PAPER • **OPEN ACCESS**

Performance comparison among popular implementations of H.264 encoders

To cite this article: H Y El-Arsh *et al* 2021 *IOP Conf. Ser.: Mater. Sci. Eng.* **1172** 012036

Performance comparison among popular implementations of H.264 encoders

H Y El-Arsh, A S Elliethy, A M Abdelaziz and H A Aly
Dept. of Computer Engineering, Military Technical College, Cairo, Egypt, 11766

E-mail: hassan.yakout@ymail.com

Abstract. Remote sensing videos captured by Unmanned Aerial Vehicle (UAV) air-born high-resolution cameras require an efficient compression scheme that preserves the details of the visual contents of the videos while reducing the total size of the data to be managed in real-time. This paper presents a detailed comparison between different open-source implementations for the H.264 video compression scheme. While the high-resolution videos allow analysts to extract more descriptive interpretations and draw more conclusive results, the increase in the consequent data size consumes more storage, resulting in more channel bandwidth, more power, and encounters an extra delay in transmission time. An efficient implementation of video compression can alleviate these large data size effects. In this paper, we analyze and compare the JM-encoder, the X264, the FFmpeg, and Cisco's OpenH264 open-source implementations in terms of compression efficiency, video quality, and computational load. Moreover, we present the rate-distortion curves in terms of PSNR as a quality metric against the bit-rate for a combination of 20 videos with various resolutions and dynamic contents. Albeit H.64 is superseded by H.265, till now H.264 is used in more than 65% of video coding applications. For example, YouTube only allows H.264 for live streaming.

1. Introduction

Since the last decade, high-resolution videos are required in the field of surveillance and reconnaissance, especially in association with UAV air-born drones. This is due to the fact that increasing video resolution enables more meaningful analyses to obtain more rigorous results. Accordingly, raising video resolution for the sake of obtaining precise results will consume more storage, more channel bandwidth, and more delay in transmission time. Subsequently, efficient video codecs gain more attention in the field of remote sensing, especially for limited-resources implementations such as UAV.

H.264, the most ubiquitous video codec, was standardized as the *Advanced Video Codec* (AVC) through [1]. Although superseded by H.265 [2], H.264 is still the most widely used video codec [3]. More than 65% of security monitoring devices support H.264 (powered by the included hardware chips) [3]. Also, for software, statistics show that 67% of network videos utilize H.264 technologies [3]. Moreover, according to YouTube answer number 2853702, the only allowed video codec for YouTube live streaming is H.264.

The H.264 video codec performs the video decoding with low-complexity. However, the corresponding encoder is complex because the building blocks of the codec require intensive computations, especially the motion estimation part. Specifically, the motion estimation process

consumes 60% to 80% of the total encoding resources [4] and this consumes processing and power resources. Hence, optimized hardware implementations are considered an excellent choice over software for implementing the H.264 encoder. Nevertheless, these implementations are not suitable for special applications where customized design of the codec is required, such as crypto-coding [5, 6] and steganography [7, 8, 9]. Crypto-coding is a class of techniques in which coding and encryption are done within the multimedia codec to reduce the time and complexity for both operations and providing extra selective encryption capabilities [10]. Steganography is the art and science of data hiding within an innocent-looking cover such as all multimedia types. However, the long time to market for the hardware implementations and their lack of flexibility when changes in design parameters are required, stand as barriers against the usage of the hardware implementations in these types of applications. Consequently, the software implementation of the codecs, especially the open-source, is a viable choice.

In this paper, we introduce a practical benchmarking for three open-source software implementations for H.264 video codec, which are: FFmpeg [11], X264 [12] and OpenH264 [13]. The benchmarking is conducted in terms of compression efficiency (measured by Rate-Distortion (RD) curves) and computational resources. Also, we use JM v19 H.264 encoder [14] as a reference implementation for H.264 encoder to evaluate the RD-curves of the former three codecs. Experimental results on 20 test video sequences with different resolutions and constructions show that OpenH264 is the only software encoder that achieves real-time performance. The reason for the real-time performance of OpenH264 implementation is the utilization of Single-Instruction-Multiple-Data (SIMD) extensions, which are available for most recent processors. These extensions allow parallel computations in vector representations to speed up the computations of the different operations of the codec.

This paper is organized as follows. Section 2 introduces a brief description of H.264 encoder. Section 3 explains the basic ideas for SIMD instruction sets and its dominant rule for amplifying software performance, especially video encoders. Section 4 discusses our experiments and explains their resulting outcomes. Section 5 highlights the conclusion of the paper.

2. Overview for H.264 encoder

In this section, we introduce a brief overview of H.264 encoder. For more technical details, readers can refer to [1, 15]. Fig.1 describes the general coding steps for H.264 video encoder [1, 15]. The motion estimation module (ME) estimates a motion vector (MV) for every macro-block (MB) in the current frame (I_n) by searching the best match for (MB) corresponding to minimum Sum of Absolute Differences (SAD) in a search window within the previously reconstructed frame (I_{n-1}), which is stored in the frame memory (FM) module. Then, the motion compensation module (MC) uses (MV) and (I_{n-1}) to reconstruct the predicted frame (P). Then, the prediction error (P_E) is calculated by subtracting (P) from (I_n). The (P_E) is then coded by transformation (T) then lossy-compressed by the quantization module (Q) to compose the residual data (R_D). R_D is fed to the inverse quantization module (Q^{-1}) followed by the inverse transform module (T^{-1}) to compose the compressed prediction error (P'_E), which is added to (P), then saved in FM. Additionally, R_D is also fed to the entropy coder (ENT) module to generate the final output bitstream (BS). Two types of entropy coding exist in H264: Context-adaptive binary arithmetic coding (CABAC) and Context-adaptive variable-length coding (CAVLC). Although CABAC is more complex than CAVLC, CABAC reduces the compressed stream size by more than 9% [15].

H.264 defines several groups of coding features, defined as *profiles*. Each profile defines the required decoder's features for a certain class of applications, which set limits to the encoding features. The most common profiles are the *Constrained-Baseline*, *Baseline*, *Main*, *Extended*, *High* and *Intra* profiles. The *Constrained-Baseline* profile is used by applications such as mobile videos and video conferencing running on limited computing resources platforms, in which the

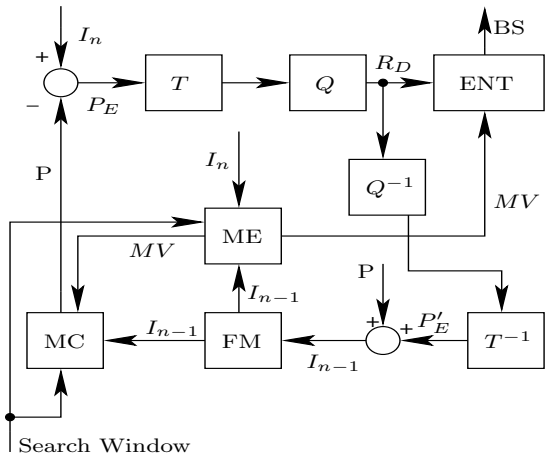


Figure 1. General description for H.264 encoder.

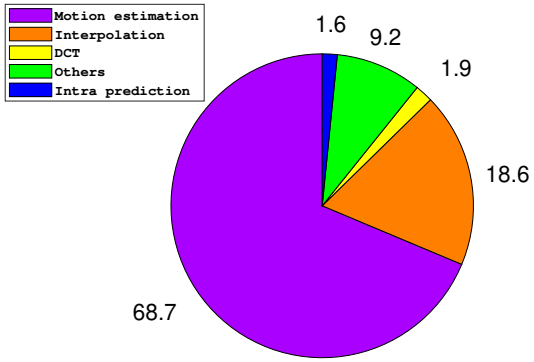


Figure 2. Profiling for H264 components [16].

error-resilience features are disabled. The *Baseline* profile has the same coding features as the *Constrained-Baseline* profile with additional error resilience features for applications that require robust data transmission. The *Main* profile is a superset for the *Baseline* profile, with additional complex features such as CABAC and B-frames [15], which make it suitable for SD TV broadcasting. The *Extended* profile is utilized for streaming applications, where switching frames features are enabled to allow extra error resilience performance. *High* profiles are used for HD TV broadcasting and HD video DVDs as they allow advanced coding features such as higher pixel precision and different chroma-subsampling schemes. *Intra* Profiles are subsets of *High* profiles without inter prediction (*i.e.* no P or B frames), used mainly for video editing applications. Other profiles exist for stereoscopic video coding.

H.264 also utilizes a term called *level* that defines the decoder performance, which depends on the amount of computational resources available for the decoder, and hence sets another limit for the encoder in accompaniment with profile number. Each *level number* specifies the maximum number of macro-blocks per frame, maximum decoding speed (macro-blocks/seconds) and the maximum video bit rate. For instance, level 1 limits are 99 macro-blocks/frame, 1485 macro-blocks/sec and video bitrate of 64 Kbits/s, while level 4 limits are 8192 macro-blocks/frame, 245760 macro-blocks/sec and video bitrate of 20 Mbits/s. Thus, for a video resolution of 1080p with 30 fps, a decoder with a minimum level number of 4 is required. This is due to that each frame with a resolution of 1080p contains 8100 macro-blocks and with frame rate of 30 fps, thus the minimum coding speed is 243000 macro-blocks/sec.

3. Video coding optimization with SIMD

In this section, we will explain how SIMD instruction sets can allow software-based encoder's implementations to achieve superior performance over traditional implementation techniques.

In recent years, CPUs' clock speed did not considerably increase, as it has reached some physical and power limitations. But as computational requirements still increasing over time, parallelization techniques are the only available solutions. By increasing the number of cores, the operating system can distribute its computational load among different cores and running programs can construct multiple threads to maximize usage efficiency. Another solution is by adding vector operation capabilities to each core, thus allowing the CPU to perform the same instructions, but for a vector of data.

SIMD instruction-sets extension is an example of vector processing akin to specialized Digital

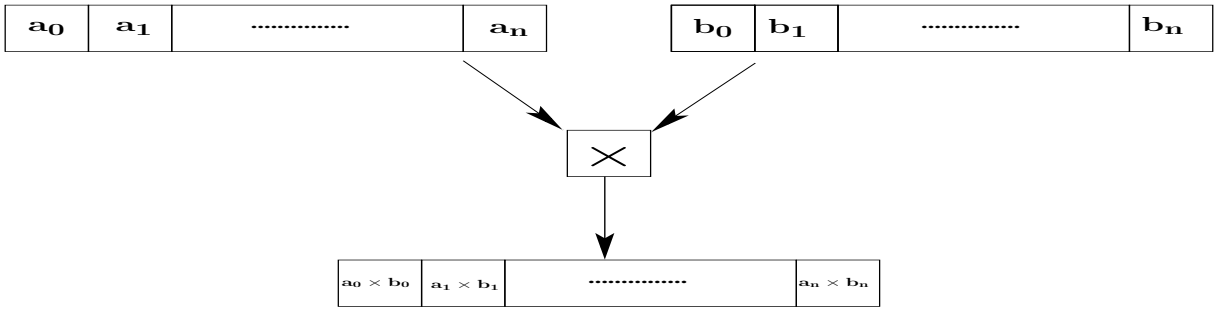


Figure 3. General description for SIMD operation (multiply).

Signal Processor (DSP). SIMD has become an essential technology for many modern CPU architectures. Intel MMX/SSE/AVX instructions for x86 architecture and NEON for ARM are all examples of SIMD instruction [17]. SIMD uses only one instruction to compute multiple data organized in a vector format, performing an operation in a single work cycle that was traditionally requires multiple working cycles, and hence can increase the performance for an application up to 11.5x [17]. Fig.3 illustrates the main idea of SIMD. Vector length for SIMD varies from 64-bits for MMX, 128-bits for SSE, 256-bits for AVX/AVX2 and 512-bits for the latest AVX-512 [18]. For ARM, NEON vector length is 128-bits [19]. Moreover, besides these notable performance enhancements, energy consumption is also reduced as it eliminates the number of instructions for fetch and decode processes. Many applications can be enhanced by SIMD instructions, such as machine learning, virtual reality and DSP applications especially video encoding, which is the main focus of this paper.

As described in fig.2, motion estimation algorithms consume 60% to 80% of the total encoding resources and both DCT and 1/4 pixel interpolation consume more than 20%. As SAD is the main atomic operation for the motion estimation process, optimizing SAD with "**_mm256_sad_epu8**" [20] for 32-bytes in a single cycle speeds up the calculations by 63x. This is due to calculating SAD for two 32-bytes arrays requires 32 subtraction operations and 31 addition operations. The convolution multiply and add (MullAdd) is the main atomic operation for both DCT and 1/4 pixel interpolation, utilizing "**_mm256_maddubs_epi16**" [20] for 16 signed 16-bit integers speeds up the calculations by 31x. Table 1 illustrates this comparison, where:

- *CPI* (Cycles Per Instruction): Measure of throughput by the number of CPU cycles required to perform single instruction, through which the part of CPU performing this instruction cannot execute another instruction [21].
- *Latency*: Number of CPU cycles from the start of the instruction until its result is available; usually considered only when this instruction is a part of a loop dependency chain [21].

Table 1. Performance evaluation for AVX2-intrinsic for Intel®Core i5-5200U in terms of CPI (Cycles Per Instruction) for throughput and latency [20].

Purpose	C/C++ Function Name	Assembly Instruction Name	CPI	Latency	Speed up factor
SAD for motion estimation	**_mm256_sad_epu8**	**vpsadbw**	1	5	63x
Convolution for DCT & interpolation	**_mm256_maddubs_epi16**	**vpmaddubsw**	1	5	31x

Table 2. Measured FPS for each encoder per each video.

Encoder \ Video file	Fig.4(a)	Fig.4(b)	Fig.4(c)	Fig.4(d)	Fig.4(e)	Fig.4(f)	Fig.4(g)	Fig.4(h)	Fig.4(i)
Openh264	44.12	62.856	73.509	102.041	108.828	155.801	152.127	183.199	245.976
X264	4.101	4.509	5.181	4.174	6.818	11.48	16.51	20.728	43.036
FFmpeg	4.114	4.518	4.958	3.84	6.166	10.068	15.137	20.128	30.595

4. Experimental Results

All experiments had been carried out with Intel®Core i5-5200U CPU @2.20GHz, except JM-encoder coding experiments were performed on higher performance PC, as JM-encoder is a single thread, implemented with pure C without any optimization, aiming to be a reference for compliance with the standard described in [1], in contrast to other implementations, especially OpenH264, with its optimized implementation utilizing SIMD instruction sets for CPUs such as SSE, AVX, MMX and NEON [13]. Our testbench environment is Ubuntu 18.04 x64 OS. We have utilized a combination of 20 HD, SD and low resolution online videos with various construction schemes from video datasets utilized in [22, 23]. Due to space limitations, we only demonstrate a sample of 9 videos representing different types of resolutions and construction schemes, as shown in table 3. All videos are coded with the *Main* profile with *level* 4.1 and CABAC enabled. It should be noted that although OpenH264 supports the *Main* profile within its "Layer Configuration File", and we have verified the output H264 stream from the encoder using both *FFmpeg* and *Mediainfo* utilities, the support for the *Main* profile is not included in the OpenH264 documentations.

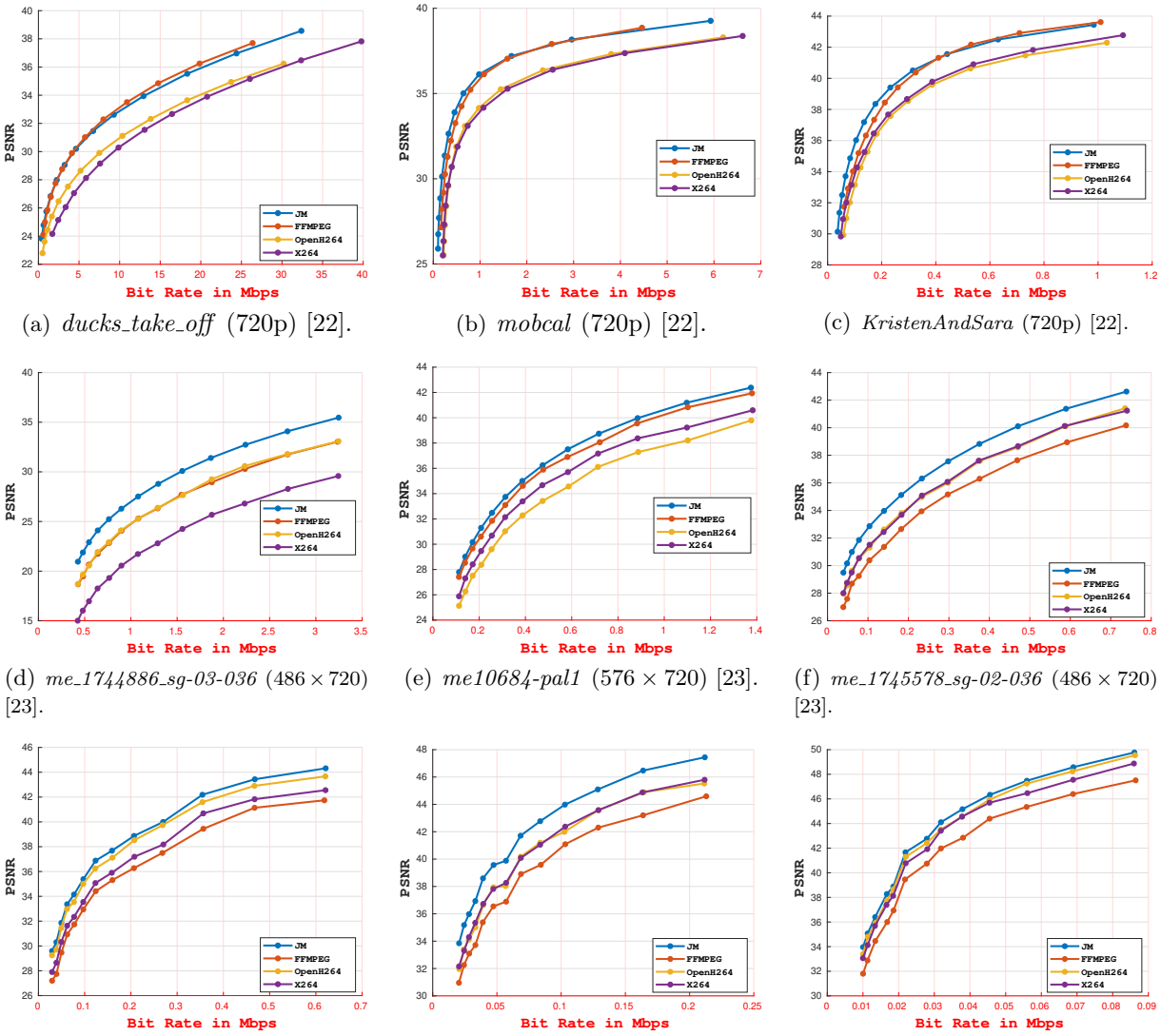
Fig.4 illustrates the RD-curves with PSNR metric in dB for our selected 9 videos. It can be concluded from fig.4 that JM-encoder achieves higher RD performance than other codecs, as it allows Full-search technique as ME algorithm, in contrast with other codecs which implement more optimized ME algorithms such as 3-step search, 4-step-search [24] and diamond-search [25] algorithms. Moreover, fig.4 verifies that RD performance is approximately the same for FFmpeg, X264 and OpenH264.

For performance measurements, we have utilized the built-in Linux "*top*" utility for calculating CPU utilization for each coding operation per each video's bitrate and for OpenH264, we set the *UsageType* as camera video. Also, the Linux built-in "*taskset*" tool is used to dedicate a single logical CPU core for each coding operation. Table 2 illustrates the average frame per second (FPS) for each encoder per each video under the previously mentioned conditions. It can be concluded from table 2 that OpenH264 is the only encoder that achieves real-time constraints among other tested video encoders, as it achieves an encoding rate above 25 FPS for all selected 9 videos, as well as for all 20 video combination. This is due to its implementation utilizing SIMD extended instruction sets such as AVX/AVX2, MMX, SSE and NEON, discussed in section 3.

5. Conclusion and future work

Through this paper, we have introduced a detailed explanation for our practical video encoding experiments for the most common open-source implementations for H.264 video encoder. Our experiments show that OpenH264 encoder is the only software encoder that achieves real-time constraints with acceptable resources margins. Moreover, the reasons which led to these results are clearly explained. These practical results can be used as a base for future implementations for specially customized video usages, such as cryptocoding or steganography, utilized for streaming

and broadcasting and other real-time-constrained applications.



(a) *ducks_take_off* (720p) [22]. (b) *mobcal* (720p) [22]. (c) *KristenAndSara* (720p) [22].

(d) *me_1744886_sg-03-036* (486 × 720) [23]. (e) *me10684-pal1* (576 × 720) [23]. (f) *me_1745578_sg-02-036* (486 × 720) [23].

(g) *me_583335* (360 × 640) [23]. (h) *me_583380_muscle* (360 × 640) [23]. (i) *me_587716_birds* (360×640) [23].

Figure 4. RD curves for the 9 selected videos representing different dynamics and resolutions.

Table 3. Resolutions and dynamic contents for test video sequences in fig.4.

Resolution \ Dynamics	High dynamics with complex motions	Moderate dynamics	Low dynamics with simple motions
HD	Fig.4(a)	Fig.4(b)	Fig.4(c)
SD	Fig.4(d)	Fig.4(e)	Fig.4(f)
Low resolution	Fig.4(g)	Fig.4(h)	Fig.4(i)

References

[1] ITU-T 2013 *Advanced video coding for generic audiovisual services* E 38445 (www.itu.int)
[2] ITU-T 2016 *High efficiency video coding* E 41298 (www.itu.int)
[3] Jiao S, Luan L, Qu H and Zhang M 2019 *2019 IEEE 19th International Conference on Communication Technology (ICCT)* pp 510–513
[4] Chen Z, Zhou P and He Y 2002 *Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG,* 6^{th} *meeting: Awaji Island, JP* **JVT-F017rl**
[5] Neji N, Jridi M, Alfalou A and Masmoudi N 2013 *10th International Multi-Conferences on Systems, Signals Devices (SSD13)* pp 1–4
[6] Boyadjis B, Bergeron C, Pesquet-Popescu B and Dufaux F 2017 *IEEE Transactions on Circuits and Systems for Video Technology* **27** 892–906
[7] Aly H A 2011 *IEEE Transactions on Information Forensics and Security* **6** 14–18 ISSN 1556-6013
[8] Tew Y and Wong K 2014 *IEEE Transactions on Circuits and Systems for Video Technology* **24** 305–319 ISSN 1051-8215
[9] Kapotas S K and Skodras A N 2009 *Journal of Real-Time Image Processing* **4** 33–41 ISSN 1861-8219
[10] El-Arsh H Y and Mohasseb Y Z 2013 *MILCOM 2013 - IEEE Military Communications Conference* pp 1844–1849
[11] Tomar S 2006 *Linux J.* **2006** 10 ISSN 1075-3583
[12] Shamieh F and Wang X 2019 *IEEE Transactions on Multimedia* **21** 1893–1904
[13] Roux L and Gouaillard A 2020 *Principles, Systems and Applications of IP Telecommunications (IPTComm)* pp 1–8
[14] Jankar J R and Shah S K 2017 *International Conference on Intelligent Sustainable Systems (ICISS)* pp 250–255
[15] Richardson I E 2010 *The H.264 Advanced Video Compression Standard* 2^{nd} ed (Wiley) ISBN 978-0470516928
[16] Shengfa Y, Zhenping C and Zhaowen Z 2006 *2006 International Conference on Communications, Circuits and Systems* vol 1 pp 126–129
[17] Fu S, Wu J and Hsu W 2015 Improving simd code generation in qemu *2015 Design, Automation Test in Europe Conference Exhibition (DATE)* pp 1233–1236
[18] Rucci E, Garcia Sanchez C, Botella Juan G, Giusti A D, Naiouf M and Prieto-Matias M 2019 *International Journal of Parallel Programming* **47** 296–316 ISSN 1573-7640
[19] Seo H, Liu Z, Großschädl J, Choi J and Kim H 2015 *Information Security and Cryptology - ICISC 2014* ed Lee J and Kim J (Cham: Springer International Publishing) pp 328–342 ISBN 978-3-319-15943-0
[20] Kusswurm D 2018 *Modern X86 Assembly Language Programming: Covers x86 64-bit, AVX, AVX2, and AVX-512* 2^{nd} ed (Apress) ISBN 978-1484240625
[21] Liu H H 2009 *Software Performance and Scalability: A Quantitative Approach* (Wiley-Blackwell) ISBN 978-0-470-46253-9
[22] Gryzov G Y and Dvorkovich A V 2017 *2017 6th Mediterranean Conference on Embedded Computing (MECO)* pp 1–4
[23] Masanovic B, Bavcevic T and Prskalo I 2019 *Montenegrin Journal of Sports Science and Medicine* **8** 69
[24] Po L M and Ma W C 1996 *IEEE Transactions on Circuits and Systems for Video Technology* **6** 313–317 ISSN 1051-8215
[25] Zhu S and Ma K K 2000 *IEEE Transactions on Image Processing* **9** 287–290 ISSN 1057-7149