FACULTY OF AUTOMATIC CONTROL, ELECTRONICS AND COMPUTER SCIENCE

PROGRAMME: CONTROL, ELECTRONIC AND INFORMATION ENGINEERING

Master Thesis

Improving the efficiency of lossless image compression using extensions of Part 2 of the JPEG 2000 standard

Author: Szymon Zosgórnik, BEng

Supervisor: Roman Starosolski, DSc PhD

Gliwice, September 2021

Załącznik Nr 2 do Zarz. Nr 97/08/09

O ś w i a d c z e n i e
w sprawie udostępnienia pracy

Wyrażam zgodę/nie wyrażam zgody* na udostępnianie mojej pracy dyplomowej/rozprawy doktorskiej*.

Gliwice, dn. .....

.....
(czytelny podpis studenta)

.....
(poświadczenie wiarygodności podpisu przez Biuro Obsługi Studentów)

* niepotrzebne skreślić

Abstract

The objective of presented paper is to demonstrate JPEG 2000 Part 2 compliant solution responsible for reducing lossless image compression ratio. The chosen extensions from the Part 2 of JPEG 2000 standard include decomposition using discrete wavelet transform in a way which is not compliant with Part 1 of the mentioned before standard. Moreover, different pairs of low- and high-pass filters are used. The application features entropy estimation to reduce runtime of image processing. As far as the results are concerned, the mean improvement in terms of image compression between Part 1 and Part 2 compliant solutions turned out to be on average equal to 1.5% with the standard deviation of 1.305%. The reference results that were gathered using actual JPEG 2000 codec came in to be on average equal to 1.982%. The standard deviation equaled 1.305% in this particular case. Moreover, the multithreading allowed to greatly reduce runtime of the application.

Keywords: lossless image compression, image processing, JPEG 2000, discrete wavelet transform, entropy estimation, multithreading, modern C++

Contents

Chapter 1. Introduction

The usage of digital images is constantly growing across whole world. There are multiple types of applications where memory usage matters to the users. Image compression is a possible solution to this problem in some of these fields. For example it is mission critical component in medical and picture archiving and communication systems (PACSs) [25]. There are two major types of such compression. The one is lossy variant and the other one is lossless. Applying lossy methods to the image can result in the occurrence of compression artifacts. However, there are applications where such disadvantage is negligible, e.g. natural images and photographs processing in Internet day-to-day usage [5]. On the other hand lossless image compression does not produce such artefacts, sacrificing some performance and bitrates optimizations. It is employed in mentioned before medical systems. Images used for the sake of diagnostics can be taken as an example. In some countries there are regulations that forbid applying lossy compression to such images [25]. Moreover, the usage of lossless variant is more desired when there exists some uncertainty whether information contained in the image can be discarded. In these scenarios not using any variant of compression can be the only substitute of lossless one [25].

Taking into account mentioned before reasons, some compression algorithms have been introduced as ISO standards [25]. Some notable examples of such algorithms are PNG, JPEG and JPEG 2000 (often written as JP2). The latter was originally developed from 1997 to 2000 with the desire of expanding JPEG capabilities. The main feature of this standard is usage of discrete wavelet transform (DWT) instead of discrete cosine transform (DCT) which was introduced in the predecessor [6]. The other feature of JPEG 2000 is support for lossy and lossless compression. As described before, such compression is needed to be performed in mission critical systems such as medicine. Therefore, the JPEG 2000 standard is utilized in PACSs and Digital Imaging and Communications in Medicine DICOM standard [25]. This standard consist of 16 ISO parts which contain wide set of features. Some notable ones are core system coding and its extensions, motion images, testing and reference software [6].

The successor of JPEG standard improved several aspects over its predecessor. With the usage of its algorithms, e.g. DWT, it was possible to improve compres-

sion performance over JPEG. Moreover, there are other improved areas with even greater importance. The few examples of such features are scalability and editability [6]. The JPEG 2000 standard supports both very low and very high rates of the compression. It comes crucial in applications that require such flexibility. Another main advantage of this standard is the ability of effective handling large range of bit rates. It allows to reduce number of steps taken in processing certain images in comparison to JPEG. As an example, reducing the number of bits in some image below certain amount using JPEG standard compliant solution requires reducing the resolution of the input at first. Only after this procedure encoding of the image can be applied. The JPEG 2000 standard supplies adequate feature named multiresolution decomposition structure which makes such transformation transparent and one step only [6].

The standard way of performing discrete wavelet transform (DWT) in the JPEG 2000 compliant with Part 1 is to decompose the image into sub-bands using a pair of low- and high-pass filters. This decomposition is applied multiple times using higher DWT orders. The standard order which is used across whole industry is five [20] [15]. The Part 2 of the standard contains several types of extensions which can be applied to modify the encoding algorithm. For instance DWT can be modified in a way that makes decomposition of the image into sub-bands of different shapes possible. Moreover, the strict selection of the pair of filters imposed by Part 1 of the standard can be broken. However, the same pair has to be used for all sub-bands of the image [20]. The other type of applicable modification is skipping some steps of discrete wavelet transform (SS-DWT). It is usually beneficial for processing non-photographic and screen content images. With the help of described Part 2 compliant extensions to the JPEG 2000 standard it is possible to adaptively adjust the transform for a specified image to improve the compression ratio. The result of this operation can still be correctly decoded by every decompressor which is compatible with the Part 2 of the JPEG 2000 standard.

The objective of the thesis is to develop, implement and test several forms of heuristics which can determine the optimal transform in terms of compression ratio of the given image. Transform shall be compliant with the Part 2 of JPEG 2000. The heuristics shall be rather fast and use entropy as an estimation of the JPEG 2000 encoding. Moreover, they can be greedy and use trial and error approach to some extent. The implementation of the program shall be done in modern C++ to utilize such language capabilities as cross-platform threads. The main target of the application are multi-core CPU architectures. The result of the project work is a tool that quickly determines the transform for the specified image and invokes the JPEG

2000 encoder with selected transform. However, it is acceptable to achieve small time overhead in terms of the entire compression process. The resulting image shall come with the improvement of the lossless JPEG 2000 compression ratio.

At the beginning of this paper there is introduction to the domain problem of image processing and compression. Some methods of applying this kind of compression are described in chapter 1 Introduction. Moreover, objective and scope of the thesis are described there. In the chapter 2 Problem analysis theory connected to discrete wavelet transform is presented in details. There are examples related to calculation process of one and two dimensional DWT discussed. Moreover, introduction to JPEG 2000 technicalities including Part 1 and Part 2 is depicted in that particular chapter. Lastly, there are scientific paper research and description of existing solutions available. The next chapter, i.e. chapter 3 Subject of the thesis is strictly connected to the description of proposed solution and implementation details. There are presented code snippets of the most crucial parts of the application. Another one which is chapter 4 Experiments concentrates on research methodology and final results. Both image compression and time execution of described solution to the problem are presented. The last part is chapter 5 Summary which wraps up all results and makes some valuable conclusions. At the end there are appendices available such as technical documentation and list of used tables, listings, etc.

Chapter 2. Problem analysis

2.1 Discrete Wavelet Transform

2.1.1 One dimensional DWT

The linear convolution (filtering) of sequences $x(n)$ and $h(n)$ is defined as in equation 2.1:

$$y(n) = \sum_{m=-\infty}^{\infty} x(m)h(n-m) \tag{2.1}$$

The one dimensional discrete wavelet transform can be depicted as successive applications (convolutions) of one selected pair of high and low-pass filters. The output of such application is then followed by downsampling by the factor of two. For example, it can be achieved by discarding samples with odd indices after each of filtering operation. It is better visualized in the Figure 2.1 [20]. The pair of low and high-pass filters is known as analysis filter bank in the encoding process. In the signal decoding process it is featured as a synthesis filter bank. The decoding step requires using the inverse of discrete wavelet transform.

Take into consideration a one dimensional signal $x(n) = \{55, 234, 70, 21, 88, 37\}$. It can be better understood as values of pixels in a part of the grayscale image row. It is followed with a pair of low and high-pass filters designated by $h_0(n)$ and $h_1(n)$ respectively. An example of such pair is a lowpass filter $h_0(n) = \{-1, 2, 6, 2, -1\}/8$ and a high-pass filter $h_1(n) = \{-1, 2, -1\}/2$. They are both symmetric and consist of only

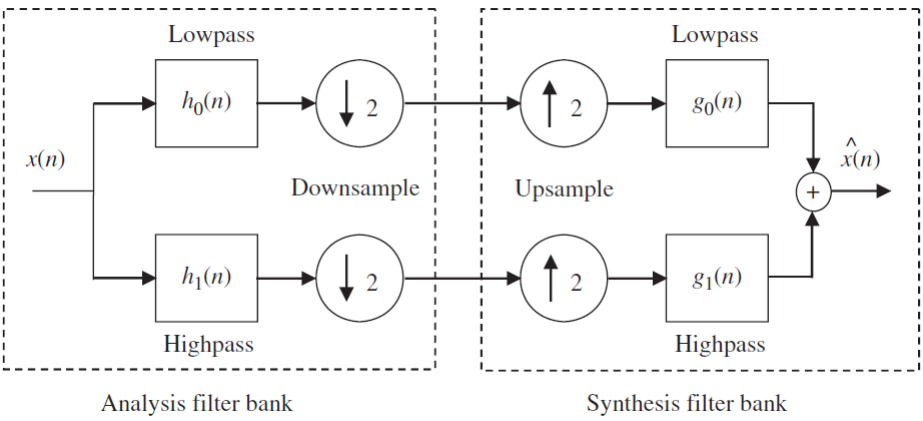

Figure 2.1: 1-D DWT, two-band wavelet analysis and synthesis filter banks [20]

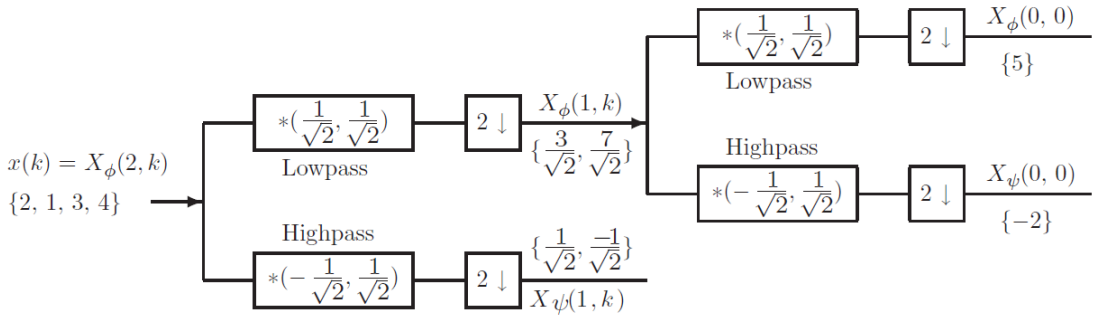

Figure 2.2: Computation of a 2-level 4-point DWT using a two-stage two-channel Haar analysis filter bank [29]

integer operations. Such pair can be presented in the notation of (5, 3) filter bank. This convention indicates that the length of lowpass filter is five and the length of high-pass filter is three. In fact the analysis filter bank presented here was firstly proposed by LeGall and Tabatabai in 1988 and is used in the JPEG 2000 standard for lossless compression of images. The filtering operation has to be defined at the signal boundaries. Therefore, the one dimensional signal is extended in both directions. The Part 1 of the JPEG 2000 standard requires symmetrical extension to be performed in such case [20]. After applying the required symmetrical padding the signal is extended to $x(n) = \{21, 70, 234, 55, 55, 234, 70, 21, 88, 37, 37, 88, 21, 70\}$. Then, the low-pass filter is applied resulting in $x'_0(n) = \{197.25, 75.5, 98.375, 67.125, 45.375\}$ and the high-pass one which results in $x'_1(n) = \{44.75, -85.75, 29, 12.75, -29.5\}$.

The next example shows how to compute the two levels of discrete wavelet transform. To speed up the process no padding option is chosen this time which makes it non-compliant with the JPEG 2000 standard. The filter used here is the most basic one, i.e. Haar analysis filter bank. It is the first wavelet from the Daubechies wavelet family. The calculation process is visualized in the Figure 2.2 [29].

The input is chosen as 4-point signal $X_\phi(2,k) = \{2, 1, 3, 4\}$. This notation emphasizes the fact that it is approximation of the input at scale 2. The so called scaling coefficients (or in other term approximation at scale 1) $X_\phi(1,k)$ are computed by convolving the input $x(k)$ with the low-pass Haar filter impulse response $l(k) = \{1/\sqrt{2}, 1/\sqrt{2}\}$. In the next step there is downsampling by a factor of 2 applied. The output of convolution has five values. The middle three from these fives correspond to cases where both the given input values overlap with the impulse response. As it was described earlier, the odd values are preserved in the downsampling process. In a result first and third value of these three middle ones are the approximation output $X_\phi(1,k)$. In the similar way, the detail coefficients at scale 1 $X_\psi(1,k)$ are computed. The input $x(k)$ is convolved with the high-pass filter impulse response $h(k) = \{-1/\sqrt{2}, 1/\sqrt{2}\}$. Then the downsampling by factor of 2 is

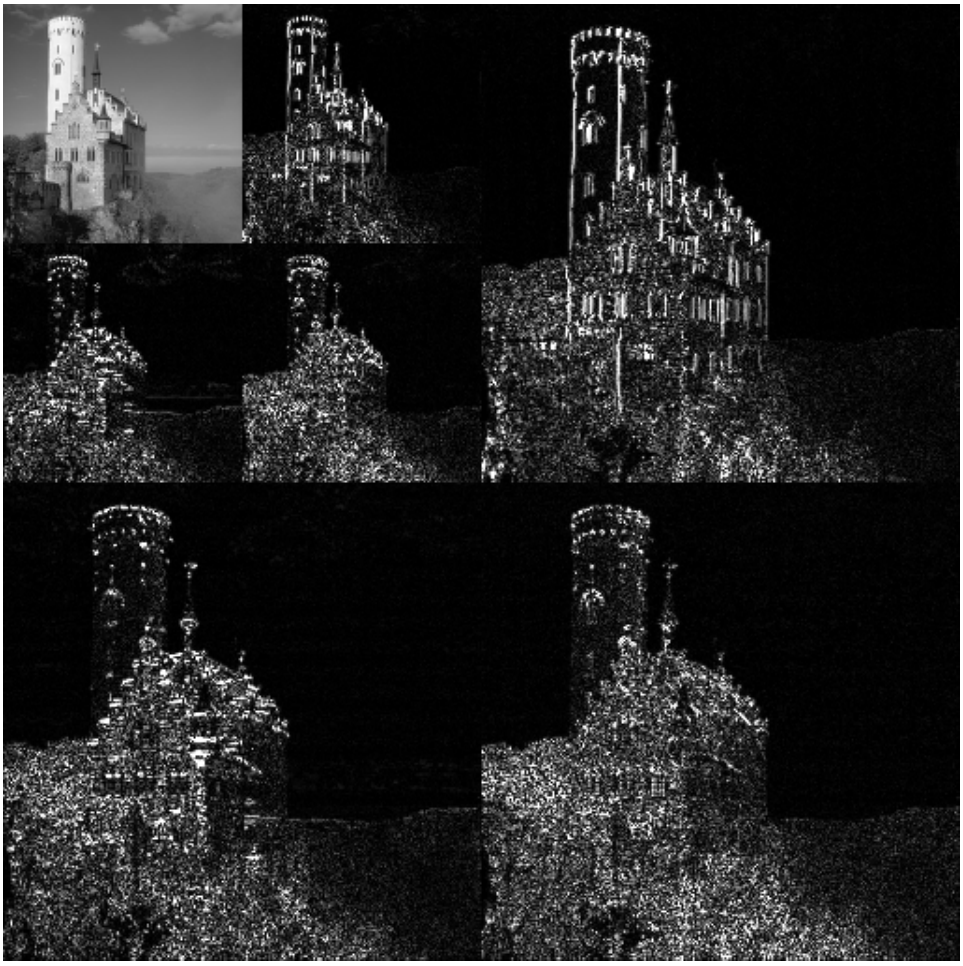

Figure 2.3: 2D DWT applied 2 times to an exemplary image [1]

performed. Note that only only approximation output $X_\phi(1,k)$ of the first stage goes to the second one. The $X_\phi(0,0)$ and $X_\psi(0,0)$ are calculated accordingly at the end of the second stage [29].

2.1.2 Two dimensional DWT

The idea of using lowpass filter is the preservation of low frequencies of a signal while trying to eliminate or at least attenuate the high frequencies. In a result the output signal is the blurred version of the original one. Therefore, the operating principle of the high-pass filter is completely opposite. As a result of applying such filter, the high frequencies of the signal are preserved and the low ones are discarded or at least diminished. The output is a signal consisting of edges, textures and other details [20].

There is presented an example of the effects of the two dimensional discrete wavelet transform on the Figure 2.3. The DWT used here is compliant with the Part 1 of the JPEG2000 standard. The number of DWT stages presented in this example

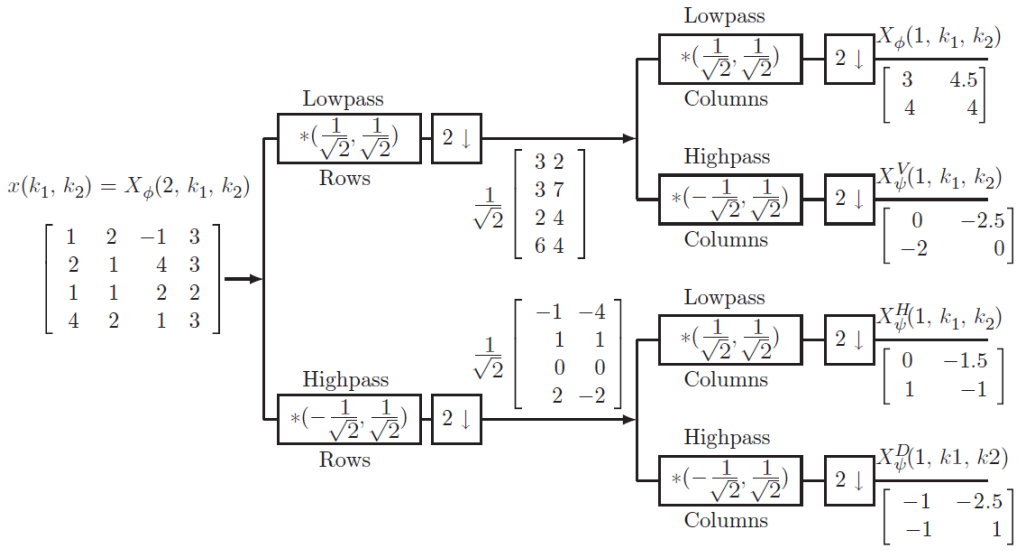

Figure 2.4: Computation of a 1-level 4 $\times$ 4 2-D Haar DWT using a two-stage filter bank [29]

is equal to two. Two dimensional discrete wavelet transform applied first time to the original image yields four same sized sub-images. The LL layer (upper left sub-image) is an approximation of the image and contains the low frequencies. This layer is once more transformed in the next stage. The LH layer (upper right sub-image) preserves high frequencies from the rows of the image. As a result vertical lines and details (brightness) can be seen in the produced sub-image. On the other hand, the HL layer (bottom left) contains high frequencies from the columns of the image. The horizontal details and lines can be noticed there. Lastly, the HH layer (bottom right) preserves the diagonal lines [1].

The process of computing a 1 level two dimensional discrete wavelet transform with usage of two-stage analysis Haar filter bank is shown in Figure 2.4. Coefficients $\mathbf{X}_\phi$ are calculated as a result of lowpass filtering and downsampling to each row of the two dimensional data. Next, similar process process, i.e. lowpass convolution and downsampling is applied to each column of resulting data. The rest of coefficients is obtained in very similar fashion to the previous ones. Coefficients $\mathbf{X}_\psi^H$ are calculated by applying high-pass filtering and downsampling to each row of the 2-D data $\mathbf{x}$ and then followed by applying sequence of low-pass filtering and downsampling to each column of the resulting data. Coefficients $\mathbf{X}_\psi^V$ are obtained by applying low-pass filtering and then downsampling to each column of the resulting data. Lastly, coefficients $\mathbf{X}_\psi^D$ are obtained by applying high-pass filtering and downsampling to each row of the 2-D data $\mathbf{x}$ followed by applying high-pass filtering and downsampling to each column of the resulting data. In the next stage of more com-

plex DWT calculating process only the coefficients $\mathbf{X}_\phi$ are taken into consideration [29].

2.1.3 DWT features summary

- In a nutshell, the discrete wavelet transform is a set of bandpass filters. Usually it is implemented with the usage of low and high-pass filters recursively.
- The computational complexity of computing the DWT in the best case is linear, i.e. $O(N)$.
- The first approach to implement the DWT efficiently is evaluation of the required convolutions with the usage of the polyphase filter structure.
- The second approach is factorization of the polyphase matrix into a product of a set of sparse matrices.
- The two dimensional discrete wavelet transform (with separable filters) is usually computed by the row-column method. One dimensional DWT of all the columns is computed at first. Then the 1-D DWT of all the resulting rows is calculated. The order of the computation in the row-column method can be swapped. The result remains the same.
- Additional memory of approximate half the size of the given data is required in the some implementations of the DWT. However, there also exist methods of computing DWT in-place which do not require additional memory.
- Data reordering is required for an in-place computation of the DWT.
- Data expansion problem can occur due to the finite length of the data in the implementation of the asymmetric filters.
- Symmetric filters provide linear phase response and an effective solution to the border problem [29].

2.2 Part 2 of the JPEG 2000

2.2.1 Introduction

Many ideas have been emerging as the JPEG 2000 was developed. These concept were full of value-added capabilities. However, they were not that important to be gone through the time-consuming ISO standardization process. The Part 1

(ISO/IEC, 2004a) of the standard, i.e. Core coding system, was originally published in 2000. There was a need to create additional parts to include missing features. The Part 2 of the standard, published as ISO/IEC 15444-2 or ITU Recommendation T.801 (ISO/IEC, 2004b), contains multiple such extensions. There is present group of rather small additions that could not merit entire documents of their own. In the Part 1 Core of JPEG 2000 standard decoders are supposed to handle all of the code-stream functionality. The Part 2 is different from first one in this aspect. It is a collection of options that can be implemented on demand to meet very specific requirements of the given market. Moreover, sections within an extension annex can be implemented separately. For example, subsets of extended file format JPX can be used on their own. Therefore, some features of the Part 2 may be present in the wide spectrum of JPEG 2000 applications while the other ones can be less common in the decoders [20].

As it was shown in the previous paragraph, the extensions present in the Part 2 consist of very different set of topics that can modify or add some features to the Part 1 JPEG 2000 compliant processing chain. Some tools can result in the compression efficiency improvement. Others can ameliorate the visual appearance of compressed images. Another group of extensions can modify or extend some functionalities in the other ways. The list of the major topics is presented below [20].

Compression efficiency:

- Variable DC offset (VDCO) - Annex B
- Variable scalar quantization (VSQ) - Annex C
- Trellis coded quantization (TCQ) - Annex D
- Extended visual masking - Annex E
- Arbitrary wavelet decomposition - Annex F
- Arbitrary wavelet transform kernel - Annexes G and H
- Multiple component transform - Annex J
- Nonlinear point transform - Annex K [20]

Functionalities:

- Geometric manipulation - Annex I

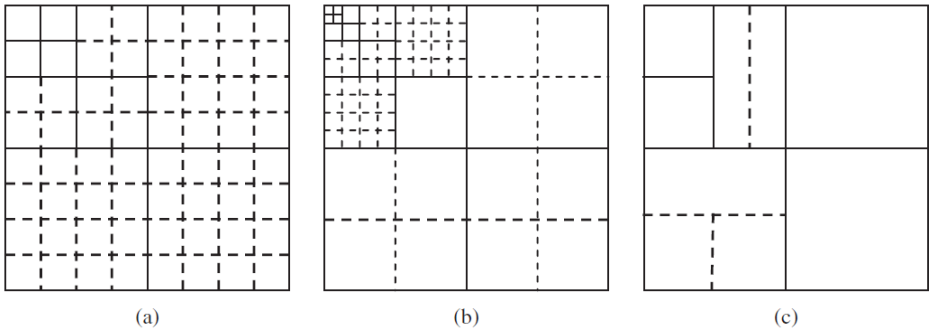

Figure 2.5: Some examples of decomposition compliant with the Part 2 [20]

- Single-sample overlap (SSO/TSSO) - Annex I
- Precinct-dependent quantization - Amendment 1
- Extended region of interest - Annex L
- Extended file format/metadata (JPX) - Annexes M and N
- Extended capabilities signaling - Amendment 2 [20]

2.2.2 Arbitrary Decomposition

In the Part 1 of the JPEG 2000 standard there is only one wavelet decomposition structure allowed. This wavelet is called Mallat dyadic decomposition. Such decomposition is a good first choice to be applied across a wide spectrum of images. However, other ones can improve the quality of the image over specialized classes of the applications. The other effect of applying such decompositions are unequal reductions in the horizontal and vertical dimensional of reduced resolution extracts [20].

Other decomposition styles can be found in the wavelet literature. They include the full packet tree processing and some of its derivatives. The applied packet decomposition derivatives can outperform the solution from Part 1 of the JPEG 2000 standard in some applications. For instance, they come crucial at maintaining regular fine-grain texture. Moreover, the applications that require processing synthetic aperture radar images can benefit from using this extension. The US Federal Bureau of Investigation actively uses a 500 ppi fingerprint compression standard, i.e. WSQ (CJIS, 1997). The decomposition is specialized for the characteristics of fingerprint imagery at 500 dpi [20].

Some of these decomposition can be seen of the Figure 2.5. Resolution decomposition is depicted as solid lines. Dashed lines represent extra sublevel decomposition. On the first example, i.e. image (a), there is available full packet

Table 2.1: Analysis and synthesis filter taps for the floating-point Daubechies (9, 7) filter bank

n	Low-pass, $h_0(n)$	Low-pass, $g_0(n)$
0	+0.602949018236360	+1.115087052457000
± 1	+0.266864118442875	+0.591271763114250
± 2	-0.078223266528990	-0.057543526228500
± 3	-0.016864118442875	-0.091271763114250
± 4	+0.026748757410810	

n	High-pass, $h_1(n)$
-1	+1.115087052457000
$-2, 0$	-0.591271763114250
$-3, 1$	-0.057543526228500
$-4, 2$	+0.091271763114250

n	High-pass, $g_1(n)$
1	+0.602949018236360
$0, 2$	-0.266864118442875
$-1, 3$	-0.078223266528990
$-2, 4$	+0.016864118442875
$-3, 5$	+0.026748757410810

decomposition with such parameters: NL = 3: Ddfs = 111, Doads = 321, Dsads = all 1s. The next picture illustrates FBI decomposition wit specified parameters: NL = 5: Ddfs = 11111, Doads = 2321, Dsads = 11101111111111111. The last image is juts an arbitrary example [20].

The prespecified decomposition structures are not the only feature of this extension. Wavelet packet analysis can be also used to design custom decompositions for specific images or some types of images. It was implemented in these papers [21], [18] and [16]. Such applications often start with a large decomposition tree. Then, they tend to locate a good decomposition based upon specified optimization metric [20].

2.2.3 Arbitrary Wavelet Transforms

The Part 1 of the JPEG 2000 standard specifies only two possible wavelet transforms. The reversible one (5-3R, Table 2.2) and the irreversible one (9-7I, Table 2.1). As it was stated before, both are required to perform periodic symmetric signal extension at the boundaries. It is similar case to the Mallat dyadic decomposition in terms of generic implementation. These filters can compress quite well a wide set of image types. However, certain image classes can be compressed more efficiently with other types of wavelets. Such a flexibility is allowed in the Part 2 compliant applications. The range of wavelet transforms is broadened to include not only the wider range of whole-sample symmetric ones but also half-sample and generic non-symmetric ones. Such ability to handle generic filters makes JPEG 2000 standard

Table 2.2: Analysis and synthesis filter taps for the integer (5, 3) filter bank

n	Low-pass, $h_0(n)$	Low-pass, $g_0(n)$
0	+0.75	+1
± 1	+0.25	+0.5
± 2	-0.125	

n	High-pass, $h_1(n)$
-1	+1
$-2, 0$	-0.5

n	High-pass, $g_1(n)$
1	+0.75
$0, 2$	-0.25
$-1, 3$	-0.125

a powerful research tool, together with supporting more than niche compression applications [20].

2.3 Parallelism in computer architecture

The parallelism as general term can be simply understood as two or more separate activities that appear to happen at the same time. It is not only computer science specific abstraction but also natural part of life. For instance a person can drive a car and simultaneously talk on the phone or one person can have a walk, while the other one is riding bike. However, the computer science aspect of parallelism is the interesting part in this paper. It can be depicted as single system performing multiple independent tasks at the same time rather than one after the other [30]. Multitasking operating systems have been in usage for many years. However, running multiple applications was firstly done only by context switching. The situation has been a little bit different in the server industry. High-end machines with multiple processors have been available there for years making utilization of the real parallelism possible. In recent years there is observed increase of personal computers capable of running multiple tasks in similar manner [30].

Nowadays, there is a trend in producing increasing amount of processors in multicore solutions by chip manufacturers. There can be found 16 or even more processor cores on a single chip. Following this way is easier for improving performance over strengthening single core solutions. Therefore, multicore desktop computers and even embedded devices are growing share in the market. In the past, it was possible for programmers to get their application run faster without doing anything. Such situation was linked with the growing computation power within new generation for single core solutions. The situation is quite different at the present time. If a

software has to take advantage of increasing computing power, one has to design and utilize concurrent run of multiple tasks [30].

There are two approaches of utilizing concurrency in terms of performance. The most obvious one is just to divide a single task into separate parts. Then, such subtasks can be run in parallel reducing the total execution time of the program. This approach is called “task parallelism”. Despite sounding rather straightforward, such operation can become a complex process. Especially whether there exist many dependencies between the separated parts. Such division can be applied in two different ways. The first one is connected with processing, i.e. one thread executes certain part of the given algorithm while the other ones perform operations at the different part. The other way is connected strictly with the data. Each thread performs the same operation on different parts of the given data. The latter approach is called data level parallelism [30].

There exists quite notable subgroup of algorithms which are basically ready to be parallelized. They are often referred as “embarrassingly parallel”, “naturally parallel” or “conveniently concurrent” [30]. Such algorithms are especially good in terms of scalability properties. As the number of available hardware resources, i.e. threads increases, the parallelism level in the algorithm is trivial to match. However, if there exist parts of the algorithm which are not easy to parallelize, one has to divide the algorithm into a fixed number of tasks. Therefore such application becomes not scalable one [30].

The another way to use parallelism for improving performance is to employ existing concurrent solutions to solve bigger issues. This approach can be depicted as processing 2, 5 or 10 files at given time instead of just one. Although it seems like another application of data level parallelism, there is different focus in performing such operation on sets of data concurrently. The amount of time needed to process one chunk of data is still the same. However, more data can be processed in the same amount of time. Unfortunately, there are limits on this specific approach as well. Moreover, there are cases where such attitude can be nonbeneficial at all. In the end, the increase in throughput which comes from this specific approach makes new things possible, e.g. increased resolution in video processing where different areas of the picture can be processed at the same time [30].

2.4 Known solutions

2.4.1 Part 1 compliant applications

There are multiple solutions that implement features from the Part 1 of the JPEG 2000 standard. The OpenJPEG library is an example of such application. It is open-source solution developed to promote wider usage of this standard. The main part of this project is the codec compliant with the Part 1 of the JPEG 2000 standard. Moreover, the OpenJPEG library integrates other parts of the standard [20]. Few of them can be seen on the list below:

- from the Part 2: handling the JP2 boxes and extended multiple component transforms for multi and hyperspectral imagery.
- from the Part 3: MJ2 (Motion JPEG 2000).
- from the Part 11: JPWL (JPEG 2000 Wireless).
- OPJViewer, a GUI based tool for visualization of the J2K (extension used for storing code-stream JPEG 2000 data), JP2 (standard JPEG 2000 file extension), JPWL and MJ2 files.
- OPJIndexer, a code-stream indexer to view information about the headers and packets localization. This tool is also capable of reading the rate-distortion contribution of each packet to the image [20].

The OpenJPEG library is written in the C programming language and released under the BSD license. The main targets of this software are desktop platforms, i.e. Win32, Unix and Mac OS platforms. The Communications and Remote Sensing Lab (TELE) of the Catholic University of Louvain (UCL) is the main developer group of this library. CS company and CNES is the supporting side. Some of the modules are maintained by the Digital Signal Processing Lab (DSPLab) of the University of Perugia, mainly the JPWL and OPJViewer. [20].

Another example of Part 1 compliant application is the JasPer. This computer software project is aimed to create reference implementation of the standard codec. The project was started by the Image Power Inc. and the University of British Columbia back in the 1997. As OpenJPEG JasPer is written in the C programming language. There are some sample applications available in the codebase. They can come in handy while testing the codec. JasPer is currently released publicly under the MIT license. The library is a component of several notable software projects.

It includes but is not limited to netpbm, ImageMagick and KDE. In series of the JPEG 2000 compression tests conducted in 2004 JasPer turned out to be the top performing solution, closely followed by IrfanView and Kakadu. The disadvantage of this implementation is its time performance. In the same tests JasPer was the slowest software. However, it is a feature as this codec was designed to be used as reference in non performance-critical systems.

There is one more notable implementation of the Part 1, i.e. Grok. It is open-sourced software released under the GNU Affero General Public License (AGPL) version 3 license. Its design aims to provide stable, high performant and low memory using solution. The function responsible for decoding process (grk_decompress) is currently over 0.5 the speed of the Kakadu software. Moreover, Grok supports fast sub-tile decoding to standard output for png, jpeg, bmp, pnm and raw formats. This library supports both TLM and PLT code stream markers for fast single-tile and sub-tile decoding of large tiled images. The support of meta-data formats such as XML, IPTC, XMP and ICC profiles is also included. There is also initial version of Part 15 implementation (High Throughput JPEG 2000) available. The final version of this solution should be ten times faster over the original Part 1 of the JPEG 2000 standard.

2.4.2 Kakadu

Kakadu is not only a complete implementation of the JPEG 2000 standard Part 1 but also an application that supports Part 2 and Part 3 in a significant amount of features. The software was originally developed by David Taubman of the University of New South Wales (UNSW) in Australia. The author is also noticeably known for being the designer of EBCOT, i.e. the core coding component of JPEG 2000. The name of this library comes from “Kakadu National Park” which is located in the Northern Territory of Australia [20]. Licensing is more advanced in comparison to solutions mentioned in subsection 2.4.1 Part 1 compliant applications. There are separate licensing schemes for research, commercial and demonstration-only applications.

The Kakadu software framework is widely adopted in the substantial range of JPEG 2000 products. The few examples are Apple’s Quicktime v6 for MAC, Yahoo’s Messenger, Google Earth and Internet Archive [20]. Features that can be realized thanks to Kakadu’s implementation by the named beforehand solutions include live video and products for geospatial imagery such as MicroImages TNT. Moreover, this framework is used in medical imaging applications, interactive image rendering applications, remote browsing of large collection and images, digital cinema appli-

cations and the other fields that require compression and decompression of JPEG 2000 images and videos [20].

Kakadu is considered as a comprehensive, heavily optimized and fully compliant software toolkit for JPEG 2000 developers. There are multiple features available that make runtime execution so flawless. Multithreaded processing is supported in such way that makes the most of parallel processing resources, i.e. multiple CPUS, multicore CPUs and hyperthreading [20]. Compiler intrinsic functions are utilized to manually vectorize processing of data in processor's pipeline. Moreover, Kakadu comes with built-in thread scheduler. Such implementation enables possibility of utilizing all computational resources close to 100%. In a 2007 the JasPer library was outperformed by Kakadu in terms of speed.

Supported features of Part 2 include arbitrary wavelet transform kernels and general multicomponent transforms. The Kakadu library offers extensive support for interactive mode of client–server applications. It is done thanks to the implementation of the most notable features from the JPIP (JPEG 2000 Internet Protocols) standard [20].

2.4.3 Reversible denoising and lifting steps with step skipping

Methods such as reversible denoising and lifting based color component transformation aim to improve overall quality of image compression. Color space transformation in reversible manner is required by the lossless image compression framework used in the JPEG 2000 standard. The undesirable side effect of such transformation is contamination of transformed components with noise from other ones. Therefore, the compression ratio of given image is substantially diminished [24].

The work in this specific paper is aimed to remove correlation without increasing the noise. Therefore, a reversible denoising and lifting step (RDLS) was proposed. The RDLS integrates usage of denoising filters into lifting step. New image component transformation is a result of applying RDLS to color space transformation. The main feature of this operation is being "reversible despite involving the inherently irreversible denoising" [24].

The main targets using application of RDLS to the RDgDb color space transformation with simple denoising filters are the JPEG-LS, JPEG 2000 and JPEG XR lossless modes of the standard algorithms. The images which have native optical resolution of acquisition devices benefit the most from this application. Improvement in terms of compression ratios is the most visible in subset of images which unmodified color space transformation either improved or worsened ratios in comparison to the untransformed image. On the average improvement is between 5% and 6%

for such specific images. However, things differ for images from “standard test-sets” resulting in the improvement only up to 2.2% [24].

Another application of the RDLS (reversible denoising and lifting step) is described in the paper [25]. New method of using “hybrid transform” was introduced there. The main input to that paper was discrete wavelet transform using custom prediction mechanism. Although simple prediction is considered to be ineffective when combined with DWT, the application of modified discrete wavelet transform using RLDS with step skipping is able to take advantages of its strengths. The usage of heuristics and estimation of entropy resulted in making described transform “image-adaptive” [25].

The data set used for experiments in the presented paper contained 247 non-photographic and 499 photographic images. As a result it was found that described approach using combination of RDLS with step skipping, DWT and prediction turned out to be effective. The prediction mechanism allowed to double the JPEG 2000 compression ratio improvements made by using only RDLS. Compression schemes with various tradeoffs were proposed due to fact that for some images applying prediction instead of DWT could be more beneficial. The compression ratios of non-photographic and photographic images improved by 30.9% and 1.2%, respectively, in comparison to solution compliant with Part 1 of the JPEG 2000 standard. The cost was in the increase of compression time by 2% and breaking mentioned before compliance. Greater ratio improvements were claimed to be possible with greater cost in runtime performance [25].

2.4.4 Skipping Selected Steps of DWT Computation

The other method of improving image compression, i.e. bitrates of lossless JPEG 2000 is skipping selected steps of discrete wavelet computation (SS-DWT). At the first phase of implementing this particular method fixed SS-DWT variants were employed. Then, heuristic was employed to select from the mentioned before variants best one for certain image [22].

The experiments on diverse set of images resulted in the improvement of bitrates vastly of non-photographic images. “The entropy estimation of encoding effects” was used to select the most feasible variant of applied fixed SS-DWT. It is especially important from the practical standpoint as time execution of such application is reduced. “Such compression scheme is compliant with the Part 2 of the JPEG 2000 standard as opposed to the general SS-DWT case” [22].

The average improvement in terms of bitrate was around 5% for the entire test-set. Moreover, the time needed to perform compression of the image was only 3%

great than the part 1 of the JPEG 2000 compliant variant. However, the results are quite different across the set of photographic and non-photographic images. The first ones are improved on average by 0.5% while the latter by 14%. The heuristic can be further exploited to perform more modifications and result with improvement of bitrate up to 17.5%. These extra modifications include “skipping the steps based on the actual bitrate rather than the estimated one and applying reversible denoising and lifting steps to SS-DWT”. However it is achieved with great time penalty [22]. Furthermore, it has been evaluated that applying the fixed skipped steps discrete wavelet transform (fixed SS-DWT) variants with the lossless compression compliant with the Part 2 of JPEG 2000 standard can provide another improvements [23].

Chapter 3. Subject of the thesis

3.1 Solution to the problem

3.1.1 Proposed algorithm

Taking into consideration existing solutions mentioned in chapter 2 Problem analysis it was decided to implement several extensions of the JPEG 2000 standard, Part 2. Moreover, proposed algorithm was designed to be parallelized. Such decision improves runtime performance especially in applications that require processing of great number of images. As it was announced in chapter 1 Introduction, the standard way of computing discrete wavelet transform in the Part 1 of the JPEG 2000 standard is to decompose the image into sub-bands using a pair of low- and high-pass filters. This decomposition is performed both in vertical and horizontal directions. Moreover, it is applied multiple times across whole processing tree.

Vast majority of available extensions was mentioned in section 2.2 Part 2 of the JPEG 2000. It was decided to choose solutions described in subsection 2.2.2 Arbitrary Decomposition and subsection 2.2.3 Arbitrary Wavelet Transforms in the planning phase of thesis development. As it was shown in the mentioned beforehand chapters, arbitrary decomposition extension creates a possibility of splitting the image into sub-bands of different shapes. The latter feature, i.e. arbitrary wavelet transform breaks strict selection rule of filters pair. With the help of these extensions and carefully chosen JPEG 2000 coder estimation process it is possible to adaptively choose best possible variant for the given image.

The workflow of the presented solution is as follows.

- The image is chosen by the user in the command line interface.
- The image is transformed to grayscale or left in RGB space upon choice of the user.
- The scheduler determines number of present hardware threads able to run program.
- Each thread dispatches possible combination of decomposition from precomputed look up table.

- The chain of selected DWTs is calculated in the context of each thread.
- During decomposition process entropy of image is calculated and stored for later usage.
- Minimal entropy is chosen after finish of the above mentioned calculations.
- Process is repeated for other filter pairs.
- Global minimum entropy is chosen and the best combination is shown to the user.

3.1.2 Different variants of DWT decomposition

As it was described in subsection 2.2.3 Arbitrary Wavelet Transforms, the Part 2 of JPEG 2000 standard allows to perform multiple modifications of decomposition even in the same step of DWT. The solution mentioned in subsection 2.4.2 Kakadu supports such extensive set of DWT variants. However, in this solution the combinations were narrowed to the most generic case. Moreover, the depth of discrete wavelet transform chain was limited to 5.

The chosen heuristic consists of four possible operation which are applied on the certain DWT level. The first one is a transformation compliant with Part 1 of the JPEG 2000 standard, i.e. 2D DWT. Therefore, in a result of such operation four sub-bands are created. The sub-band which was two time filtered with low-pass filter is once again transformed in the next DWT level. Another two options imply on transforming given image only row-wise or column-wise by applying 1D DWT. The low-pass filtered image is once again promoted to next DWT operations. Last possible transformation is basically no operation and resignation from another calculations

3.1.3 Selected filters

Although the design of the application is flexible in terms of selecting the pair of filters, it was decided to limit initial research to three options. The first option is Part 1 compliant solution, i.e. 5/3 LeGall filter. The other two filters are Haar (due to being special case of the Daubechies wavelet, the Haar wavelet is also known as “db1”) and 13x7 biorthogonal wavelet. The first pair of filters is depicted in the Table 3.1, while the second one is presented in the Table 3.2).

Table 3.1: Analysis and synthesis filter taps for the Haar filter bank

n	Low-pass, $h_0(n)$	Low-pass, $g_0(n)$
0	0.7071067811865476	0.7071067811865476
1	0.7071067811865476	0.7071067811865476

n	High-pass, $h_1(n)$	High-pass, $g_1(n)$
0	-0.7071067811865476	0.7071067811865476
1	0.7071067811865476	-0.7071067811865476

Table 3.2: Analysis and synthesis filter taps for the 13x7 biorthogonal wavelet filter bank

n	Low-pass, $h_0(n)$	Low-pass, $g_0(n)$
0	+0.966747552403483	+0.7071067811865476
±1	+0.4474660099696121	+0.3535533905932738
±2	-0.16987135563661201	-0.057543526228500
±3	-0.1077232986963881	-0.091271763114250
±4	+0.04695630968816917	
±5	+0.013810679320049757	
±6	-0.006905339660024878	

n	High-pass, $h_1(n)$	High-pass, $g_1(n)$
0	-0.7071067811865476	-0.966747552403483
±1	+0.3535533905932738	+0.4474660099696121
±2	-0.057543526228500	+0.16987135563661201
±3	+0.091271763114250	-0.1077232986963881
±4		-0.04695630968816917
±5		+0.013810679320049757
±6		+0.006905339660024878

3.1.4 Entropy as JPEG 2000 coder estimator

The entropy can be understood as the average level of “information” in the variable’s possible outcomes. Other ways of specifying this concept involve mentioning terms like “surprise” or “uncertainty”. Another name that is linked to entropy comes from the paper “A Mathematical Theory of Communication” authored by Claude Shannon. This name is related to original author’s surname as this type of entropy is introduced in some sources as Shannon’s entropy [4]. The entropy formal definition is as follows. A random discrete variable X is given. Possible outcomes of this variable are defined as $x_1, \ldots, x_n$. Then, the probability of occurrence of these

outcomes is depicted as $P(x_1), \ldots, P(x_n)$. With this set of variables the entropy of X can be presented in the way as shown in the formula 3.1 [25].

$$H(X) = -\sum_{i=1}^{n} P(x_i) log P(x_i) \tag{3.1}$$

The base of logarithm mentioned in the formula 3.1 can vary across different implementations. Base 2 is used in this particular application as it was dictated by the method used in the paper [25]. The units of selected logarithm base are called bits. Naming of the units is strictly related to choice of the specific logarithm. There are infinite number of possible bases but few of them are worth noticing. For instance, the base 10 can be useful in some applications. The name of the units is "dits" in this particular example. Moreover, natural logarithm with the base of e is also possible choice. In this case units are often referred as "nats" [4]. In this particular implementation, it assumed that $0 * log_2 0 = 0$ in the process of entropy calculation.

As it was suggested earlier, roots of entropy are present in the communication theory. This includes mainly work provided by Claude Shannon. The initial assumption in this field is the definition of the data communication system. Three components are included in this particular system. Data source is the first one, followed by communication channel and lastly the receiver is specified. The main function of the latter is to recognize what was produced by the given source relying only on the data propagated through the specific channel. Such functionality can be only established as long as there are used some techniques to properly compress and decode the signal. Moreover, it is necessary to distinguish the main signal from the noise. Multiple ways of encoding and compressing messages were discussed. The transmission from the source was also taken into account in the Shannon's theory. It concludes to another definition of entropy, the practical one. The entropy can be described as "an absolute mathematical limit on how well data from the source can be losslessly compressed and sent through perfectly noiseless channel" [4].

The memoryless entropy was chosen as an estimator of real compression results. Although it is not ideal estimation, such operation is good enough as results are proportional to these from JPEG 2000 coder [25]. Therefore entropy can be used just to determine choice between variants of DWT as described in 3.1.2. However, such calculation of entropy can achieve not satisfying results for low filtered images. The reason for such behavior lays in the very different characteristics of these images. The actual JPEG 2000 coder is not context-free as memoryless entropy. High filtered images are free of such error due to context neutralizing nature of such filters. To achieve same effect for low filtered images several methods were

tested. During tests it turned out that best results are achieved using difference between certain pixel and its left neighbor.

3.2 Implementation details

3.2.1 Chosen programming language

The features of given language exist to provide support for certain styles of programming. An individual language feature should never be perceived as a solution. Bjarne Stroustrup in his book “The C++ Programming Language” suggests that language feature should be rather perceived as one building brick from a varied set which can furtherly combined to express desired solutions [26]. Moreover, there are listed basic requirements for design and programming:

- Ideas shall directly be expressed in the code.
- Independent ideas shall be independently expressed in the code.
- Relationships shall be represented directly in code among other ideas.
- Expressed ideas shall be combined freely, where and only where combinations make sense.
- Simple ideas shall be expressed simply [26].

Given that, the C++ programming language supports four styles of programming:

- Procedural programming.
- Abstraction of data.
- Object-oriented programming.
- Generic programming.

Items in the list are not exclusive to each other. Moreover, as combination of chosen features more desirable solution can be achieved. There are several aspects that make software application successful, e.g. maintainability, readability, small size and fast time execution. To achieve it for a nontrivial problem these styles are recommended to use in conjunction [26].

The most notable solution present in the scope of algorithm implementation is generic programming. This type of programming is focused on the design, implementation and runtime use of general purpose algorithms. By “general” it is understood possibility of accepting variety of types by the algorithm. However, these types have to meet specified requirements of the given algorithm. The main support of generic programming in the C++ programming language is $template$ which provide static, i.e. compile-time polymorphism [26].

The C++11 standard introduced for the first time a thread library. It is heavily influenced by its ancestors, e.g. Boost Thread Library. These Boost classes were the primary model on which the standard library is based. Moreover, many entities share same names and structure [30]. However, there are some disadvantages in the standard threading library. Notable features are missing in comparison with other threading APIs, i.e. “pthreads” and Windows threads. The reason lays in the extensive set of constraints on compiler implementers. Nevertheless, multithreaded applications can be developed with standard behavior across all platforms. This enables a solid foundation on which libraries utilizing parallelism can be built. The concurrency elements of the Standard Library like futures, threads, mutexes and atomics are the important but not the only available solutions for developing concurrent C++ software [19].

The thread-based programming provide higher level of abstraction in comparison to task-based one. Therefore it makes thread management easier. However, there are multiple types of “thread” that can be encountered in concurrent C++ software:

- Hardware threads are the ones that actually perform computation. Machine architectures offer one or more hardware threads per CPU core.

- Software threads, also known as system threads are the ones that operating system manages across all processes. The scheduler prepares their execution on hardware threads. Therefore, it is possible to create more software threads than the hardware ones. The motivation is execution of unblocked threads when the other ones are blocked, e.g. wait for mutex.

- “std::threads” are C++ objects that act like handles to underlying software threads. Standard thread can correspond to nothing so it represents “null” handle. There are several scenarios leading to this behavior. Default-constructed objects with no function to execute are the first example. The other ones include object state after being moved from, joined and detached thread [19].

3.2.2 Build environment

The build environment is setup around CMake. This software is open-sourced, cross-platform tool designed to not only build but also test and package various applications. CMake is aimed to control the compilation process of given software using compiler and platform agnostic configuration text files. Such setup allows to generate platform native tools to build project, e.g. Makefiles for GNU/Linux or nmake for Windows. Moreover, it is possible to generate Visual Studio project files with the help of CMake. This software is also capable of generating wrappers, creating both dynamic and static libraries and building executables in arbitrary combinations. Another viable CMake feature is its caching system. Upon running build such text file is generated and is ready to be investigated using graphical editor. The located files, libraries, executables and optional build directives are placed in the cache [3].

The CMake tool is arranged to support complex structure of directory hierarchies and applications which can be dependent on other libraries. The other complex scenario that can be handled by CMake is building the executables in the specific order to generate code which is compiled and linked into eventual application. The building processes is managed by CMakeLists.txt files that have to be located in each directory containing source files. The sample CMakeLists.txt file can be seen at the listing 1. This software provides not only predefined commands but also a possibility of adding user functions.

3.2.3 DWT interface

As shown at the listing 2, two dimensional discrete wavelet transform is strictly connected to the one dimensional. The reuse of this function makes design more flexible. It is worth pointing out that these function are allocation agnostic. Therefore the user is obligated allocate memory at first and then make use of dwt processing. Such decision is dictated upon fact that actual type of matrix containing data is not specified. Therefore a pointer to first element and total size are the most feasible solution. There is one disadvantage that hits the runtime of presented algorithm. The results from “cols” and “rows” functions are not cached in memory. Thus, upon calculating “full” DWT variant operations are unnecessarily doubled.

3.2.4 Testing

The testing, especially test driven development takes essential place in the software development process. For the purpose of checking correctness of proposed discrete wavelet transform algorithm “PyWavelets” (Wavelet Transforms in

```
project(jpeg2000_src)
add_subdirectory(dwt)
if(BUILD_EXTERNALS AND EXISTS ${JP3D_PATH})
    add_subdirectory(${JP3D_PATH})
endif()

set(SOURCES
    demo_dwt.cpp
    demo_queue.cpp
    demo_opencv.cpp
    arg_parser.cpp
    main.cpp
)
add_executable(jpeg2000 ${SOURCES})
set_target_properties(jpeg2000 PROPERTIES COMPILE_FLAGS "-save-temps")
if(BUILD_EXTERNALS)
    target_include_directories(jpeg2000 SYSTEM PRIVATE
        ${OPENCV_PATH}/modules/core/include
        ${OPENCV_PATH}/modules/imgcodecs/include
        ${OPENCV_PATH}/modules/imgproc/include
        ${CMAKE_BINARY_DIR} # OpenCV is looking for
        ↪  opencv2/opencv_modules.hpp
        ${CXXOPTS_PATH}/include
    )
    target_compile_definitions(jpeg2000 PRIVATE BUILD_OPENCV)
endif()
target_include_directories(jpeg2000 SYSTEM PRIVATE
    ${CMAKE_SOURCE_DIR}
    ${TIMER_PATH}
)
target_link_libraries(jpeg2000 PRIVATE
    dwt
    cxxopts
)
if(BUILD_EXTERNALS)
    target_link_libraries(jpeg2000 PRIVATE
        opencv_core
        opencv_imgcodecs
        opencv_imgproc
    )
    if(EXISTS ${JP3D_PATH})
        target_link_libraries(jpeg2000 PRIVATE jp3d)
    endif()
endif()
```

Listing 1: Sample CMakeLists.txt file

```
REGISTER_DWT_2D(dwt_2d_rows) {
    std::size_t out_cols = get_n_dwt_output(cols, n_filter);
    for (std::size_t i{}; i < rows; i++) {
        dwt_1d(input + i * cols,
               cols,
               filter,
               n_filter,
               output + i * out_cols,
               mode);
    }
}

REGISTER_DWT_2D(dwt_2d_cols) {
    for (std::size_t i{}; i < cols; i++) {
        dwt_1d(input + i, rows, filter, n_filter, output + i, mode,
         ↪  cols);
    }
}

REGISTER_DWT_2D(dwt_2d) {
    std::size_t out_cols = get_n_dwt_output(cols, n_filter);
    std::vector<T> tmp;
    tmp.reserve(out_cols * rows);
    dwt_2d_rows(input, rows, cols, filter, n_filter, tmp.data(), mode);
    dwt_2d_cols(tmp.data(), rows, out_cols, filter, n_filter, output,
     ↪  mode);
}
```

Listing 2: Two dimensional discrete wavelet implementation

Python) software was used as the reference model [12]. Thanks to pybind11 [9] C++ was exposed to the Python and then tested using pytest framework. The first one is a lightweight header-only library that synergizes both Python and C++. The main purpose is to create Python bindings for existing C++ code. Boilerplate code is minimized thanks to inferring type information using introspection during compile-time [12]. On the other hand pytest framework makes writing small and readable tests easier. Moreover, it can scale up to support complex functional testing for other libraries and applications [10].

The used technique to check correctness of written DWT functions in this application are testing fixtures. Their purpose is to provide a defined, reliable and consistent context for the written tests. The environment, e.g. database configured with known parameters or content, e.g. dataset can also be included in such a fixture. Moreover, steps and data constituting the arrange phase of a test are defined.

For more complex scenarios fixtures can be also used to define active phase of given test which is a case in the thesis. Moreover, the state, services and other operating environment set up by the fixtures can be accessed by testing functions through argument lists. Every fixture that is used by test function has its special parameter named after the fixture name in the definition of the function [10]. The example fixture defined with the help of pytest is present at the listing 3.

```
def get_all_test_suits():
    return chain(
        get_precision_test_suits(np.float32),
        get_precision_test_suits(np.float64)
    )


@pytest.fixture
def dwt_fixture(request):
    data, wavelet, padding_mode, dwt_func = request.param
    precision = data.dtype.name
    dwt_ref, dwt_impl = DWT_FUNCS[dwt_func]
    dwt_impl = dwt_impl[PRECISION[precision]]
    ref = dwt_ref(data, wavelet, padding_mode)
    out = dwt_impl(data, WAVELETS[wavelet][PRECISION[precision]],
                   PADDING_MODES[padding_mode])
    print(f"{ref}\n{out}")
    return ref, out


@pytest.mark.parametrize("dwt_fixture", get_all_test_suits(),
    indirect=True)
def test_dwt(dwt_fixture):
    assert np.allclose(*dwt_fixture)
```

Listing 3: Test fixture of DWT

3.2.5 Parallel for

The base function of the parallelized for version is available at the listing 4. The signature of this function makes its usage very flexible. All types of callable entities can be employed to perform operations on specified thread index. Moreover, vector from Standard Template Library is used as safe and extensible abstraction over dynamically created arrays. Each thread from c++ standard requires initialization with some type of callback to be able to start. The thing that is worth to notice is intentional pass of such function by constant reference instead of forwarding reference.

The reason of this implementation is that threads should operate on copied context from the main thread. There is no advantage of moving function objects between the master thread and first forked one because other threads may end up with hanging state of its context. Lastly, main thread has to wait for worker threads to be ready with their specified tasks. Therefore join method is used at the end of scope.

```cpp
#ifndef JPEG2000_PARALLEL_FOR_HPP
#define JPEG2000_PARALLEL_FOR_HPP

#include "config.hpp"

#include <concepts>
#include <functional>
#include <thread>
#include <vector>

namespace mgr {
namespace detail {

// clang-format off
template<typename Func, typename... Args>
// std::invocable<Func, Args...> should really be used in conjunction
// enable it with the release of libc++13 (support of <concepts>
    ↪ header)
concept no_returnable = std::same_as<std::invoke_result_t<Func,
    ↪ Args...>, void>;

template<typename Func, typename... Args>
concept returnable = !no_returnable<Func, Args...>;
// clang-format on

template<typename Func>
void parallel_for(std::size_t n_threads, const Func& func) {
    std::vector<std::thread> threads;
    threads.reserve(n_threads);
    for (std::size_t thread_idx{}; thread_idx < n_threads;
        ↪ thread_idx++) {
        threads.emplace_back(func, thread_idx);
    }
    for (auto& thread : threads) {
        thread.join();
    }
}
} // namespace detail
```

Listing 4: parallel_for.hpp: Base function

User interface of the parallel for is available at the listing 5. Two separate functions were developed during implementation phase of thesis creation. Thanks to one of the major features of C++20, i.e. concepts, interface of these functions is not that complicated. Other solutions include using SFINAE based approach which is very hard to reason about and has many pitfalls. Another possible substitute of concepts is just different naming of these functions. The drawback of this scenario is loose in readability and usability. It is worth noticing that there is no possibility of overload function basing on its return type in C++. The reason of this behavior is that return type is not included in the signature of function.

The first function allows to conveniently parallelize set of computations with no return type. Thanks to functionalities split there is no runtime penalty in this case. From the user perspective any callable type can be passed as parameter of this function. However, there is one unnamed requirement. Fortunately enough it is caught during compilation time instead of runtime. Passed function-like object has to be invoked with one argument. The reason of such design is to make user responsible for specifying what should happen during calculation process of each loop iteration. During dispatchment of first for iteration each thread starts at the different index. Furtherly such index is incremented with number of threads specified by the user. Therefore, each calculation is guaranteed to take place in the appropriate thread context.

The second function extends the properties of the first one by adding the possibility of returning value from each loop iteration. The return type is deduced thanks to compile time introspection capabilities of “std::invoke_result” which works on any callable entity [2]. It is worth to notice that “std::vector” which stores eventual results is zero-initialized at first. The reasoning includes concurrent access to its elements. Therefore, it is virtually impossible to ensure that “push_back” or “emplace_back” methods do not result in data race without locking mechanism.

3.2.6 Queue generation

The calculation process of DWT processing size is available at the listing 7. As was described earlier in the thesis, depth of DWT calculations is set to five. From the mathematical point of view, such processing queue is a cartesian product of its values. However, the processing chain includes not performing discrete wavelet transform at all and returning early from algorithm. Therefore, it was decided to limit scope of possible results. This function features new keyword in C++20, i.e. “consteval” which makes sure that function can be used only in the compile time.

```cpp
template<typename Func>
requires detail::no_returnable<Func, std::size_t>
void parallel_for(std::size_t n_threads, std::size_t n_elements, Func&&
 ↪  func) {
    detail::parallel_for(
        n_threads,
        [n_threads, n_elements, func = std::forward<Func>(func)](
            std::size_t thread_idx) mutable {
            for (std::size_t i{thread_idx}; i < n_elements; i +=
             ↪  n_threads) {
                func(i);
            }
        });
}

template<typename Func>
requires detail::returnable<Func, std::size_t>
auto parallel_for(std::size_t n_threads, std::size_t n_elements, Func&&
 ↪  func) {
    std::vector<std::invoke_result_t<Func, std::size_t>>
     ↪  result(n_elements);
    detail::parallel_for(
        n_threads,
        [n_threads, n_elements, func = std::forward<Func>(func),
         ↪  &result](
            std::size_t thread_idx) mutable {
            for (std::size_t i{thread_idx}; i < n_elements; i +=
             ↪  n_threads) {
                result[i] = func(i);
            }
        });
    return result;
}
} // namespace mgr

#endif // JPEG2000_PARALLEL_FOR_HPP
```

Listing 5: parallel_for.hpp: User interface

The generation and then instantiation of DWT processing queue is available at the consecutive listings 7 and 8. As it was specified earlier, these computations are done in the compile time thanks to “template” and “constexpr” meta programming. The instantiation process requires three pieces of information. First one is type of single element in the queue which is set up to specific DWT callback in this case.

```cpp
#ifndef JPEG2000_QUEUE_HPP
#define JPEG2000_QUEUE_HPP

#include "config.hpp"
#include "dwt_2d.hpp"
#include "template_helpers.hpp"

#include <array>

namespace mgr {
namespace detail {

constexpr std::size_t depth = 5;
constexpr std::size_t n_dwt_cbs = lut_dwt_2d_cbs<float, float>.size() -
    1;

consteval std::size_t get_queue_size() {
    std::size_t temp{1};
    std::size_t result{temp};
    for (std::size_t i{}; i < depth; i++) {
        temp *= n_dwt_cbs;
        result += temp;
    }
    return result;
}
```

Listing 6: Calculation of size in DWT processing queue

Second one is related to the element which is supposed to stop chain of calculations. Lastly, array containing all needed callbacks is required.

On the other hand, generation process is more convoluted. The main algorithm consists of recursive call of function which performs calculation of modified cartesian product described earlier. In the basic path previously generated arguments are forwarded and the new one is appended at the end. However, if the “stop” operation is detected, the recursive chain is terminated and the results are saved to the buffer. Last case involves break of the whole process when the maximum DWT level is reached. All of these steps are performed during compile-time so there is no runtime penalty. In the result look-up table is produced in the “.rodata” memory section.

3.2.7 OpenCV

The OpenCV, i.e. Open Source Computer Vision Library, is a software module that includes several hundreds of specific algorithms used in computer vision. This

```
template<typename Elem, Elem Pad>
struct cartesian_prod {
    template<typename T,
                std::size_t CartesianCounter = depth,
                typename... Args>
    consteval auto calculate(T&& array, Args&&... args) {
        for (std::size_t i{}; i < array.size(); i++) {
            if constexpr (CartesianCounter != 0) {
                if (i) {
                    calculate<T, CartesianCounter - 1>(
                        array,
                        std::forward<Args>(args)...,
                        array[i]);
                } else {
                    fill_queue(pad_sequence<Elem, Pad,
                     ↪ CartesianCounter>{},
                                std::forward<Args>(args)...);
                }
            } else {
                fill_queue(std::forward<Args>(args)...);
                break;
            }
        }
        return queue_;
    }

private:
    template<typename... Args, typename T, T... Is>
    consteval void fill_queue(sequence<T, Is...>, Args&&... args) {
        queue_[queue_counter_++] = std::to_array(
            {std::forward<Args>(args)..., Is...});
    }

    template<typename... Args>
    consteval void fill_queue(Args&&... args) {
        queue_[queue_counter_++] = std::to_array(
            {std::forward<Args>(args)...});
    }

    std::array<std::array<Elem, depth>, get_queue_size()> queue_{};
    std::size_t queue_counter_{};
};
} // namespace detail
```

Listing 7: Generation of DWT processing queue

```
template<typename T>
inline constexpr auto
    dwt_queue = detail::cartesian_prod<dwt_2d_cb<T, T>, no_dwt_2d<T,
    ↪  T>>{}
                    .calculate(lut_dwt_2d_cbs<T, T>);

} // namespace mgr
```

Listing 8: Instantiation of DWT processing queue

library has modular structure. Therefore, packages can include several shared or static sub-libraries. Since version 2.x OpenCV exposes only C++ API in opposition to C based first version. The following list of modules is available:

- Core functionality is a compact library that defines basic data structures. Dense multi-dimensional array, i.e. Mat and fundamental functions, used all over the place by other parts of the OpenCV, are included in this module.
- Image Processing includes both liner and non-linear image filtering and geometrical transformations such as resizing, affine and perspective warping. Moreover, this module support color space conversion and histogram calculation.
- Video Analysis consists of features like motion estimation, background subtraction and object tracking algorithms.
- Camera Calibration and 3D Reconstruction includes basic multiple-view geometry algorithms, both single and stereo camera calibration, object pose estimation, stereo correspondence algorithms and elements of three dimensional reconstruction.
- 2D Features Framework consists of salient feature detectors, descriptors and their matchers.
- Object Detection features possibility of objects and instances of predefined classes detection. It includes for instance faces, eyes, mugs, people and cars.
- High-Level GUI is a simple interface supporting basic user needs.
- Video I/O supports video capturing of various video codecs.
- Other helper modules, e.g. Google test wrapper and Python bindings.

Chapter 4. Experiments

4.1 Methodology

4.1.1 Time execution

The main purpose of time execution comparison in this paper is to deliver results of hypothetical runtime improvements between the single threaded application, C++ standard execution policies and the custom implementation described in the subsection 3.2.5 Parallel for. Therefore described methods are benchmarked with the usage of the same hardware. Moreover, several threads configuration are tested in the case of custom implementation. The expected best performance is to be available around the number of available hardware threads.

At the listing 9 there is presented timer class used to measure execution of DWT processing chain. It is written in RAII format, i.e. Resource Acquisition Is Initialization. RAII is popular modern C++ idiom to control the lifetime of the object, e.g. it is used for scope based dynamic memory management wrappers. Another use case is control of locking and unlocking mechanisms such as mutexes. The C++ standard library implements such resource managers in a similar way. One action is taken upon construction time of the object, i.e. in its constructor and the opposite procedure takes place during destruction process. This approach enables possibility of time measurement automatization. Firstly, time is measured at the end of initialization process to avoid overhead of constructing helper objects. Secondly, such measurement takes place at the beginning of the destruction handling for very similar reason [2].

Class “std::chrono::high_resolution_clock” is aimed to represent time measurement mechanism with the smallest tick period that is provided by certain implementation of the standard C++ library. Usually the compiler vendors implement this entity as an alias of “std::chrono::system_clock” or “std::chrono::steady_clock”, or another independent clock [2]. In the case of used implementation, i.e. gcc’s libstdc++, it is an alias of “std::chrono::system_clock”. Such decision is dictated by the need of using so called wall-clock time [2].

Elapsed real time or wall-clock time is the actual time that is needed to take from the start of a given computer program to its end. It can be understood as a

difference between the time when a certain task finishes and the time when that task started running [13]. Wall time is thus distinguishable from CPU time. The latter one measures only the time during active work on given task is performed by the processor. This difference comes from architecture and runtime dependent factors. For example, it can involve intentional delays and waiting for availability of certain system resources. It can be better understood in the following example. Take into consideration mathematical script which results in producing such report: “Measured CPU time: 0.09s and wall-time: 5m 20.36s”. For over 5 minutes this program was active and during that time the processor spent only fraction of second performing mathematical operations. Therefore it was found to be suitable solution in the concurrent execution of DWT processing chain.

At the listing 10 there are depicted several template specializations for handling different types of measurements. The user of this class can specify the accuracy with the C++ standard aliases for time primitives. If there is need for such low unit of time as nanoseconds, the actual time is returned to the user in the bigger format. This approach is a convenient way of handling time measurement for this particular application.

The measurements took place on the one specific hardware setup. The notebook used in the experiments is not usually used as reference in the benchmarking process. However, it represents quite well a day-to-day usage for average person. The specification of this hardware and software setup is presented below. Processor details are as follows:

- Name: AMD Ryzen 2500U with Radeon Vega Mobile Gfx.
- Producer: AuthenticAMD.
- Number of cores: 4.
- Number of threads: 8.
- Base clock: 2.0 GHz.
- Max boost clock: up to 3.6 GHz.
- Total L1 cache: 384 kB.
- Total L2 cache: 2 MB.
- Total L3 cache: 384 kB.
- CMOS: 14 nm.

```cpp
#include <chrono>
#include <fstream>
#include <iostream>

template<typename T = std::chrono::milliseconds>
class Timer {
public:
    using clock = std::chrono::high_resolution_clock;

    Timer() : ostream_{std::cout} {}
    explicit Timer(std::ofstream& fileStream) : ostream_{fileStream} {}
    explicit Timer(const std::string& filepath) :
        fileStream_{filepath, std::ios::app},
        ostream_{fileStream_} {}

    ~Timer() {
        StopTimer();
    }
    Timer(const Timer& other) = delete;
    Timer& operator=(const Timer& other) = delete;
    Timer(Timer&& other) = delete;
    Timer& operator=(Timer&& other) = delete;

private:
    std::ofstream fileStream_{};
    std::ostream& ostream_;
    std::chrono::time_point<clock> startTimePoint_{clock::now()};

    void StopTimer() {
        const auto endTimePoint = clock::now();
        const auto start =
        ↪  std::chrono::time_point_cast<T>(startTimePoint_)
                                .time_since_epoch()
                                .count();
        const auto end = std::chrono::time_point_cast<T>(endTimePoint)
                               .time_since_epoch()
                               .count();
        SaveTime(start, end);
    }

    template<typename Count>
    void SaveTime(Count start, Count end);
};
```

Listing 9: Timer.hpp: User interface

```
template<>
template<typename Count>
void Timer<std::chrono::nanoseconds>::SaveTime(Count start, Count end)
    ↪ {
    ostream_ << "Elapsed time: " << static_cast<double>(end - start) /
        ↪ 1000.0 << " us  \n";
}

template<>
template<typename Count>
void Timer<std::chrono::microseconds>::SaveTime(Count start, Count end)
    ↪ {
    ostream_ << "Elapsed time: " << static_cast<double>(end - start) /
        ↪ 1000.0 << " ms  \n";
}

template<>
template<typename Count>
void Timer<std::chrono::milliseconds>::SaveTime(Count start, Count end)
    ↪ {
    ostream_ << "Elapsed time: " << static_cast<double>(end - start) /
        ↪ 1000.0 << " s  \n";
}

template<>
template<typename Count>
void Timer<std::chrono::seconds>::SaveTime(Count start, Count end) {
    ostream_ << "Elapsed time: " << end - start << " s  \n";
}

template<>
template<typename Count>
void Timer<std::chrono::minutes>::SaveTime(Count start, Count end) {
    ostream_ << "Elapsed time: " << end - start << " min  \n";
}

template<>
template<typename Count>
void Timer<std::chrono::hours>::SaveTime(Count start, Count end) {
    ostream_ << "Elapsed time: " << end - start << " h  \n";
}
```

Listing 10: Timer.hpp: Template specializations

The Random Access Memory (RAM) installed in the notebook has following capabilities:

- Installed memory: 8 GB.
- Available memory: 6.9 GB.
- Configuration of slots: 2 x 4 GB.
- Frequency: 2400 Mhz.
- System Memory Type: DDR4.

The operating system used is Windows 10 Home Edition 21H1, while chosen compiler is gcc 10.3 from MSYS2 distribution.

4.1.2 Image compression

The output of proposed algorithm is set of chosen filter pair and discrete wavelet transform decomposition. It is necessary to somehow evaluate these results with the real JPEG 2000 encoding technique. Moreover, it is essential to compare such results with the best possible ones obtained using mentioned earlier encoder. Therefore Kakadu software was used both as the reference and validation codec.

However, during the study it was found out that solution described in this thesis poorly chooses the best possible pair of filters to be applied on the image. To mitigate such inconvenience it was decided to modify main algorithm. The modification includes storing information of discrete wavelet decomposition type with every filter under the test. Although such approach results in the performance hit of proposed solution, it enables basic functionality which is determination of the more efficient image compression. Described mitigation involves validation at the side of the actual JPEG 2000 Part 2 compliant codec. The results are gathered and later processed with the basic statistics methods.

The reference results are acquired in slightly different fashion. The main disadvantage of using whole JPEG 2000 encoding process is time needed to perform such calculations. The amount of possible discrete wavelet transform decompositions in product with different filter pairs can exceed the number of thousand in specific cases. Taking into consideration not small number of test images it was decided to improve this process by dropping empirically chosen types of transforms. Firstly, during the experiments it was observed that vast majority of images performs best in terms of image compression when the level of DWT chain reached number of five. Therefore it was decided to drop exhaustive search method for smaller ranks. Another observation was made, i.e. the pair of used filters with the Part 1 compliant decomposition determined the obtained result in the more significant way than

Part 2 compliant discrete wavelet transform chain with the Part 1 filter. Therefore to greatly optimize runtime of this procedure, filter pairs with the Part 1 decomposition are checked at first.

Lastly, the gathered results, both from reference and from proposed solution are evaluated and compared with each other.

4.2 Data sets

The image data set gathered to evaluate results of the thesis takes its origin from one of Tilo Strutz's papers [14]. It was used at first to evaluate multiplierless reversible color transforms. The main purpose of that paper was to propose an entire family of multiplierless reversible color transforms and investigate their performance in lossless image compression [28]. Therefore it was found out to be suitable for the case of this thesis as lossless image compression on color pictures is applied as well. In the selected data set there are 104 element available. The exemplary elements of the image data set are available at figures 4.1, 4.2 and 4.3. It is worth noticing that only ppm (portable PixMap) images were selected due to lack of png format support in Kakadu library.

Figure 4.1: The example image from the reference data set [14]

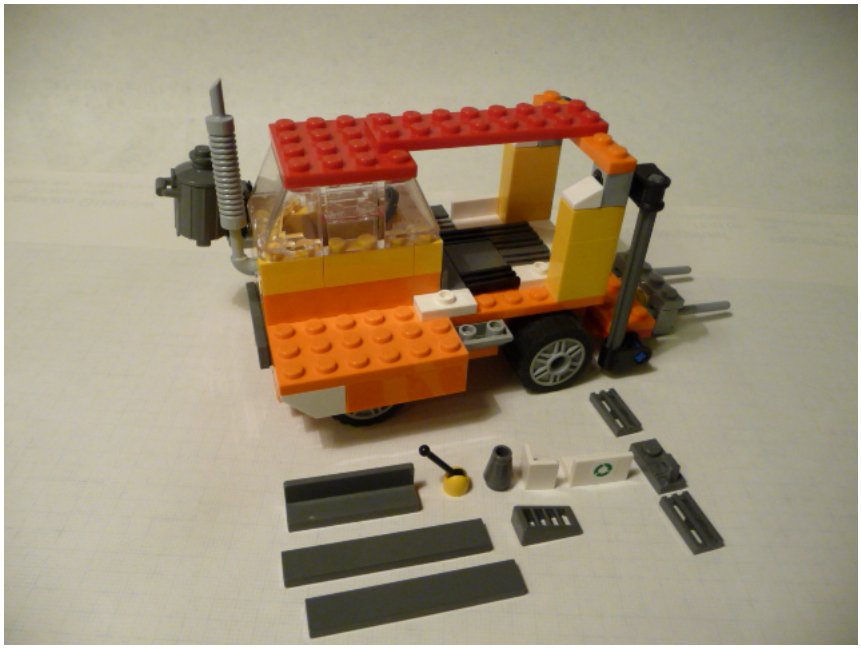

Figure 4.2: The example image from the reference data set [14]

Figure 4.3: The example image from the reference data set [14]

4.3 Results

4.3.1 Reference results

The reference results were gathered using the actual JPEG 2000 codec instead of an estimation. The mean improvement in terms of image compression between Part 1 and Part 2 compliant solutions turned out to be on average equal to 1.982% with the standard deviation of 1.307%. The maximum possible improvement achieved on specific image was 10.661% while the minimum was set to 0.

As far as only reference results are concerned it can be concluded that Part 2 extensions such as modification of discrete wavelet decomposition and change of selected filter pair improve compression ratio in selected data set. However the results are quite differentiated. For some images the improvement was significant (over 10%), while for others there was no improvement. Such behavior probably involves crash of Kakadu application for certain image. Taking into consideration the size of data set (104 images), the standard deviation seems credible.

4.3.2 Estimated best configuration

The mean improvement in terms of image compression between Part 1 and Part 2 compliant solutions turned out to be on average equal to 1.5% with the standard deviation of 1.305%. The maximum possible improvement achieved on specific image was 10.169% while the minimum was set to 0.

As far as only proposed solution is concerned it can be concluded that Part 2 extensions such as modification of discrete wavelet decomposition and change of selected filter pair improve compression ratio in selected data set. However the results are quite differentiated. For some images the improvement was significant (over 10%), while for others there was no improvement. Such behavior is present due to fact that Part 1 solution was used as fallback when no better results were found

4.3.3 Comparison with reference

Taking into consideration results presented in the subsection 4.3.1 Reference results and subsection 4.3.2 Estimated best configuration it can be stated that they are quite similar. The improvement coming from the reference results on average is better by 33%. This behavior is expected as the estimation method of proposed solution was memoryless entropy, while the JPEG 2000 codec has its own memory. Another thing that is worth to notice is that the choice of filter pair was more important for the final result than the DWT decomposition. Therefore proposed solution of estimating different shapes of discrete wavelet transform applied to the image does not guarantee the best possible results. This is due to fact that there was an impasse on selecting best feasible filter.

4.3.4 Time comparison

The runtime of discussed solution was measured and later evaluated. Comparison of different methods at exemplary image is presented at the figure 4.4. The

consecutive numbers placed after “thread” indicate usage of parallel for version presented earlier. These numbers correspond to number of software threads used in the experiment. The last record, i.e. “policies” describes usage of execution policies introduced in C++17. As can be seen at this figure, selected compiler has not managed to optimize this particular problem. The reason of this failure may be caused by obscure and non-trivial code. However, proposed version of parallel computation works as expected. The speedup is strictly connected to the number of hardware threads which is 8 in this case.

During evaluation process one anomaly was observed (figure 4.5). After investigation it was concluded that operating system decided to suspend execution of the program due to inactivity. As can be observed at the figure 4.6 this anomaly highly contributed to the average measures. It was decided that it has to be eliminated from the data set in the preprocessing step. At the figure 4.7 such change is shown. These results are very similar to the ones presented at first (figure 4.4). The improvement was exhausted around number of hardware threads. Therefore it is not beneficial to use number of workers greater than hardware concurrency.

The time comparison of discussed solution with the reference cannot be established as the computations took place on different machine. However, it was observed that using Kakadu with JPEG 2000 compliant codec was several times slower on average than the proposed algorithm.

4.3.5 Visualized sample results

There are presented sample outputs of various DWT variants applied to famous 512x512 picture of Lena Forsén [8]. All the images are results of low-pass filtering from the last rank of discrete wavelet transform. The first image 4.8 represents the Part 2 compatible decomposition variant where transformation is applied each time only to columns. Therefore the shape of result is very thin rectangle. Second picture 4.9 illustrates mainly row-wised based DWT. Another figure 4.10 depicts mixed version of discrete wavelet transform described before. Last one 4.11 is the output of Part 1 compliant decomposition.

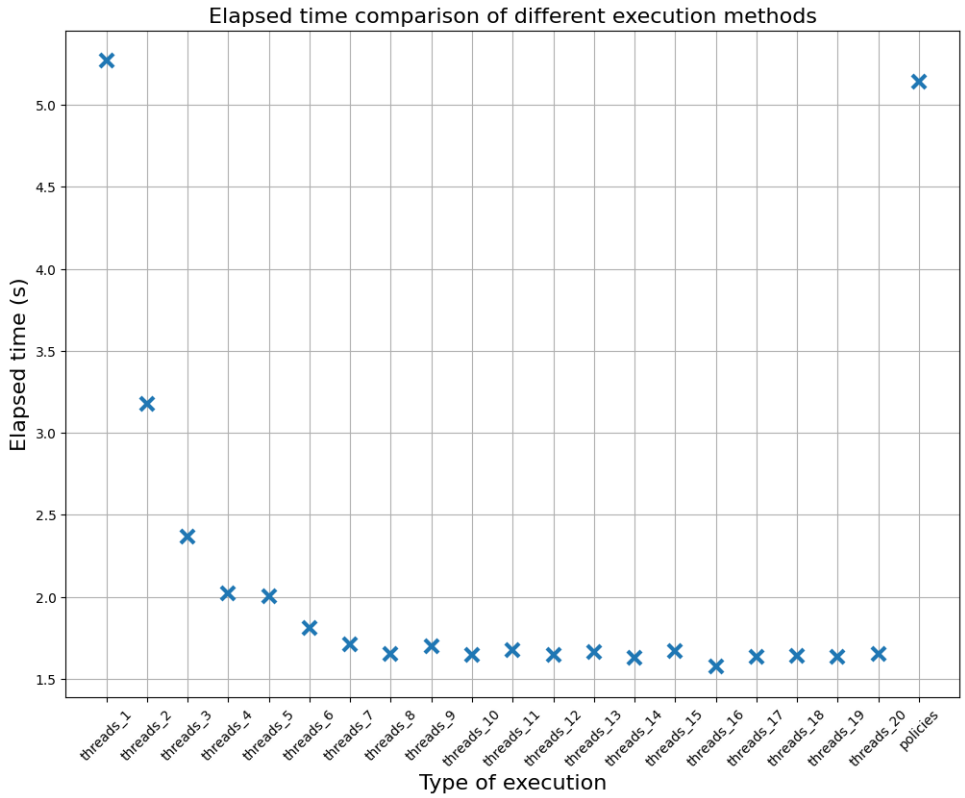

Figure 4.4: Elapsed time comparison of different execution methods at exemplary image

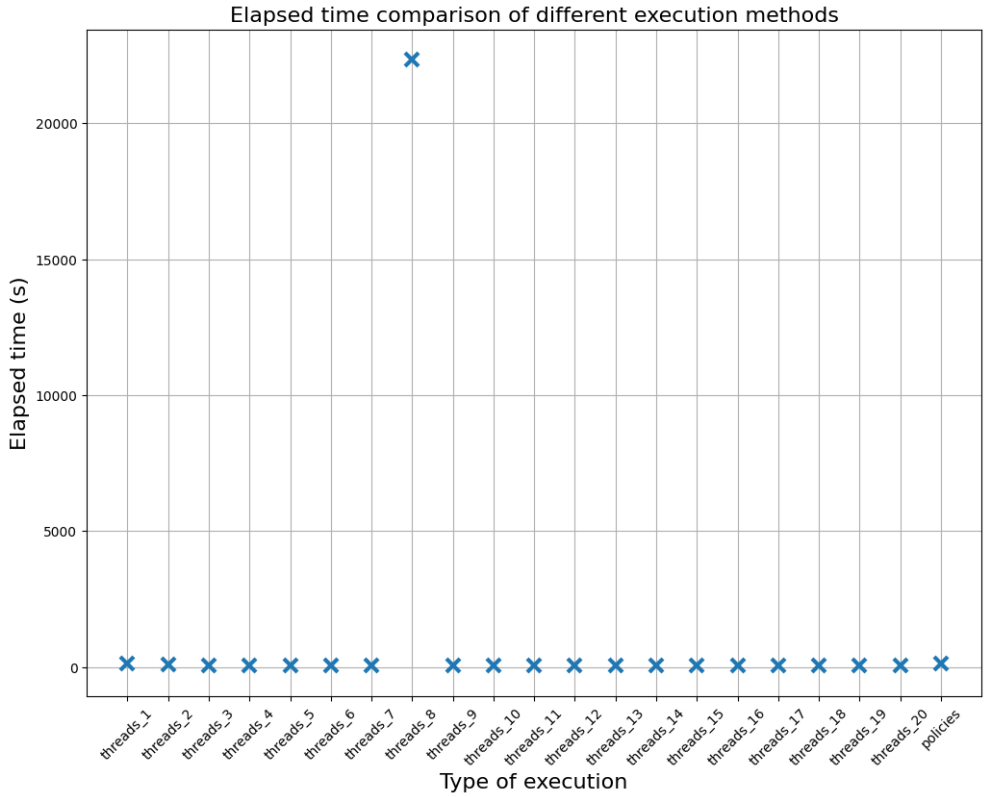

Figure 4.5: Elapsed time comparison of different execution methods at suspicious run

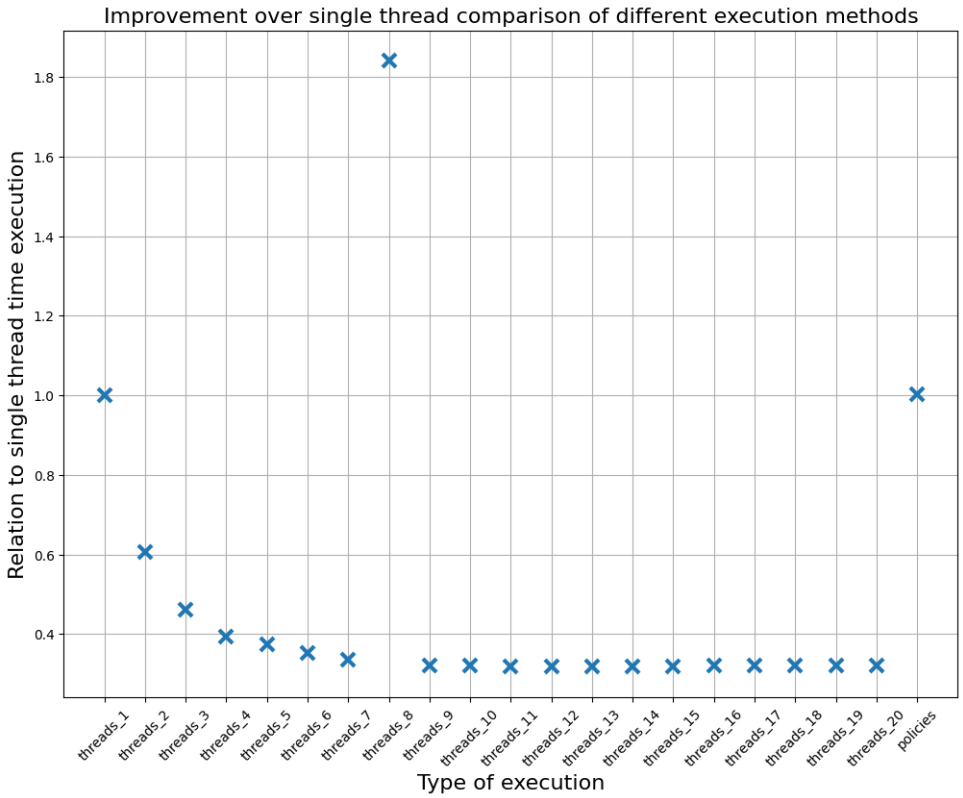

Figure 4.6: Mean improvement over single thread execution comparison without outliers detection

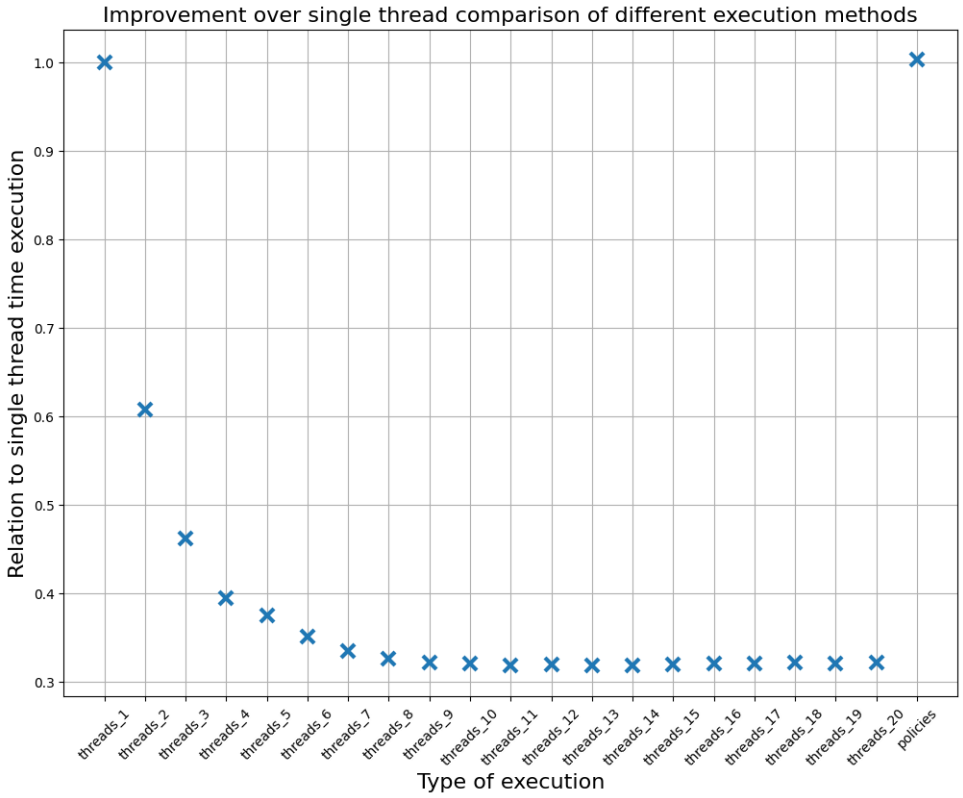

Figure 4.7: Mean improvement over single thread execution comparison after outliers removal

Figure 4.8: The example output image of DWT processing chain using Lena test image [8]

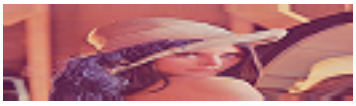

Figure 4.9: The example output image of DWT processing chain using Lena test image [8]

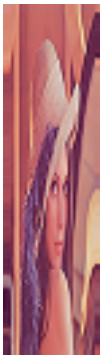

Figure 4.10: The example output image of DWT processing chain using Lena test image [8]

Figure 4.11: The example output image of DWT processing chain using Lena test image [8]

Chapter 5. Summary

The objective of the thesis was to develop, implement and test certain type of application. The software should be capable of determining specific configuration of chosen JPEG 2000 Part 2 extensions for the given image. These extensions include decomposition using discrete wavelet transform in a way which is not compliant with Part 1 of the JPEG 2000 standard and making use of different pair of filters. The main goal was to achieve better results in terms of compression ratio than the mentioned before “Part 1”. As was concluded in the section 4.3 Results, the objective of the thesis was fulfilled. The realized tasks during all the phases of thesis development are as follows.

- Initial research in fields of image processing and compression, analysis of JPEG 2000 algorithm.
- More advanced research of algorithms such as DWT, SS-DWT and JPEG 2000 implementation - Kakadu.
- Development of basic DWT implementation.
- Development of advanced 2D DWT implementation with possibility of skipping transformation of columns or rows.
- Setup of Continuous Integration system and implementation of DWT testing component
- Development of initial heuristics allowing to study the effects of DWT modifications compliant with the JPEG 2000 standard such as decompositions into sub-bands and usage of different filters
- Support for both grayscale and color images.
- Initial implementation of multi-threaded heuristics.
- Conducting preliminary tests and selecting modifications or their variants to be included in the final heuristics.
- Development of multi-threaded optimized implementation of final heuristics.

- Research on final heuristics - comparison in terms of obtained compression ratio and time with: unmodified JPEG 2000, SS-DWT transformation and the transformation determined by an exhaustive search.

As far as the results are concerned, the mean improvement in terms of image compression between Part 1 and Part 2 compliant solutions turned out to be on average equal to 1.5% with the standard deviation of 1.305%. The reference results that were gathered using actual JPEG 2000 codec came in to be on average equal to 1.982%. As was described in section 4.3 Results, the standard deviation equaled 1.305% in this particular case.

Moreover, some conclusions were made according to the achieved results. As it was presented in the subsection 4.3.4 Time comparison, the custom implementation of multithreading achieved the best performance. The number of worker threads should be closely related to number of hardware threads available on the specific platform. There are also enhancements in terms of image compression as described in subsection 4.3.3 Comparison with reference. Despite the improvement from reference results being better on average by 33%, the described solution managed to improve bit rate by 1.5% in comparison to JPEG 2000 Part 1 compliant solution. Therefore it can be concluded that the objective of the thesis has been reached.

Despite being successful in the field of compression improvement in lossless image compression, the best possible performance was not achieved. Moreover, run-time of described algorithm can also be reduced. However, it was observed that using Kakadu with JPEG 2000 compliant codec was several times slower on average than the proposed algorithm. As study shows, compression rate on average was lagging behind the reference at the rate improvement of 33%. Although this gap cannot be fully filled, more research on investigating better heuristic's weights can be developed. The hypothetical improvement can be also achieved by proposing algorithm that is able to distinguish best possible filter pair from the other ones.

As it was stated before, the run-time performance of the algorithm and its future extensions can also be improved. The process of best possible filter pair and discrete wavelet transformation searching can be optimized to perform more iterative based approach. Currently all possible configuration are generated at the compile time. At the run-time of application every possible decomposition for selected filter pair is calculated and then compared with other ones. At the end global minimum of entropy is chosen. This process can be simplified. Instead, each of possible decomposition at certain discrete wavelet transform step can be evaluated. Therefore, the minimum entropy could be calculated on the fly and not promising results could be discarded.

The proposed algorithmic development is not as trivial to parallelize as current solution. There are at least three approaches that are worth to be tested during implementation phase. Firstly, asynchronous primitives from the standard C++ library such as “std::async”, “std::future” and “std::promise” can be employed to perform this task. This approach leads to the C++ run-time execution determining whether it is beneficial to employ several threads for certain task. However, as it was shown before in subsection 4.3.4 Time comparison, one can easy be deceived by relying on the automatic tools that are supposed to make work parallel. The other solution requires employing producer-consumer architecture to solve described problem. The run-time execution should be able to determine whether more than one filter pairs can be processed at the same time. Therefore it should be evaluated whether there are hardware threads available to speed up whole process. Lastly, more sophisticated architecture connected with build custom thread pools can be employed and benchmarked.

Currently, the implementation of discrete wavelet transform is done in native C++ or even C language. To ensure the best possible performance this algorithm can be optimized by explicit applying vector instructions depending on the used CPU. There are two approaches to solve this issue. Firstly, existing implementation which is wrapping essential function such convolution and downsampling can be used. Secondly, single instruction, multiple data (SIMD) can be written in-place to leverage the optimal solution. SIMD is a type of parallel computing primitive which stands for multiple processing of elements performing the same operation on multiple points of data simultaneously. It is usually part of hardware design and can be directly accessed through the instruction set (ISA). The application can take advantage of SIMD when the same value is being added from a large number of data points. This is the case in many multimedia applications.

It is desired not only to compare described earlier solutions but also to profile them at first. This process can include profiling of generated machine code which is often the case when dealing with compiler intrinsics such as SIMD functions. For this particular “llvm-mca” tool can be used. This application statically measures the performance of machine code in a specific CPU. It is measured in terms of throughput as well as processor resource consumption. On the other hand, profiling can include analysis of overall time complexity, frequency and duration of function calls. Such tools, i.e. profilers use great variety of techniques that aim to collect data. Hardware interrupts, operating system hooks and performance are included. For this task tools “callgrind” (a subprogram of “valgrind”), “gprof” and “orbit” can be employed. The first one is able of recording call history among functions in a

program's run in the form of call-graph. Such data consists of number of instructions executed, their relationship to source lines, caller and callee dependencies among functions and the general number of such calls. The other tools, i.e. "gprof" and "orbit" (Open Run-time Binary Instrumentation Tool) work in similar fashion.

Bibliography

[1] 2d dwt applied 2 times to an exemplary image. https://en.wikipedia.org/wiki/File:Jpeg2000_2-level_wavelet_transform-lichtenstein.png. [Latest available: 13.07.2021].

[2] C++ reference. https://en.cppreference.com/. [Latest available: 12.07.2021].

[3] Cmake. https://cmake.org/. [Latest available: 12.07.2021].

[4] Entropy in information theory. https://en.wikipedia.org/wiki/Entropy_(information_theory). [Latest available: 27.07.2021].

[5] Image compression. https://en.wikipedia.org/wiki/Image_compression. [Latest available: 12.07.2021].

[6] Jpeg 2000. https://en.wikipedia.org/wiki/JPEG_2000/. [Latest available: 12.07.2021].

[7] Kakadu. https://kakadusoftware.com/. [Latest available: 12.07.2021].

[8] Lena, test image. https://en.wikipedia.org/wiki/File:Lenna_(test_image).png. [Latest available: 05.08.2021].

[9] pybind11. https://pybind11.readthedocs.io/en/latest/index.html. [Latest available: 30.07.2021].

[10] pytest. https://docs.pytest.org/en/6.2.x/. [Latest available: 30.07.2021].

[11] Python. https://www.python.org/. [Latest available: 12.07.2021].

[12] Pywavelets - wavelet transforms in python. https://pywavelets.readthedocs.io/en/latest/. [Latest available: 30.07.2021].

[13] Real time, wall-clock. https://en.wikipedia.org/wiki/Elapsed_real_time. [Latest available: 07.08.2021].

[14] Set of images used for evaluation of multiplierless reversible colour transforms. http://www1.hft-leipzig.de/strutz/Papers/Testimages/CT2/. [Latest available: 05.08.2021].

[15] Touradj Ebrahimi Athanassios Skodras, Charilaos Christopoulos. The jpeg 2000 still image compression standard. *IEEE Signal Processing Magazine*, 2001.

[16] J. O. Strömberg F. G. Meyer, A. Z. Averbuch. Fast adaptive wavelet packet image compression. *IEEE Transactions on Image Processing*, 2000.

[17] David A. Patterson John L. Hennessy. *Computer Architecture: A Quantitative Approach*. Elsevier, Incs, 2012.

[18] Martin Vetterli Kannan Ramchandran. Best wavelet packet bases in a rate-distortion sense. *IEEE Transactions on Image Processing*, 1993.

[19] Scott Meyers. *Effective Modern C++*. O'Reilly Media, 2014.

[20] Touradj Ebrahimi Peter Schelkens, Athanassios Skodras. *THE JPEG 2000 SUITE*. John Wiley & Sons, Singapore Pte. Ltd., 2009.

[21] M.V. Wickerhauser Ronald Coifman. Entropy-based algorithms for best basis selection. *IEEE Transactions on Information Theory*, 1992.

[22] Roman Starosolski. Skipping selected steps of dwt computation in lossless jpeg 2000 for improved bitrates. *Plos one*, 2016.

[23] Roman Starosolski. A practical application of skipped steps dwt in jpeg 2000 part 2-compliant compressor. *Springer Link*, 2018.

[24] Roman Starosolski. Reversible denoising and lifting based color component transformation for lossless image compression. *Springer Link*, 2019.

[25] Roman Starosolski. Hybrid adaptive lossless image compression based on discrete wavelet transform. *Entropy*, 2020.

[26] Bjarne Stroustrup. *The C++ Programming Language*. Pearson Education, 2013.

[27] Bjarne Stroustrup. *A Tour of C++*. Pearson Education, 2013.

[28] Tilo Strutz. Multiplierless reversible color transforms and their automatic selection for image data compression. *IEEE Transactions on Circuits and Systems for Video Technology*, 2013.

[29] D. Sundararajan. *Discretewavelet Transform: A Signal Processing Approach*. John Wiley & Sons, Singapore Pte. Ltd., 2015.

[30] Anthony Williams. *C++ Concurrency in Action*. Manning Publications, 2019.

Appendices

Technical documentation

At the listing 11 there are available both “FILTERS” dictionary containing needed options to perform lossless image compression of given input using Kakadu software and wrapper of running such command from the subshell with the output decoding. The keys from mentioned before dictionary correspond to their “pywt” names. More complex filters require more verbose command line input. The output of Kakadu is searched with specific regular expression to retrieve the bitrate of compressed image and then converted to floating point number. On the other hand, at the listing 12 the generation process of possible DWT decompositions is depicted. In this example only five level of DWT chain is calculated. “V(-)” stands for vertical decomposition, “H(-)” for horizontal and “B(-:-:-)” for both. It is worth pointing out that further decomposition can be applied at the certain level by filling “-” in the brackets.

The exemplary command using arbitrary DWT decomposition is shown below.

```
./bin/Linux-x86-64-gcc/kdu_compress -i ./img/02_Schluesselfelder_Schiff.ppm
-o ./image.jpx Creversible=yes Clevels=5
Cdecomp="B(-:-:-),B(-:-:-),B(-:-:-),B(-:-:-),H(-)".
```

Other example involves using not Part 1 compliant filter, i.e. Haar

```
./bin/Linux-x86-64-gcc/kdu_compress -i ./img/02_Schluesselfelder_Schiff.ppm
-o ./image.jpx Catk=2 Kkernels:I2=R2X2 Clevels=5
Cdecomp="B(-:-:-),B(-:-:-),B(-:-:-),H(-),V(-)".
```

```python
import os
import re
import subprocess
from itertools import product
from pathlib import Path


BASE = "./bin/Linux-x86-64-gcc/kdu_compress"
FILTERS = {
    'bior2.2': 'Creversible=yes',
    'haar': 'Catk=2 Kkernels:I2=R2X2',
    'bior2.6': 'Catk=2 Kextension:I2=SYM Kreversible:I2=yes '
               '"Ksteps:I2={4,-1,4,8},{4,-2,4,8}" '
               'Kcoeffs:I2=0.0625,-0.5625,-0.5625,0.0625,'
               '-0.0625,0.3125,0.3125,-0.0625'
}


def run_cmd(path, cmd):
    imgs = f"-i {path} -o ./image.jpx"
    cmd_base = f"{BASE} {imgs}"
    res = subprocess.run(f"{cmd_base} {cmd}",
                         stdout=subprocess.PIPE, shell=True)
    out = res.stdout.decode('utf-8')
    try:
        out = re.search(R"(Layer.+\n\D+)([\d\.]+)", out, re.M).group(2)
    except AttributeError:
        out = "inf"
    return float(out)
```

Listing 11: Python function used to call Kakadu with desired parameters

```python
def test_kdu_dwt_comp(path, dwt_filter):
    vertical = ['V(-)']
    horizontal = ['H(-)']
    both = ['B(-:-:-)']
    comb = [vertical, horizontal, both]
    comps = [*product(comb, repeat=5), ]
    cmds = []
    level = "Clevels=5"
    for comp in comps:
        comp = ','.join(x[0] for x in comp)
        cmds.append(f'{FILTERS[dwt_filter]} {level} Cdecomp="{comp}"')
    results = dict()
    for cmd in cmds:
        results[cmd] = run_cmd(path, cmd)
    min_bitrate = min(results, key=results.get)
    with open(f"./ref_results/{Path(path).stem}.txt", "a") as f:
        f.write(f"best: {results[min_bitrate]}\n")


def test_kdu_filter(path):
    results = dict()
    for dwt_filter, cmd in FILTERS.items():
        results[dwt_filter] = run_cmd(path, cmd)
    with open(f"./ref_results/{Path(path).stem}.txt", "a") as f:
        f.write(f"ref: {results['bior2.2']}\n")
    return min(results, key=results.get)


def main():
    for file in os.scandir('./img'):
        print(file.path)
        dwt_filter = test_kdu_filter(file.path)
        test_kdu_dwt_comp(file.path, dwt_filter)
        print("")


if __name__ == '__main__':
    main()
```

Listing 12: Python script used to test various configurations using Kakadu

List of abbreviations and symbols

JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
PACSs	Picture Archiving and Communication Systems
DICOM	Digital Imaging and Communications in Medicine
ISO	International Organization for Standardization
DCT	Discrete Cosine Transform
DWT	Discrete Wavelet Transform
SS-DWT	Skipped Steps Discrete Wavelet Transform
LL	Low and then low-pass filtered image
LH	Low and then high-pass filtered image
HL	High and then low-pass filtered image
HH	High and then high-pass filtered image
ppi	pixels per inch
JP2	standard JPEG 2000 file extension
J2K	extension used for storing code-stream JPEG 2000 data
MJ2	Motion JPEG 2000
JPWL	JPEG 2000 Wireless
DSP	Digital Signal Processing
SFINAE	Substitution Failure Is Not An Error
RAII	Resource Acquisition Is Initialization

Contents of attached CD

The thesis is accompanied by a CD containing:

- thesis (`pdf` file),
- source code of applications,
- data sets used in experiments.

List of Figures

List of Tables

List of listings