена разработке быстрого программного алгоритма арифметического кодирования для задач сжатия цифровых изображений. Предметом данной работы является задача исследования зависимости сложности алгоритма арифметического кодера от различных критериев сложности помимо размера входа. Задачей работы является поиск тех составных частей алгоритма арифметического кодера, которые являются наиболее вычислительно сложными с последующей оптимизацией производительности их программной реализации. Целью данной работы является разработка быстрого алгоритма арифметического кодера в составе видеокодека для сжатия изображений без учёта межкадровой разницы для применения новых кодеков. Предложен новый метод нахождения сложности алгоритма в среднем и оптимизированный программный алгоритм арифметического кодера, проведено теоретическое исследование с последующим проведением вычислительного эксперимента, при этом использована выборка спутниковых снимков акватории Азовского моря. Увеличена производительность программной реализации арифметического кодера на примере видеокодека VP9. Скорость работы арифметического кодера увеличена на 7%. Полученные практические результаты позволяют увеличить скорость работы новейших алгоритмов сжатия цифровых фото- и видеоизображений и делают возможным их применение на мобильных вычислительных платформах, в том числе в составе бортовой электроники БПЛА. Теоретические результаты данной работы расширяют методы анализа сложности алгоритма в среднем для тех случаев, когда количество шагов алгоритма зависит не только от размеров входа, но также и от неизмеримых критериев, например, от схемы обращения к общей оперативной памяти со стороны параллельных процессоров.

**Ключевые слова:** арифметическое кодирование, оптимизация производительности, сжатие изображений, сложность алгоритма в среднем, видеокодек

**Авторы:**

**Арзуманян Роман Вадимович,** Южный Федеральный Университет (347922, РФ, г. Таганрог, ул. Чехова, д. 22), аспирант

** E-mail: roman.arzum@gmail.com.