Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування факультету)

Кафедра обчислювальної техніки
(повна назва кафедри)

Пояснювальна записка

до магістерської кваліфікаційної роботи

на тему **Методи та засоби ущільнення багатоканального відеозображення цифрових телеканалів для систем комп'ютерного моніторингу**

Виконав: студент 2 курсу, групи 1КІ-19м
спеціальності:
123 «Комп'ютерна інженерія»
(шифр і назва напряму підготовки, спеціальності)

Жилін М. В.
(прізвище та ініціали)
Керівник к.т.н доц. Крупельницький Л. В.
(прізвище та ініціали)

Вінниця
2020 р.

АНОТАЦІЯ

Дана магістерська кваліфікаційна робота присвячена розробці спеціалізованого методу й програмного забезпечення для комп’ютерної системи цифрового телевізійного мовлення, задачею якого є ущільнення багатоканального відеозображення.

Спецілазоване програмне забезпечення являє собою консольну програму, за допомогою команд в якої можна виконати ущільнення. Програмне забеспечення розроблено на основі стандарту H.264 та реалізує запропонований метод багатоканального узільнення відеопотоку. Команди дозволють задати формат та назву вихідного файлу, а також ступінь ущільнення.

У роботі розроблено: специфічний метод ущільнення багатоканального відео зображення, а також програмне забеспечення для систем комп'ютерного моніторингу.

ABSTRACT

This master's thesis is devoted to the development of specific software for the computer system of digital television broadcasting, the task of which is the compression of multi-channel video.

Specialized software is a console program with the help of commands in which you can perform seals. The software is developed on the basis of the H.264 standard and implements the proposed method of multi-channel video streaming. The commands will specify the format and name of the source file, as well as the degree of compression.

In the thesis developed: a specific method of compression of multi-channel video. as well as software for computer monitoring systems.

ЗМІСТ

ВСТУП

На сьогоднішній день розвиток обчислювальної техніки йде швидкими темпами – постійно зростає частота і продуктивність процесорів, збільшуються обсяги пам'яті і прискорюється час доступу до неї. При такому бурхливому зростанні швидкостей різних пристроїв виникають проблеми передавання й зберігання різноманітних даних. Це відбувається через те, що особливістю більшості типів даних є їх надлишковість.

При передачі та збереженні великих обсягів інформації надмірність відіграє негативну роль, оскільки вона не тільки призводить до збільшення часу передачі і функціональної надійності передачі інформації та її зберігання, а й до зростання сукупної вартості. В зв'язку з цим на сьогоднішній день для забезпечення ефективності передачі великих обсягів інформації та зберігання широко використовуються різноманітні способи ущільнення.

Існує велика кількість програм, які реалізують різноманітні кодеки ущільнення відеопотоку з втратами та без. Але з новими удосконаленнями залишається актуальність проблеми ущільнення багатоканального цифрового відеозображення. Дана розробка, в першу чергу, актуальна для використання в Національній раді України з питань телебачення і радіомовлення та її регіональних представництв. Телекомпанії, які працюють на території України повинні працювати з великими потоками даних, які з кожним роком все збільшуються. Але існуючі методи ущільнення недосконалі для багатоканального відеозображення. Тому є необхідність у вдосконаленні методів ущільнення для більшої економії простору на носіях пам’яті та реалізації в багатоканальному режиму.

Актуальність проблеми полягає в необхідності розробки нових методів та алгоритмів ущільнення відео-аудіо потоків одночасно для декількох цифрових каналів, що передаються в багатоканальному масиві цифрового телевізійного мовлення.

Об’єктом дослідження є процеси моніторингу багатоканального телевізійного мовлення за допомогою спеціальних комп’ютерних систем.

Предметом дослідження є методи та алгоритми ущільнення відеозображень, що враховують специфіку багатоканальних сигналів.

Метою дослідження є збільшення ступеня ущільнення багатоканального відеозображення і зменшення обчислювальних витрат за рахунок розробки спеціалізованого методу та його програмних засобів ущільнення.

Для досягнення цієї мети **потрібно вирішити такі завдання**:

— проаналізувати наявну інформацію щодо методів ущільнення;

— розробити специфічні методи, які враховують специфіку ущільнення багатоканального відеозображення;

— розробити і реалізувати алгоритм та програмне забезпечення для опрацювання багатоканальних відеосигналів;

— обґрунтувати техніко-економічні показники і ефективність виконаної розробки.

Методи дослідження, що використовуються в роботі:

— системний аналіз, який застосовується для дослідження методів ущільнення ентропійного кодування, квантування, кольорової субдискретизації, а також компенсації руху;

— об’єктно-орієнтовані методи програмування, які використовуються для створення програмного забезпечення ущільнення відеопотоку;

— методи порівняльного комп’ютерного тестування з системою оцінки ефективності запропонованих алгоритмів і програм.

Наукова новизна магістерської кваліфікаційної роботи:

— дістав подальшого розвитку гібридний метод ущільнення відеозображень, який у випадку багатоканальних цифрових відеопотоків збільшує ступінь ущільнення та зменшує час опрацювання даних.

Практичне значення отриманих результатів:

— розроблено алгоритм та програму для багатоканального ущільнення відеозображення, які мають самостійне значення;

— розроблено програмне забезпечення, інтегроване в систему моніторингу центрального та регіонального цифрового телерадіомовлення.

Апробація результатів: відбулась на XLVIII Науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії ВНТУ.

Публікація за темою роботи: Крупельницький Л. В., Жилін М.В., Самко В.В. Комп’ютерна система моніторингу телевізійного мовлення / Л.В. Крупельницький, М.В. Жилін, В.В. Самко // Тези доповіді XLVIII Науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії . Вінниця, – 2019 р,.

1 АНАЛІЗ І КЛАСИФІКАЦІЯ МЕТОДІВ УЩІЛЬНЕННЯ ВІДЕОЗОБРАЖЕННЯ

Характерною особливістю більшості типів даних є їх надлишковість. Коли мова йде про зберігання або передачу даних, то надмірність відіграє негативну роль, оскільки вона призводить до зростання вартості зберігання та передачі інформації. У зв'язку з цим, постійно виникає проблема зменшення надмірності або ущільнення даних.

Ущільнення базується на усуненні надлишку інформації, яка міститься у вихідних даних. Всі алгоритми ущільнення можна розділити на працюючі без втрати і з втратою інформації. При використанні перших алгоритмів дані після подвійного перетворення (компресії і декомпресії) є ідентичні вхідним, а при застосуванні других спостерігаються деякі розходження. У багатьох випадках втрати неприпустимі, наприклад, при ущільненні тексту, числових даних і т.п. При цьому коефіцієнт ущільнення обмежений і найчастіше невеликий обсяг даних зменшується в 1,5 – 6 разів. Деякі дані взагалі не піддаються ущільненню без втрат (коефіцієнт менше одиниці). При використанні алгоритмів ущільнення з втратами подвійне перетворення приводить до зміни вихідної інформації, що, як правило, спричиняє погіршення якості. Проте вони забезпечують набагато вищий ступінь ущільнення: у 5 – 10 разів для звуку і в 5 – 200 – для відео [2].

При ущільненні з втратами відбувається видалення з потоку даних від 80% до 99% і більше інформації. Природно, якщо просто знизити частоту дискретизації звуку в 10 разів, то істотно постраждає не тільки тембр, але навіть розбірливість мови. Якщо ж з екрана комп'ютера або телевізора просто викинути 99 з кожних 100 точок, розрізнити на ньому що-небудь також виявиться досить важко. В ідеалі видаленню повинна підлягати тільки та інформація, що не сприймається людським слухом або зором. Тому при розробці алгоритмів ущільнення один з найбільш важливих етапів – створення адекватної психофізіологічної моделі визначеного органа чуттів (у даному випадку вуха або ока), а точніше – системи, що складає з відповідного органа і мозку. Саме з цим пов’язана висока вартість розробки

подібних алгоритмів [2].

1.1 Призначення і класифікація методів цифрового ущільнення інформації

Метою цифрового ущільнення є скорочення об’єму інформації, що описує телевізійне зображення, без помітного для ока погіршення його якості (рис. 1.1). Обробка повідомлення перед подачею в канал зв’язку називається кодуванням джерела. Скорочення можливе завдяки значній інформаційній надлишковості, і якостям людського зору, що не сприймає окремі деталі зображення [5].

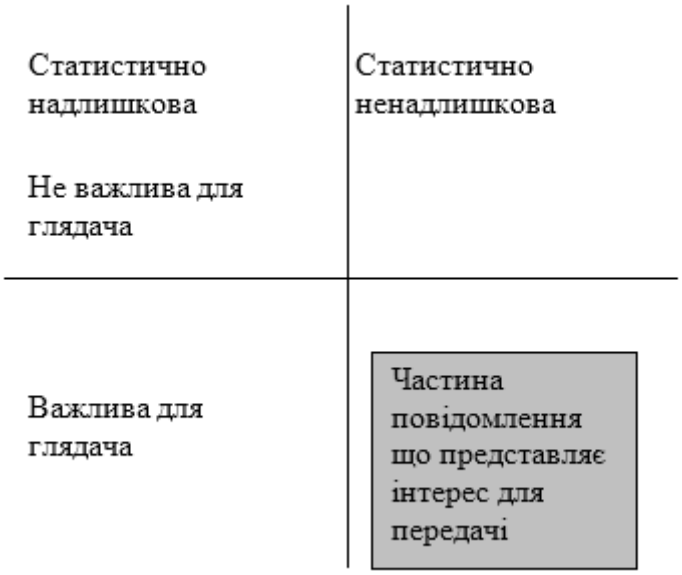

Рисунок 1.1 — Розділення площини повідомлення на сегменти з урахуванням їх надлишковості

Якщо умовно розділити площину повідомлень вертикальною лінією на ліву частину, статистично надлишкову, і праву, статистично не надлишкову. А горизонтальною лінією на верхню півплощину, яка містить інформацію, не важливу для глядача (це так звана, психофізіологічна надлишковість, обумовлена властивостями зору), і нижню півплощину з важливою інформацією, то інтерес для передачі по каналу складає інформація, розміщена в правій нижній частині площини повідомлення [5].

Поширені методи цифрового ущільнення представлені на рис. 1.2.

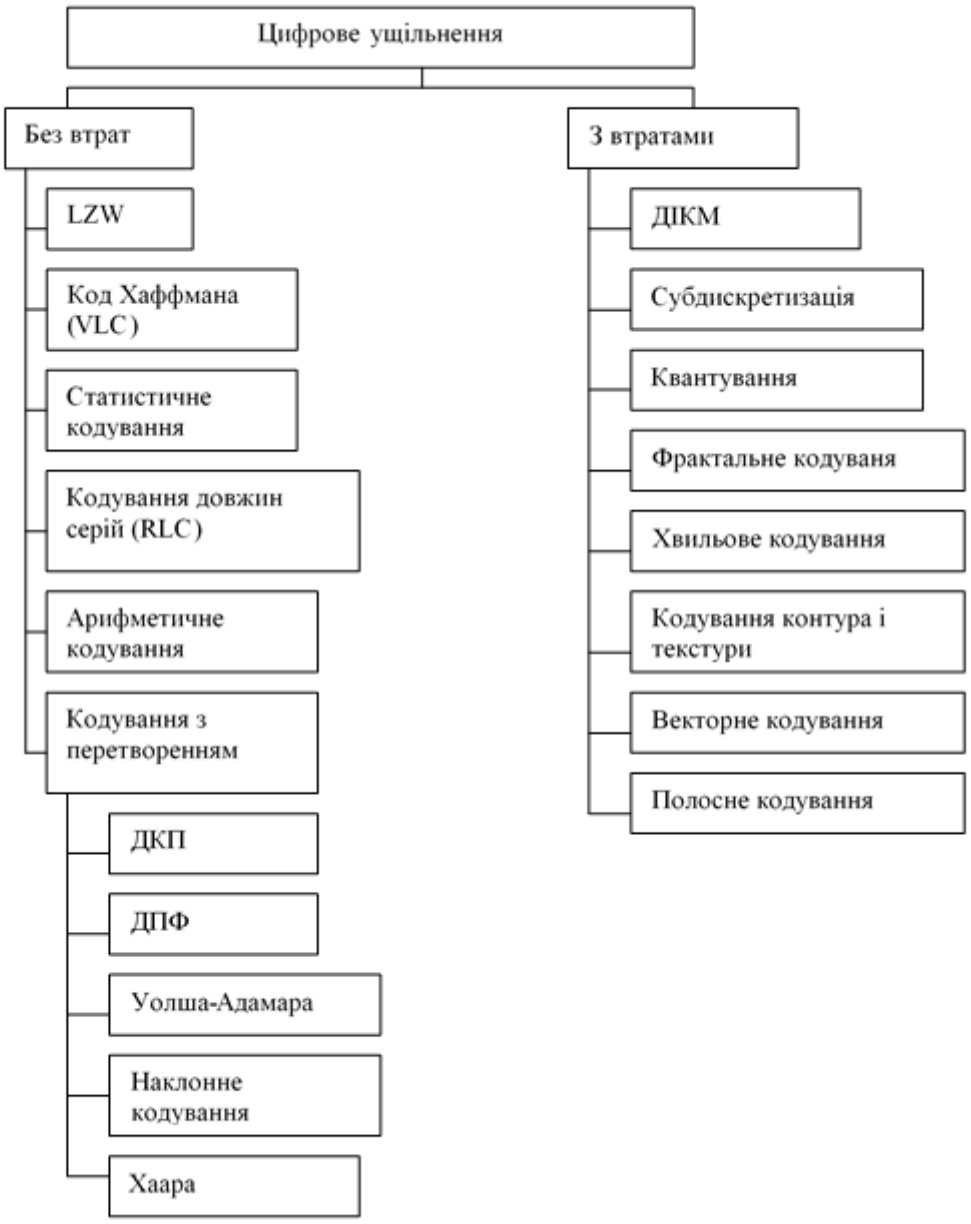

Рисунок 1.2 — Класифікація методів цифрового ущільнення

Усунення неважливої частини є безповоротним процесом, який характеризується втратою інформації, порівняно з усуненням статистичної надлишковості, не зв’язаної з втратою інформації. В цифровому мовленні використовуються обидві групи методів.

Алгоритм скорочення статистичної надлишковості не дають ущільнення більше 3:1. В комп’ютерній техніці для архівації даних давно використовують кодування LZW (Лемпеля-Зіва-Уолша), який не приводить до втрати інформації і придатне для ущільнення даних будь-якого типу. Кодування починається з

побудови кодового дерева із коротких блоків вхідного потоку даних. Потім перевіряється, чи містять наступні блоки даних кодові слова із дерев. Якщо так, то будуються нові кодові слова як комбінація існуючих і відсилаються тільки їх адреса. Цей алгоритм дає можливість ущільнювати довгі послідовності до коротких адрес. Така побудова кодів слів від кодових слів приводить до ущільнення довгих послідовностей в короткі адреси. Метод вимагає громіздких розрахунків і не використовується в цифровому телебаченні [5].

Алгоритми із змінною довжиною кодового слова (VLC) і самий відомий з них – код Хаффмана. Він присвоює словам з найбільшою ймовірністю появи короткі кодові комбінації, а більш рідким символам – більш довгі (по цьому ж принципі побудована азбука Морзе). Принцип побудови коду відбувається наступним чином: нехай алфавіт джерела нараховує чотири символи «a», «b», «c», «d» з ймовірністю появи відповідно 0,5, 0,25, 0,125, 0,125. Якщо кожному із символів присвоїти двохбітове значення 00, 01, 10, 11, середня довжина кодового слова складе, очевидно, 2 біти на символ. Присвоюється символу «a» значення 0, символу «b» – значення 10, символам «c» і «d» – значення відповідно 110 і 111. Неважко розрахувати, що в середньому для передачі одного символу витрачається 1*0,5+2*0,25+2*3*0,125=1,75 біт. Хоча максимальна довжина символу зросла, число бітів, які потрібні для передачі повідомлення, скоротилось. По своїй ефективності код наближається до теоретичної границі затратів бітів і тому називається ентропійним. Адаптивна версія коду Хаффмана застосовується в тому випадку, коли ймовірність появи кодових слів змінюється в процесі передачі [5].

До групи ентропійних відносять і так званий арифметичний код. Процедура кодування полягає в тому, що всій сукупності символів повідомлення ставиться у відповідність інтервал [0, 1], який розбивається на проміжки, відповідні вихідним ймовірностям символів, і це розбиття повідомляється декодеру. Після приходу нового символу інтервал перераховується на нові межі, які відповідають імовірності появи цього символу, і знову розбивається пропорційно вихідним імовірностям. З приходом кожного нового символу – розмір інтервала зменшується, причому в суровій відповідності з ймовірностями символів.

Символи, які зустрічаються більш частіше менше звужують інтервал, ніж ті, що зустрічаються рідше, і додають менше бітів в код інтервалу. По закінченню циклу кодування формується деякий, значно вужчий інтервал, що однозначно характеризує послідовність символів, що передаються і він легко може бути установлений в кодері по будь-якому числу із цього інтервалу [5].

Кодування довжин серій (RLC – Run Lngth Coding) – це присвоєння довгим безперервним послідовностям однакових бітів окремих кодових слів. Цей метод ефективний при дискретно-косинусному перетвореннні, коли значна кількість коефіцієнтів приймає нульові значення [5].

1.2 Методи ущільнення цифрової відеоінформації

1.2.1 Усунення просторової і часової надлишковості відеосигналу

Просторова надлишковість зображення зумовлена присутніми у відеокадрі значних за розміром однотонних однаково закрашених ділянок. Відліки відеосигналу в сусідніх точках на таких ділянках практично однакові або змінюються незначно в площині зображення, звідси слідує, що присутні тільки низькочастотні складові двомірного просторового спектру, і перетворення в спектральну область дозволить значно скоротити число бітів, необхідне для відображення кадру зображення. Найбільш ефективним перетворенням являється ДКП (дискретно-косинусне перетворення). Зображення розбивається на невеликі блоки пікселів і ДКП послідовно застосовується до кожного такого блоку. Проводились досліди по розбтиттю 4x4, 8x8, 16x16 пікселів, які показали, що розбиття 8x8 є найкращим компромісом між точністю перетворення (мінімальною середньоквадратичною помилкою) і необхідним об’ємом обчислень [6].

Перетворення у відповідність 64 відлікам цифрового сигналу 64 коефіцієнти при гармоніках просторової частоти, що являються базисними функціями ДКП. В лівому верхньому куті опиняється коефіцієнт при нульовій гармоніці, інакше кажучи, постійна складова чи середня яскравість блока, сусідні позиції займають коефіцієнти при гармоніках невисокого порядку, а з переміщенням до правого кута порядок гармонік росте і їх інтенсивність зменшується (рис.1.3). В більшості

випадків більша частина коефіцієнтів після квантування може бути відкинута [6].

12	17	15	8	3	11	1	10
15	8	12	11	6	4	10	1
6	1	10	5	8	12	4	8
11	12	15	5	4	10	6	7
11	14	11	2	8	9	3	6
14	7	11	13	2	6	9	6
13	18	15	11	6	1	6	6
11	6	8	10	4	10	5	9

а) вихідний блок відліків

⇨

10	12	8	1	1	0,3	0,2	1
12	4,5	1,1	1,4	0,2	0,2	0,3	0,4
4	1,2	1,1	0,5	0,1	0,4	0,1	0,2
1	0,3	0,4	1,1	0,4	0,2	0,4	0,1
0,1	0,1	1	0,3	0,2	0,4	0,2	0,2
0,2	0,1	0,2	0,3	0,2	0,3	0,1	0,1
0,4	0,1	0,5	0,4	0,3	0,3	0,2	0,1
0,2	0,3	0,1	0,3	0,2	0,4	0,2	0,2

б) блок коефіцієнтів

Рисунок 1.3 — Нормування коефіцієнтів при дискретно-косинусному перетворенні

Часова надлишковість зображення проявляється у відносно малій відмінності двох послідовних кадрів ТВ зображення. Як правило, зображення представляє собою нерухомий задній план і відносно повільне переміщення об’єктів на передньому плані. Можна скоротити масив інформації, що передається, якщо вміст одного кадру, що називається опорним, передавати цілим, а замість наступного кадру передавати тільки інформацію про об’єкти які перемістилися. Цей спосіб був значно удосконалений, коли застосували прогнозування кадру на основі одного або декількох попередніх та наступних - по каналу почали передавати різницю між поточним і прогнозованим кадрами. Відмітимо, що для підвищення точності прогнозування в якості опорного беруть не оригінальний попередній кадр, а відновлений в декодері, тобто в кодері для цієї мети відновлюється декодер [6].

Для покращення результатів прогнозування застосовують процес, який називають компенсацією руху. Зображення в поточному кадрі розбивається на невеликі блоки і для кожної такої ділянки шукають її найбільш ймовірне місце знаходження в попередньому кадрі, порівнюючи її з усіма ділянками такого ж розміру в заданій області пошуку. Пошук ведеться за критеріями мінімальної

абсолютної помилки, або мінімальної середньо-квадратичної помилки. Цей процес називається спряженням блоків (рис.1.4). Існують і більш ефективні алгоритми розрахунку спряження, наприклад, метод фазового спряження. В цьому методі блоки відліку із сусідніх полів за допомогою ДПФ переводяться в частотну область, значення однойменних коефіцієнтів віднімаються і над різницею виконується зворотне перетворення Фур'є. Якщо уривок містить рух, в отриманій кореляційній поверхні з'являються всплески, положення яких дає точну інформацію про направлення і переміщення, але втрачається інформація про просторове положення області, що переміщується. Для встановлення інформації доводиться проводити великий об'єм обчислень [6].

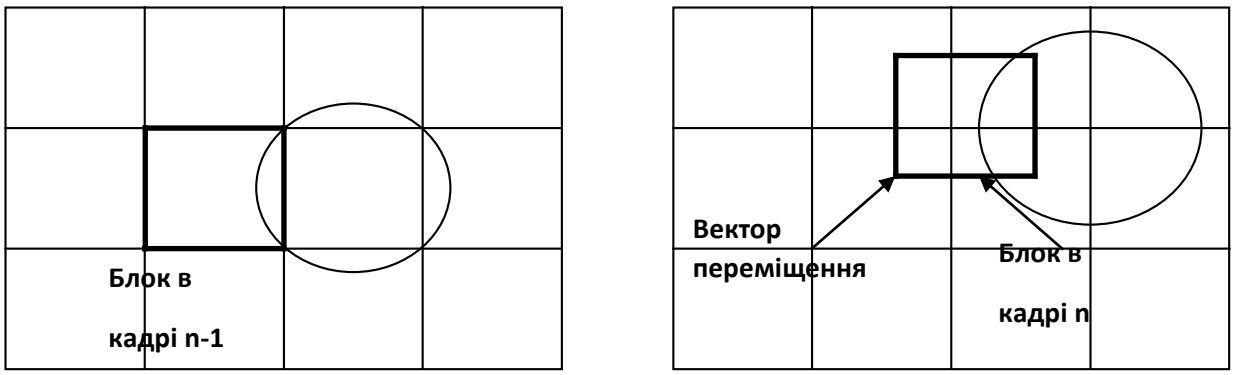

Рисунок 1.4 — Спряження блоків при компенсації руху

В більшості форматів цифрового ущільнення, що застосовуються, блок відліку називається макроблоком і його розмір складає 16x16 пікселів. Чим більший розмір блоку, тим ефективніше ущільнення, але складніше знайти схожість. Для маленьких блоків більш ймовірно знайти блоки із схожими параметрами, але степінь скорочення об'єму даних буде невисокою. Розміри області пошуку визначають максимальну швидкість переміщення об'єкта в кадрі, яка може бути скомпенсована. Кращі сучасні кодери забезпечують точність зв'язку 0,5 пікселів. Знайдені в опорному кадрі блоки, що підходять, поєднуються з блоком, що аналізується, в поточному кадрі і далі передається по каналу різниця між відліками. Завдяки компенсації руху ця різниця має тепер мінімальне значення. Крім того, обчислюється величина і напрямок переміщення – вектор переміщення,

і також передається в декодер, який на основі отриманих даних легко відновлює параметри поточного блоку. Даний алгоритм описує тільки рух переносу і неефективний при поворотах, зміні масштабу та інших більш складних рухах, але не дивлячись на це, дає хороші поточні результати [6].

1.2.2 Попереднє фільтрування та заглушення шуму

Процес усунення надлишковості показує, що для подальшого ефективного ущільнення необхідно усунути із вихідного сигналу всі фактори, які перешкоджають скороченню надлишковості телевізійного зображення. В першу чергу – це шуми відеотракту і шуми телевізійної стрічки, які проявляються на всьому зображенні. Джерелом шумів у відео тракті можуть бути тепловий шум електронних пристроїв, порогові шуми супутникових ЧМ приймачів, завади декодування аналогових композитних сигналів, шуми відеозапису. Один із вкрай неприємних для цифрової компресії дефектів вихідного зображення – неподавлений залишок несучої кольоровості після аналого-цифрового перетворення, він проявляється у високочастотній області , яка зазвичай слабо заповнена в компонентному сигналі. Негативно впливає дефект цифрової компресії вихідного зображення – не подавлений залишок несучої кольоровості після аналого-цифрового перетворення, він проявляється у високочастотній області. Також негативно впливають дефекти кінострічки – забруднення, подряпини, шуми зернистості, що проявляються при збільшенні кінокадрів. Необхідно уникати небажаних рухів, що додають у сигнал перешкоди [6, 7].

Вплив цих факторів видно після дискретно-косинусного перетворення (ДКП) великої кількості ненульових коефіцієнтів в області високих просторових частот, які кодер не може відрізнити від елементів зображення. Шуми також знижують схожість послідовними кадрами, що збільшує використання кількості бітів на кодування міжкадрових різностей. У результаті кодеру, не вистачає ресурсів на кодування зображення, і якість декодованого зображення у глядача нижче, ніж у вхідного матеріалу [7].

В таких умовах необхідно піддавати вхідні сигнали кодерів ущільненню,

спеціальній передобробці – передфільтрації, що включає шумоподавлення і різні види просторової фільтрації. У деяких випадках при високому ступені ущільнення ефективніше виконати двомірну низькочастотну фільтрацію, втрачати в чіткості, але значно знизити кількість дефектів зображення через вплив небажаних високочастотних компонентів на вході. В загальному випадку хороший результат дає комбінація рекурсивно-адаптивної фільтрації і трьохмірної медіанної фільтрації. Перша ефективно придушує тепловий шум шляхом усереднення близьких значень відеосигналу в послідовних кадрах, а друга усереднює відліки, що знаходяться вище, нижче, ліворуч і праворуч, до і після поточного відліку відеосигналу, і таким чином, добре справляється з імпульсними та іншими перешкодами [7].

1.2.3 Квантування і кодування коефіцієнтів ДКП

Самі по собі операції прогнозування і ДКП не приводять до якого-небудь скорочення числа бітів, оскільки кількість відліків на кадр і розрядність квантування залишаються без змін. Більше того, розрядність коефіцієнтів зростає з 8 до 11...14 бітів при перемноженні на дріб в процесі обчислення. Єдина, але дуже важлива властивість сигналу, якого досягли на даному етапі – різниця відліків дуже мало корельована, після перетворення коефіцієнти при гармоніках просторової частоти є невеликими і не вимагають багато бітів для передачі по каналу. Квантування (а точніше переквантування) коефіцієнтів є першою операцією ущільнення, де проявляються досягнення підготовчих етапів.

Квантування полягає в діленні значень коефіцієнта на деяке число (N>1) і заокругленню результату до найближчого цілого числа, в результаті чого зменшується число бітів, яке необхідне для їх кодування. Ті з коефіцієнтів, які після ділення, мають величину, менше 0,5, будуть замінені нулями. Одночасно з ліквідацією кожного молодшого біта вдвічі збільшуються шуми квантування [7].

У звичайній імпульсно-кодовій модуляції (ІКМ) зменшення розрядності приводить до збільшення шумів квантування на всіх частотах. Коефіцієнти ДКП визначають енергію сигналу на різних частотах і з’являється можливість змінювати

праметри квантування деференційовано в різних частотах, враховуючи різну чутливість зору до різних просторових частот (рис. 1.5). Залежність спотворень залежить також від яскравості даної ділянки зображення, степені його однорідності, довжини уривка [7].

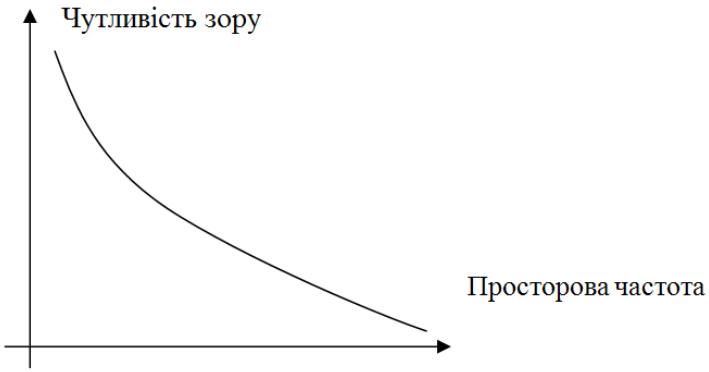

Рисунок 1.5 — Залежність чутливості зору від просторової частоти

Використання при квантуванні зважуючих множників, що враховують ступінь помітності спотворення, називається перцептуальним зважуванням і широко практикується в цифровому ущільненні. Постійна складова зазвичай квантується окремо і з більш високою точністю, більш високочастотні коефіцієнти ділять все більш зростаючі числа. Сукупність дільників для всіх коефіцієнтів блока називається матрицею квантування і передається декодеру разом з компресованим сигналом. В декодері кожен коефіцієнт множиться на відповідний дільник із матриці, відновлюючи своє (округлене) значення [7].

Для типового цифрового зображення з прогресивною розгорткою ненульові коефіцієнти зосереджені в лівому верхньому куті матриці. Найбільш ефективний метод зчитування значень коефіцієнтів – зигзагоподібне сканування: із верхнього лівого в правий нижній кут (рис. 1.6), при якому спочатку зчитуються всі ненульові, а потім нульові коефіцієнти. Для останніх застосовується описаний вище мотод кодування довжин серій, при яких кожній безперервній послідовності однакових символів приписуються окремі кодові слова [7].

Статистична ймовірність появи різних значень коефіцієнтів неоднакова: деякі з’являються дуже часто, інші рідко. Ця особливість може бути використана

для подальшого скорочення бітів, якщо більшим значенням, що зустрічаються частіше, присвоїти більш короткі послідовності символів, а тим, що рідше – більш довгі. Цю задачу вирішують коди із змінною довжиною, про які ми згадували вище [7].

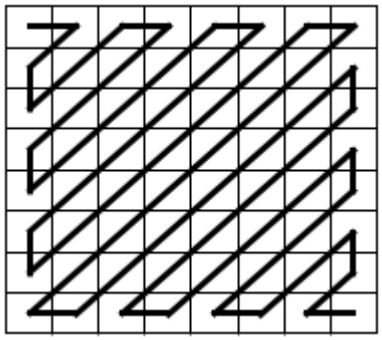

Рисунок 1.6 — Зигзаподібне сканування коефіцієнтів ДКП

В залежності від сюжету зображення і швидкості руху об’єктів, частина надлишковості, що усувається, змінюється і середнє число бітів, яке приходить на один відлік, може варіюватись в широких межах. В таких же межах буде змінюватись і швидкість цифрового потоку на виході кодера ущільнення. В той же час більшість каналів зв’язку розраховані на передачу сигналів з постійними швидкостями. Як завжди в таких випадках, допомагає буферна пам’ять. Буфер накопичує надлишкові біти в критичні для кодера моменти (зміна сюжету, швидкий рух) і віддає їх в періоди відносного спокою, коли канал виявляється недозавантаженим поточними даними. Прогнозований і механізм зворотнього зв’язку у випадку переповнення або, навпаки, спустошення буфера: змінюючи адаптивно коефіцієнти матриці квантування, буфер може в визначених межах управляти кількістю бітів, що поступають, і швидкістю потоку, за рахунок відповідного збільшення або зменшення кроку квантування і пов’язаних з ним шумів квантування [7].

1.3 Вибір оптимальних методів для багатоканального ущільнення

В 1982-1990 рр. групою експертів по кодуванню нерухомих зображень (Joint Photographic Expert Group), створеною разом МОС, МЕК і МСЕ, був розроблений ,

а потім прийнятий одноіменний стандарт кодування нерухомих зображень. Основною метою було поставлено скорочення необхідного об’єму даних для передачі високоякісного зображення (до 0,25 біт/пікс. для зображення, «що не відрізняється» 1 біт/пікс. для якості і 4 біт/пікс. для зображення «що відрізняється» від оригінала, при вихідній точності квантування 8 біт/пікс.) [12].

В процесі дослідів були випробувані багато алгоритмів цифрового ущільнення і вибрано ДКП як таке, що забезпечує найкращу якість при мінімальній швидкості цифрового потоку. В стандарті JPEG вперше були введені технічні рішення, які ввійшли в подальші стандарти: обробка не вихідних кольорів, а матричних сигналів яскравості і сигналів з різними кольорами, проріджування останніх вдвічі до формату “4:2:0”, вибір розміру блоків ДКП 8x8 пікселів, зигзагоподібне сканування коефіцієнтів, роздільне квантування постійної складової і решти коефіцієнтів ДКП, ентропійне кодування. По зрозумілим причинам, була відсутня тільки компенсація руху [3].

Наступний стандарт, що представляє інтерес, під назвою H.261 був розроблений в 1988-1990 рр. Міжнародним комітетом по телеграфії і телефонії (. попередником теперішнього МСЕ-Т). Стандарт був призначений для передачі відеозображень (відеотелефонії, відеоконференції) з швидкістю n × 64 біт/с, де n = 1...30, формат зображення не міг перевищувати CIF. В цьому стандарті уже використовувалося кодування з прогнозування і два типи кадрів: I- і P-кадри. Передбачення руху здійснювалось порівнянням з попереднім кадром, в якості критерія поєднання був прийнятий мінімум абсолютної помилки. Початкове вікно складало прямокутну ділянку розміром 64 × 48 пікселів, пошук міг здійснюватись одним із трьох способів:

— повним перебором всієї області пошуку (повільний спосіб);

— шляхом послідовного звуження вікна, поки не досягається необхідна точність спряження;

— ієрархічною оцінкою руху, при якій формується декілька послідовних версій опорного і поточного кадру зі все більш низькою роздільною здатністю і перераховується до вихідного рівня шляхом послідовної модифікації векторів

переміщення. Як і в сучасних стандартах, макроблок розміром 16 × 16 пікселів містив 4 блоки відліків сигналу яскравості і по одному блоку сигналів з різними кольорами [3, 4].

Стандарти H.261 і JPEG стали основою розробки MPEG. Робота в цьому напрямі була продовжена, і в 1996 р. МСЕ-Т прйняв для тих же застосувань H.263, який більш пристосований для низькошвидкісного кодування. В цьому стандарті підтримуються формати від SQCIF до 16CIF (1408 × 1152 пікс.), присутні всі три види кадрів, I, P і B, область пошуку спряження – не обмежена, точність спряження 0,5 пікселів До основних покращень, в порівнянні з H.261 відносяться: необмежений діапазон зміни векторів переміщеня (на межі вони можуть вказувати за межі області зображення), арифметичне кодування квантованих коефіцієнтів ДКП, покращене прогнозування (прогнозуються 4 блоки векторів 8 × 8 замість розмірності 16 × 16 пікселів), кодування PB-кадрів (послідовний P-кадр і B-кадр можуть кодуватися разом як один PB-кадр). Кодек H.263 вийшов за межі швидкості 64 біт/с і став кодеком широкого застосування для низькошвидкісного кодування [3].

У більшості форматів ущільнення відео застосовується алгоритм, аналогічний JPEG: уся площа кадру розбивається на квадратики 8x8 пікселів, що потім ущільнюється з використанням дискретного косинусного перетворення. Кратність ущільнення цього алгоритму при максимальній якості приблизно дорівнює п'яти. Можлива і більш висока кратність, але при цьому страждає якість картинки: за рахунок утрати вищих гармонік поблизу різких переходів з'являється ореол, а в області градієнта чітко проступають квадратики [3].

MPEG-1 використовується в основному для Video-CD. Розмір зображення відповідає розміру CIF, оскільки в якості носія інформації був обраний CD-диск, а на момент виходу стандарту CD-ROM приводи були одношвидкісними, тому швидкість відеопотоку у форматі MPEG-1 обмежена 150 Кб / с. Ущільнення в ньому проводиться серіями по три кадри [3].

MPEG-2 – являє собою подальше розширення MPEG-1. У ньому збільшено рекомендований розмір кадру тепер він становить 1920 x 1080 точок, додана

підтримка шестиканального звуку [3].

Ще досить поширений формат на сьогодні – це MPEG-2 з роздільною здатністю 720x576 пікселів та з швидкістю потоку 4000 – 8000 кбіт/с. Година відео займає 1,5 – 3 Гбайт [3].

MPEG-3 – був розроблений для HDTV додатків з високими параметрами в системах телебачення високої чіткості (high-defenition television, HDTV) – з максимальною роздільною здатністю (1920 * 1080 * 30) і швидкістю потоку 20 – 40 Mbps. Він не давав принципових поліпшень в порівнянні з MPEG-2, тому широкого поширення не набув і припинив використовуватись [3].

MPEG-4 враховує в собі узгоджений діапазон вимог висунутих представниками цифровий аудіовізуальної промисловості. З використанням нововведень, MPEG-4 пропонує краще ущільнення і інтерактивність з оточуючими медіа ресурсами через універсальний доступ до Інтернет або через бездротовий доступ. Стандарт забезпечує сумісність з іншими важливими стандартами, такими як H.263 і VRML [3, 10].

Його основне призначення (на думку розробників) - передача достатньо якісного відео в середовищах (мережах) з відносно малою пропускною здатністю. Основне нововведення в стандарті MPEG-4: на відміну від попередніх стандартів, які поділяли зображення при обробці на прямокутники, MPEG-4 оперує об'єктами довільної форми. Це дозволяє досягти більшого ступеня компресії при порівняно хорошій якості [3].

В MPEG-4 вдосконалений алгоритм ущільнення, якість і ефективність якого підвищено при всіх підтримуваних значеннях швидкості передачі даних. Ступінь ущільнення MPEG-4 така, що дозволяє записати повнометражний художній фільм на один компакт диск. Невеликий обсяг відео файлів дозволяє обмінюватися ними і по мережі. З цих причин він і став найбільш популярним в області «комп'ютерного» відео. Сьогодні майже всі фільми на компакт дисках пишуться в MPEG-4 [3].

Необхідність кодування окремих аудіо-візуальних об'єктів як природного походження, так і синтезованих, привело до створення ISO MPEG-4. Цей стандарт

включає в себе кілька частин, в яких розглядається, крім кодування відео також аудіокодування, кодування об'єктів і т.д. До відео відносяться частини 2 (ISO 14496-2 або MPEG-4 Part 2) і 10 (ISO 14496-10 або MPEG-4 Part 10) [3].

Розглянемо частину 10 стандарту MPEG-4. У неї є інші назви – стандарт H.264 і стандарт AVC (Advanced Video Coding). Частина 10 стандарту MPEG-4 визначає один з найсучасніших і технічно досконалих методів відео кодування. Стандарт AVC / H.264 був розроблений Joint Video Team (JVT), яка включає експертів з MPEG і VCEG (Video Coding Experts Group). Під цією назвою формат вже широко відомий. «Офіційне» найменування нового стандарту, Advanced Video Coding (AVC), було вибрано MPEG як відео додаток до аудіо формату Advanced Audio Coding (AAC) [3].

В стандарті AVC / H.264 визначені наступні профілі: базовий (baseline), основний (main) і розширений (extended). Пізніше розширений профіль був доповнений профілем для відео високої роздільної здатності (high profile), профілем high 10, профілем high 4: 2: 2 і профілем high 4: 4: 4 [3, 8-10].

Базовий профіль націлений на кодування і декодування в реальному часі для мобільних пристроїв. Він підтримує прогресивну розгортку, використовує I- і P-кадри, а також ентропійне кодування за методом CAVLC [3, 8-10].

Основний профіль призначений в основному для використання в широковіщанні. Він підтримує черезрядкову і прогресивну розгортку, використовує I-, P-, B-кадри, вагове передбачення (weighted predation), а також ентропійне кодування по методам CAVLC і CABAC [3, 10].

Розширений профіль використовується в засобах передачі, схильних до помилок – наприклад, в мобільних комунікаціях. Використовує I-, P-, B-, SP-, SI-кадри, підтримує як черезрядкову, так і прогресивну розгортку, дозволяє використовувати тільки метод CAVLC для ентропійного кодування [3, 8-10].

Профіль для відео високої роздільної здатності призначений для ефективного кодування HDV (high definition video). Він використовує адаптивний розмір блоку (8x8 або 4x4) і дозволяє застосовувати контекстно-залежні матриці квантування [3, 8-10].

Профіль high 10 є розширенням попереднього профілю для 10 біт на відлік компонентів зображення [3, 8-10].

Профіль high 4: 2: 2 підтримує формат YUV 4: 2: 2 і до 10 біт на відлік для компонентів кольору зображення [3, 8-10].

Профіль high 4: 4: 4 підтримує формат YUV 4: 4: 4 і до 12 біт на відлік компонентів кольору зображення. Крім цього він дозволяє використовувати режим кодування без втрат і пряме кодування RGB сигналу. Цей профіль призначений для кодування відео студійної якості [3, 8-10].

Кодування здійснюється по блоках. При цьому спочатку проводиться прогнозування відліків яскравості і кольорів компонентів в просторовій і тимчасовій областях. Пізніше різниця між прогнозованими значеннями і реальними піддається цілочисельному перетворенню і квантуванню. Після цього результат ущільнюється ентропійним кодером. Обробка кожного кадру ведеться в просторі YUV по блокам розміром 16x16 для компонентів luma (яскравість) і по 8x8 (для YUV 4: 2: 2) для компонентів chroma (кольору) [3, 8-10].

Цілочисельне перетворення здійснюється над блоками розміром 4x4. Це перетворення має схожі властивості з дискретним косинусним перетворенням, але відрізняється тим, що в ньому використовується цілочисельна арифметика. Це дозволяє домогтися підвищення швидкості роботи кодера і декодера. Отримані коефіцієнти за допомогою зігзаг- сканування шикуються в вектор, який квантується і піддається ентропійному ущільненню. Крім цього перетворення, в стандарті також закладена можливість використання wavelet-перетворень [3, 10].

Аналіз літературних джерел та Internet, показав, що для наших цілей по адоптації методу ущільнення найбільше підходить алгоритм стандарту H.264 (модифікація MPEG 4) [3, 10].

2 РОЗРОБКА МЕТОДУ УЩІЛЬНЕННЯ БАГАТОКАНАЛЬНОГО ВІДЕОЗОБРАЖЕННЯ ДЛЯ СИСТЕМ КОМП'ЮТЕРНОГО МОНІТОРИНГУ

2.1 Адаптація методів

2.1.1 Адаптація стандарту H.264 з метою визначення його здатності до багатоканального ущільнення

Стандарт H.264 визначає синтаксис кодованого бітового потоку і метод декодування бітового потоку. На даний момент часу цей стандарт є одним з найефективніших в сфері ущільнення відеозображення.

Про ефективність кодувальника можна судити по реалізації наступних функцій:

— внутріпрогнозірованіе з використанням прямого та зворотного дискретного косинусного перетворення (DCT), а також пряме і зворотне квантування;

— деблокуючого фільтрація;

— оцінка руху з використанням межкадрового порівняння з точність від пікселя до пікселя.

Кожна з цих функцій вимагає великої обробки і повинна виконуватися у реальному часі, щоб бути корисним.

На рисунку 2.1 представлений кодувальник H.264, а на рисунку 2.2 показаний його відповідний декодер. На малюнках терміни DCT, MV, ME, MC і Q – 1 представляють дискретне косинусне перетворення, вектор руху, оцінку руху, компенсація рух і зворотне квантування відповідно.

Як показано на рисунку 2.1, кодування кадру складається з прямого шляху і шляху відновлення. Метою процесу відновлення є отримання системи відліку для руху компенсації макроблоків наступного кадру. Багато з функцій відновлення також використовуються при декодуванні процес.

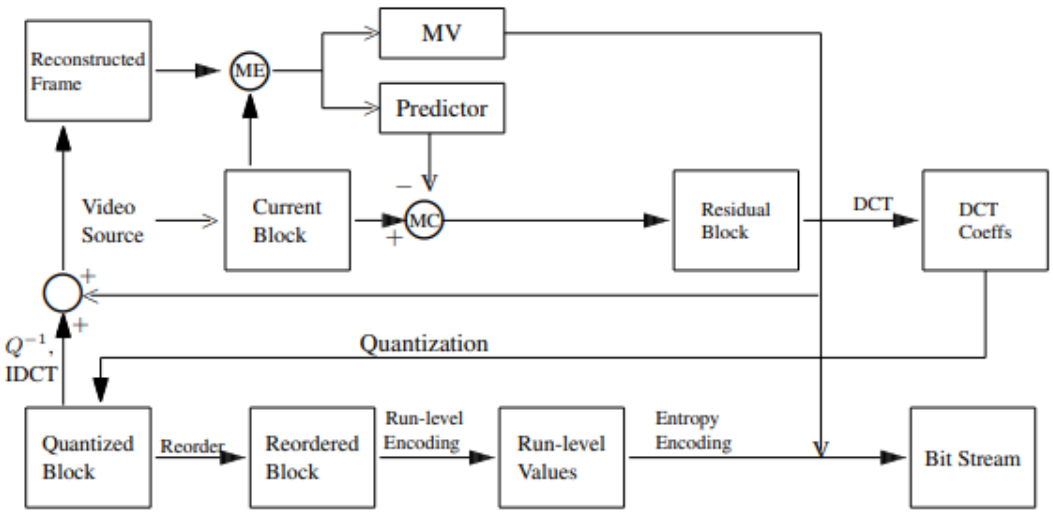

Рисунок 2.1 — Архітектура кодувальника H.264

Кодування кадрів виконуються шляхом прямого кодування та шляхом відновлення.

Алгоритм прямого кодування:

— зчитаємо 24-бітний кадр зображення RGB з нестисненого файлу avi;

— розкладуємо кадр RGB на макроблоки 16 × 16;

— зробимо перетворення і знижуйте дискретизацию кожного RGB 16 × 16 макроблоку до шести блоків 8 × 8 YCbCr(сімейство кольорового простору, який використовується у компонентному відео) у форматі YCbCr 4: 2: 0;

— знайдемо в довідковій системі "найбільш відповідний" макроблок YCbCr, з якого ми зробимо посиланням на макроблок;

— зробимо кодування вектора руху (MV) в якості посилання на макроблок за допомогою попередньо розрахованого коду Хаффмана і передамо кодові слова;

— обчислимо залишки між поточним макроблоком і опорним макроблоком;

— застосуємо дискретне косинусное перетворення (ВСЕ) до решти;

— виконаємо пряме квантування блоку DCT; реконструювати блок, як описано в шляху відновлення;

— змінемо порядок кожного квантованного блоку DCT 8 × 8 зигзаподібним методом;

— на рівні прогону кодируем кожен квантований переупорядкувати ний блок

DCT для отримання тривимірних (прогін, рівень, останній) кортежів;

— використовуємо попередньо розраховані кодами Хаффмана слова разом зі знаковими бітами для кодування тривимірних кортежів;

— передамо кодові слова.

Алгоритм шляхом відновлення:

— виконаємо зворотне квантування кожного квантованного блоку DCT , хоча це процес з втратами і, отже, відновлений блок не ідентичний блоку до квантування;

— застосуємо зворотний DCT (IDCT) до отриманого DCT блоки для отримання залишкових вибірок блоків 8 × 8 YCbCr;

— додамо блоки залишкових проб до відповідних блокам еталонних проб, які раніше були закодовані і реконструюємо для отримання опорного макроблоку;

— збережемо відновлений опорний макроблок у векторі, який буде використаний для побудови одного кадру;

— перетворює кожен макроблок YCbCr, збережений в векторі до макроблоку RGB, який потім зберігається в буфері. У буфері буде відновлений кадр.

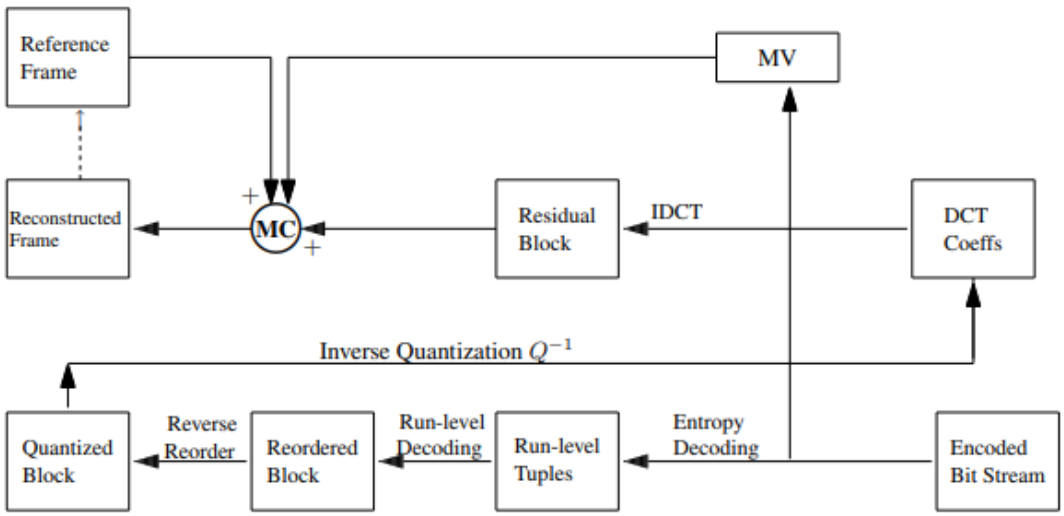

Рисунок 2.2 — Архітектура декодувальника H.264

Алгоритм декодера складається з наступних кроків, деякі з яких перетинаються з шляхом відновлення на етапі кодування:

— побудуємо дерево Хаффмана з попередньо обчислених кодових слів

Хаффмана для декодування тривимірних кортежів на рівні прогону;

— побудуємо дерево Хаффмана з попередньо обчислених кодових слів Хаффмана для декодування векторів рухів (MV);

— зчитаємо біти бітового потоку з закодованого файлу і перегляньте дерево MV Хаффмана, щоб відновити MV TSS макроблоку;

— зчітаємо біти з закодованого файлу і переглянемо тривимірне дерево Хаффмана на рівні виконання, щоб відновити тривимірні кортежі на рівні виконання, і отримаємо блоки DCT 8 × 8;

— зробимо зворотне переупорядочение і зворотне квантування кожного блоку DCT;

— застосуємо зворотний DCT (IDCT) до блокам DCT, отриманим на попередньому етапі, щоб отримати 8 × 8 блоків залишкових вибірок YCbCr;

— додамо блоки залишкових вибірок до відповідних блокам еталонних вибірок, які були раніше закодовані і реконструйовані;

— збережемо відновлений опорний макроблок в векторі, який буде використовуватися для побудови одного кадру;

— перетворимо кожен макроблок YCbCr, збережений в векторі, в макроблок RGB, який потім зберігається в буфері. У буфері буде відновлений кадр.

В результаті процес кодування H.264 вимагає великих обчислювальних ресурсів, ніж існуючі стандарти. Отже, ми зацікавлені в поліпшенні швидкості кодувальника, тому вирішено позбутись декількох методів, залишив основний методу кодування та прискорення через впровадження багатопотоковості.

2.1.2 Застосування методу Retinex для відеозображення

Retinex – це теорія кольору, розроблена Е. Х. Лендом і Дж. Дж. Маккейн. Застосування теорії Retinex дозволяє моделювати те, як світло і колір уловлюються людським мозком. Дві теорії, а саме «сталість кольору» і «сталість яскравості» (в яких передбачається, що колір і яскравість сприймаються людським оком без урахування освітлення), складають основу теорії сітківки ока. Більш того, в теорії Retinex зображення можуть бути виражені як «множення світла, що висвітлює

об'єкт, і відбивної здатності цього об'єкта». У нашому випадку ми використовуємо мотод Retinex для поліпшення ступеня стиснення, але яке не буде помітним для зору людини.

Теорія, що лежить в основі Retinex, проілюстрована на прикладі зображення Мондріана (рисунок 2.3) зі штучним плавним штрихуванням, може бути описана наступним чином. Коефіцієнт відображення двох обраних ділянок на зображенні Мондріана можна визначити, простеживши шлях між ними. По дорозі множимо виміряні коефіцієнти інтенсивності. На кордоні плям відносини інтенсивностей відповідатимуть змінам відбивної здатності, а всередині ділянок відносини інтенсивностей будуть близькі до одиниці. Ленд і Макканн [11] пропонують встановити граничне значення для цих співвідношень, щоб придушити зміни інтенсивності через затінення, яке передбачається плавним. Остаточний коефіцієнт буде дорівнює фактичному коефіцієнту відображення між двома плямами.

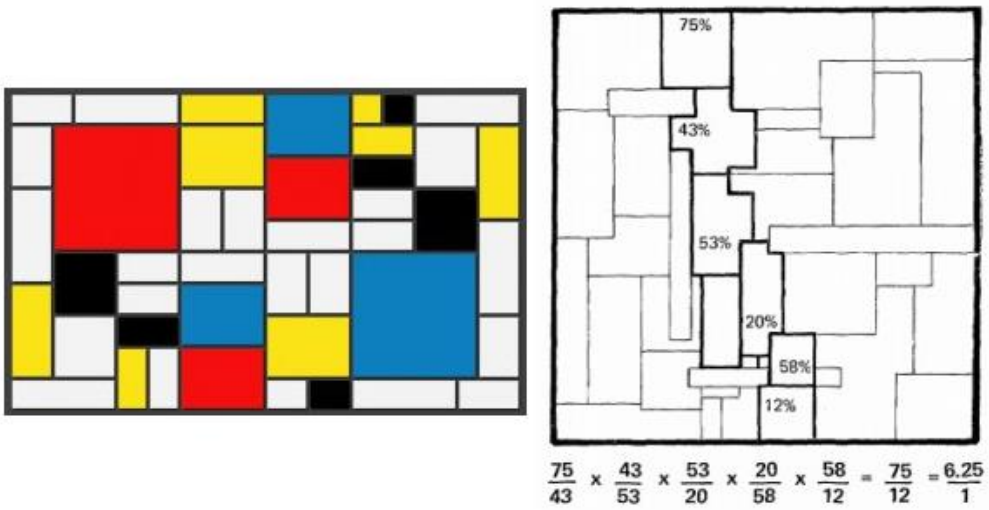

Рисунок 2.3 — Ілюстрація теорії Retinex

На рисунку 2.3 справа розташовано розрахунок відбивної здатності на основі шляху, запропонований Лендом і Маккал, а зліва – приклад типового стилю картин Мондріана.

Формалізація описаного алгоритму, а також його розширення на кольорові зображення належить Хорну [12] і Блейку [14], проте ми будемо використовувати Морелову [15]. Коефіцієнт відображення для кожного пікселя в каналі I_c

обчислюється шляхом розгляду сімейства $\{x_k\}_{Kjk=1} \subseteq H--\times W$ для J шляху. Для пікселя $x_n \in H--\times W$ всі шляхи починаються з x_n і закінчуються довільним пикселем в зображенні. Тоді коефіцієнт відображення $R_c(x_n)$ для пікселя x_n в каналі с обчислюється як:

$$Rc(xn) = \frac{1}{J}\sum_{j=1}^{J}\sum_{k=1}^{K_j}\lambda\big(I_c(x_k) - I_c(x_k+1)\big), \tag{2.1}$$

де

$$\lambda(x) = \begin{cases} |x| & if\ |x| > \tau \\ 0 & else \end{cases} \tag{2.2}$$

Основною проблемою цього підходу є проблема вибору шляхів {xk} Kjk = 1, так що рівняння 2.1 може бути ефективно обчислено і розраховує належну відбивну здатність (тобто його хороші наближення) [15].

Формалізація диференціального рівняння в приватних похідних. Морел пропонує формалізацію теорії Retinex рівняннями в приватних похідних [15].

Розглядаємо сімейство з J випадкових шляхів {xk} Kjk = 1⊆H × W, кожен з яких починається в пікселі xn і закінчується в випадково обраному пікселі. Випадкові шляху визначені на площині Z2, проте відображаються на кордонах зображення, що призводить до чітко визначених шляхах в площині зображення. Тоді яскравість пікселя xn в каналі Ic по аналогії з рівнянням 2.1 може бути записана як

$$Rc\ (xn) = 1N\Sigma Jj\Sigma Kjk = 1\lambda\ (Ic\ (xk)\ -Ic\ (xk + 1)) \tag{2.3}$$

На відміну від рівняння 2.1, це можна інтерпретувати як очікування відносної відбивної здатності, яка визначається одним таким випадковим шляхом.

Використовуючи структуру випадкового шляху Морела, що показує коефіцієнт відображення Rc, вирішемо рівняння Пуассона:

$$\{-\Delta Rc\ (xn) = Fc\ (xn)\ \partial Rc\partial n = 0xn\in H \times W - x\in\partial\ (H \times W) \qquad (2.4)$$

з граничною умовою Неймана, де ∂Rc∂n позначає нормальну похідну, а ∂ (H × W) позначає кордон H × W. Тут Fc (x) визначається як

$$Fc\ (xn) = \lambda\ (Ic\ (xn) - Ic\ (xn, 1 + 1, xn, 2)) + \lambda\ (Ic\ (xn) - Ic\ (xn, 1\text{-}1, xn, 2))$$
$$+ \Lambda\ (Ic\ (xn) - Ic\ (xn, 1, xn, 2 + 1)) + \lambda\ (Ic\ (xn) - Ic\ (xn, 1, xn, 2\text{-}1)).$$

Рівняння Пуассона, за формулою 2.4, може бути ефективно вирішено за допомогою дискретного перетворення Фур'є [15, 16]. Далі ми коротко виводимо необхідні рівняння. Дискретне двовимірне перетворення Фур'є функції Fc має вигляд

$$F\ \hat{}\ r, s = 1N\Sigma Hu = 1\Sigma Wv = 1Fu, = vexp\ (2\pi iruH)\ exp\ (2\pi svW) \qquad (2.5),$$

в той час як зворотне, дискретне, двовимірне перетворення Фур'є каналу зображення Ic може бути записано як

$$Iu, v = 1N\Sigma Hr = 1\Sigma Ws = 1I\ \hat{}\ r, sexp\ (-2\pi iurH)\ exp\ (-2\pi vsW) \qquad (2.6).$$

Перетворення обох частин рівняння 2.4 з допомогою дискретного двовимірного перетворення Фур'є дає наступне співвідношення:

$$I\ \hat{}\ r, s = F\ \hat{}\ r, s2\ (cos\ (2\pi rH) + cos\ (2\pi sW) - 2).$$

Застосування зворотного дискретного двовимірного перетворення Фур'є для відновлення Ic вирішує рівняння Пуассона, за формулою 2.4, в разі періодичних граничних умов. Однак, заміна рівняння 2.5 і рівняння 2.6 на дискретне двовимірне косинусное перетворення, служить обмеженням граничних умов Неймана, як в

рівнянні в формулі 2.4.

Узагальнення на відео. Алгоритм Retinex, реалізований Морелем легко узагальнюється на відео. Рівняння Пуассона, сформульоване для зображення I в рівнянні 2.4, в рівній мірі справедливо для відео V, і рівняння 2.5 і рівняннся 2.6 можуть бути легко узагальнені для трьох вимірів [16]. Тільки праву частину Fc (xn), Xn∈H × W× T – рівняння 2.4 необхідно адаптувати в такий спосіб:

$$F_c(x_n)=\lambda(I_c(x_n)-I_c(x_{n,1}+1,x_{n,2},x_{n,3}))+\lambda(I_c(x_n)-I_c(x_{n,1}-1,x_{n,2},x_{n,3}))Fc(xn)=\lambda(I$$
$$c(xn)-Ic(xn,1+1,xn,2,xn,3))+\lambda(Ic(xn)-Ic(xn,1-1,xn,2,xn,3))+\lambda(I_c(x_n)-I_c(x_{n,1},x_{n,2}+1,$$
$$x_{n,3}))+\lambda(I_c(x_n)-I_c(x_{n,1},x_{n,2}-1,x_{n,3}))+\lambda(Ic(xn)-Ic(xn,1,xn,2+1,xn,3))+\lambda(Ic(xn)-Ic(xn,1$$
$$,xn,2-1,xn,3))+\lambda(I_c(x_n)-I_c(x_{n,1},x_{n,2},x_{n,3}+1))+\lambda(I_c(x_n)-I_c(x_{n,1},x_{n,2},x_{n,3}-1))+\lambda(Ic(xn)$$
$$-Ic(xn,1,xn,2,xn,3+1))+\lambda(Ic(xn)-Ic(xn,1,xn,2,xn,3-1)).$$

2.1.3 Впровадження макроблоків в новому методу

H.264 має кілька профілів. Профіль - це набір функцій, які можуть бути реалізовані (або ні) в додатку відеокодера. Це означає, що не всі функції повинні підтримуватися в кожній реалізації відеокодера або декодера.

Новий метод слідує основному алгоритму та виконує більшість дій, але не всі фунції стандарту ущільнення H.264, як майже всі відеокодери на основі цього стандарту. Фактично, відмінності між декількома кодировщиками H.264 зазвичай пов'язані з кількістю реалізованих функцій. Отже, основна ідея при кодуванні видеокодека H.264 - уникати кодування максимальної кількості кадрів. Для цього існують три різні типи кадрів: I (від англ. Intra pictures, укр. усередині картинки), P (від англ. Predicted pictures, укр. прогнозовані картинки) і B (від англ. Bi-predictive pictures або Bi-directional pictures, укр. Двонаправлені картинки) (Рисунок 2.4) [17].

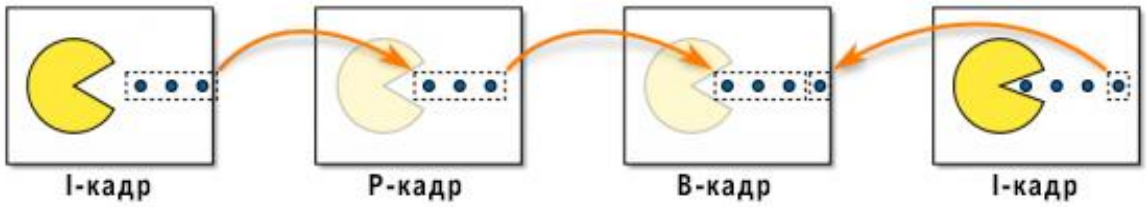

Рисунок 2.4 — Приклад кадів I, P і B

I-кадр - це зображення з внутрішнім кодуванням, фактично повністю певне зображення, подібне звичайному файлу статичного зображення. P-кадри і B-кадри містять тільки частину інформації про зображення, тому для їх зберігання потрібно менше місця, ніж для I-кадру, і, таким чином, поліпшується ступінь стиснення відео.

Кадр P (прогнозоване зображення) містить тільки зміни зображення в порівнянні з попереднім кадром. Наприклад, в сцені, де автомобіль рухається по нерухомому фону, необхідно кодувати тільки його руху. Кодеру не потрібно зберігати незмінні пікселі фону в кадрі P, що дозволяє економити місце. P-кадри також відомі як дельта-кадри.

Кадр B (зображення з двонаправленим прогнозуванням) економить ще більше місця за рахунок використання відмінностей між поточним кадром і попереднім і наступним кадрами для визначення його вмісту.

Таким чином, додаток працює на рівні макроблоку. Це означає, що кожен кадр розбивається на кілька підкадрів, що дозволяє додатку вибирати, який тип кадру використовувати для кожного сегмента кадру. Ця функція збільшує ступінь стиснення кінцевого відеопотоку, а також ускладнює код, але, звичайно, варто витрачених зусиль [17].

Після аналізу макроблоку додаток визначає його тип (I, P або B). Залежно від цього рішення процес кодування буде відрізнятися. У разі, якщо макроблок відноситься до типу P або B, то процес кодування не може розпочатися до тих пір, поки не буде закодований опорний макроблок або макроблоки. Схема алгоритму приведена на рис. 2.5.

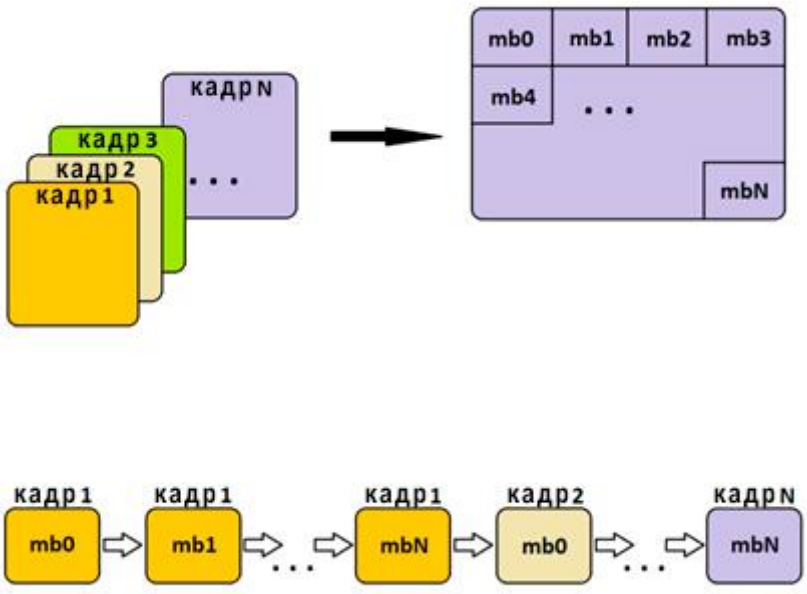

Рисунок 2.5 — Схема алгоритму

2.1.4 Адаптація моделі програмування OmpSs

Модель програмування OmpSs – це спроба інтегрувати функції моделі програмування StarSs, розробленої BSC, в єдину модель програмування. Зокрема, мета полягає в тому, щоб розширити OpenMP новими директивами для підтримки паралелізму і неоднорідності асинхронних потоків даних (як в пристроях, подібних GPU) [18]. Найбільш характерною особливістю моделі програмування OmpSs є розширення завдань OpenMP залежностями. Залежно дозволяють користувачеві коментувати дані потоку програми, таким чином, під час виконання цю інформацію можна використовувати, щоб визначити, чи може паралельне виконання двох завдань викликати гонку даних. Асинхронний паралелізм включається в OmpSs за рахунок використання залежностей даних між різними завданнями програми. Конструкція завдання OpenMP розширена з цією метою пропозиціями in (позначають введення), out (позначають висновок) і inout (позначають введення / виведення). Вони дозволяють вказати для кожного завдання в програмі, яких даних вона очікує, і сигналізувати про її готовності. Зверніть увагу, що питання про те, чи дійсно завдання використовує ці дані певним чином, - це відповідальність програміста.

У нашому випадку обрана модель програмування – OmpSs. Основна причина вибору цієї моделі програмування замість іншого полягає в тому, що ми хочемо довести, що OmpSs можуть використовуватися також для робочих навантажень з невисокою продуктивністю. Модель програмування OmpSs краще за всіх поєднує в собі всі характеристики, які повинна мати модель паралельного програмування.

Що стосується неоднорідності, OmpSs забезпечує підтримку виконання частин програми на графічному процесорі, просто додаючи в код директиву OpenMP. Отже, можна бачити, що одна з найважливіших особливостей – це простота. Це перетворюється в чіткий код, який потім легко підтримувати. Це дозволяє розробнику забути про складні функції і думати тільки про алгоритмі, який він хоче реалізувати, і ні про що інше [18].

OmpSs створює пул завдань. Ці завдання будуть прив'язані до будь-якого доступного потоку. Щоб дізнатися, який потік доступний, також створюється пул потоків. Це означає, що всі потоки створюються при ініціалізації програми, будучи одним з тих потоків, який є головним, який відповідає за виконання послідовного коду користувача. Решта потоки називаються робочими потоками. Ці робочі потоки можуть стати головними, якщо вони створюють більше завдань, це означає, що вкладення також можливо в рамках моделі програмування OmpSs.

Кожна паралельна область коду визначається як завдання в моделі програмування OmpSs. Кожне завдання має свої власні залежності, які оголошуються за допомогою директив in, out і inout. Ці директиви встановлюються користувачем, і їх використання служить тільки для допомоги середовищі виконання при побудові залежності графа, щоб мати можливість виконувати кожне завдання, коли дані доступні. Використання кожної директиви самоочевидне, директива in повідомляє середовищі виконання, що завдання має прочитати дані, зазначені в пропозиції in, директива out повідомляє середовищі виконання, що завдання змінить дані, зазначені в пропозиції out, нарешті, директива inout повідомляє середовищі виконання, що завдання необхідно буде прочитати і записати дані, зазначені в директиві inout. Є й інші директиви, такі як concurrent, які говорять середовищі виконання, що забезпечення правильного доступу до

даних буде відповідати програмісту [18].

На рис 2.6 показано, як працює модель програмування OmpSs. Ми бачимо, що з вихідного коду, компілюючи його за допомогою Mercurium, який може використовувати кілька нативних компіляторів для послідовного виконання, буде отриманий виконуваний файл. Mercurium також використовує бібліотеки часу виконання Nanos ++ під час компонування. Нарешті, виконуваний файл буде використовувати динамічні бібліотеки з Nanos ++ для паралельного виконання.

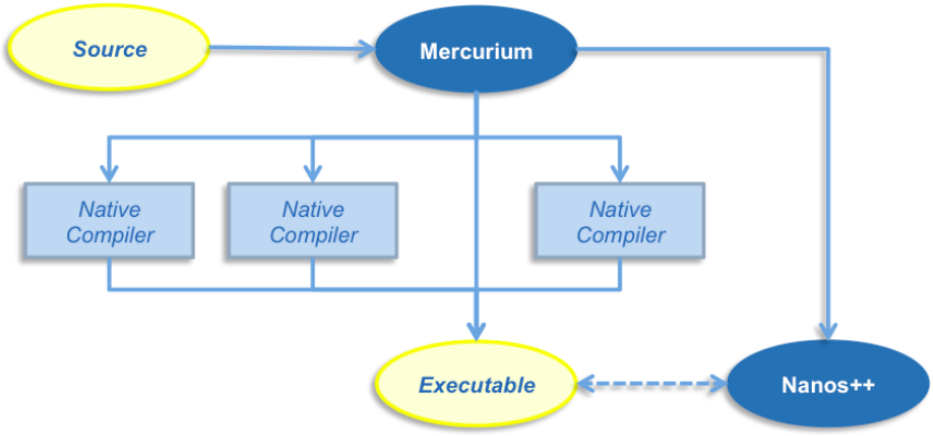

Рисунок 2.6 — Приклад реалізації моделі програмування OmpSs

2.2 Розробка структури вдосконаленого методу

Новий метод кодування створює віртуальний конвеєр кадрів для кодування. Ідея полягає в тому, щоб мати можливість кодувати одночасно найбільша кількість кадрів, але це не так просто через різних типів кадрів, точніше P і B кадрів. Такі кадри використовують пікселі, які вже закодовані в іншому кадрі або кадрах, щоб уникнути їх повторного кодування. Це прямо переводиться в залежності.

На рис. 2.7 показаний приклад віртуального конвеєра, створюваного додатком. Там також відзначені залежності між кадрами. Насправді, для кадру P не потрібно чекати, поки буде закодований весь попередній кадр. Одна з ідей полягає в тому, щоб дочекатися, поки окремі частини кадру вже обчислені, а потім почати процес кодування кадру. Потенційно нам не доведеться чекати знову, але це не буде звичайним випадком, тому при досягненні деякої точки процесу кодування кадру

буде обов'язково перевіряти, обчислена чи вже наступна частина попереднього кадру по порядку. продовжити. З B-кадрами це буде найгірше, тому що нам потрібно чекати ще й наступного кадру.

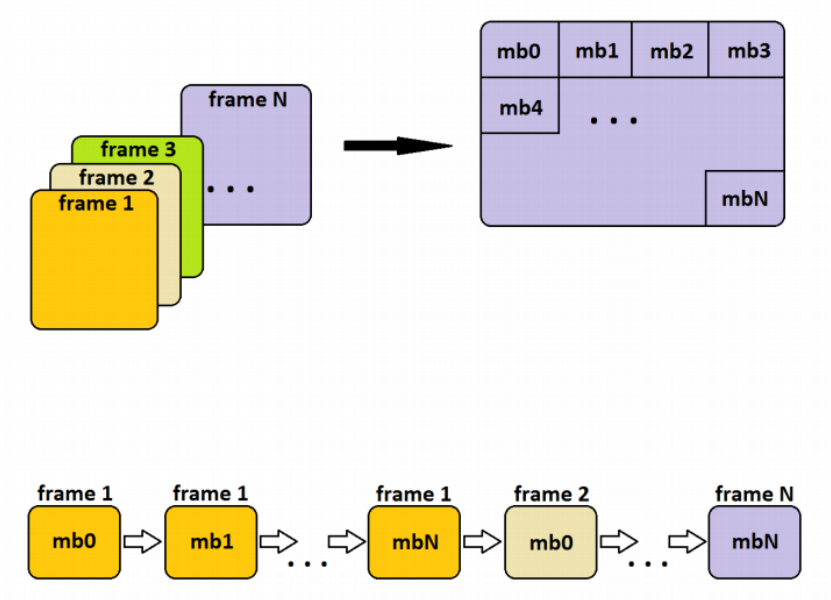

Рисунок 2.7 — Конвеєр алгоритму

Це основна залежність додатка. При кодуванні відео з низьким дозволом очікується відсутність масштабованості, але тільки при великій кількості кадрів P або B. Якщо немає (тобто всі кадри відносяться до типу I), то очікується, що масштабованість буде більш-менш лінійної, але в цьому випадку метрика кадрів в секунду буде гірше, так як буде обов'язково кодувати всі пікселі, робота, яка не виконується при кодуванні P або B кадр, в цих випадках дійсно кодуються тільки деякі пікселі. Звичайно, ідеальна масштабованість неможлива тільки тому, що ми не виконуємо паралельно все додаток, а тільки його частину.

У нас є ще одна залежність, не настільки критична. Як можна здогадатися, вихідний відеофайл необхідно прочитати перед тим, як почати його кодування. Фактично, процес читання розбивається на безліч частин, оскільки кількість кадрів має відеофайл, це означає, що ми будемо читати відеофайл кадр за кадром.

Це означає, що ми не можемо почати кодування кадру до його читання. Ця залежність не є великою проблемою тільки тому, що час, який додаток витрачає на читання кадру, мізерно мало в порівнянні з часом кодування кожного кадру. На

рис. 2.8 показаний той же конвеєр, що і раніше, але доданий процес читання для кожного кадру.

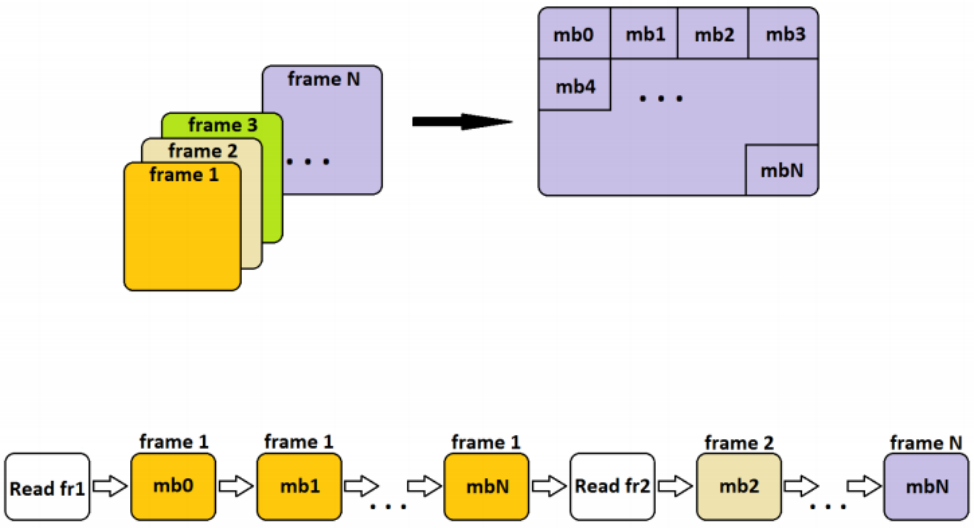

Рисунок 2.8 — Конвеєр алгоритму

Ми маємо справу з двома залежностями, які нам потрібно зруйнувати за допомогою нашого алгоритму, а саме:

— P-кадр залежить від I або P-кадру. Один P-кадр не може бути закодований до того, як попередній I або P-кадр не буде закодований;

— B-кадр залежить від попереднього I або P-кадру і наступного I-кадру. Один B-кадр не може бути закодований перед попереднім I або P-кадром і наступного I-кадр не кодується;

— рамка залежить від прочитання кадру. Один кадр можна закодувати, поки він не був прочитаний з вхідного файлу;

Останнє насправді досить очевидне, ми залежимо від вхідного файлу, який потрібно прочитати, перш ніж почати обробку (тобто кодування) потоку даних. Що стосується першого і другого виду залежності, ця міжкадрового залежність, ймовірно, однакова, нам потрібно дочекатися обробки деяких даних, перш ніж починати обробку тих, які у нас є.

Ці залежності необхідно розірвати, тому що це єдиний спосіб досягти бажаної продуктивності, оскільки тільки метод OmpS може буди застосований для робочих навантажень високої продуктивності. Фрейм залежить від читання самого

фрейму. Насправді цю залежність не так вже й складно зламати. Основна причина в тому, що між читанням фрейму і його кодуванням потрібно виконати багато роботи, наприклад, підготовка структур даних. Беручи це до уваги, ідея полягає в тому, щоб прочитати кадр і продовжити виконання програми до тих пір, поки ми не дійдемо до початку процесу кодування в той же час. Рисунок 2.9 показує цю ідею.

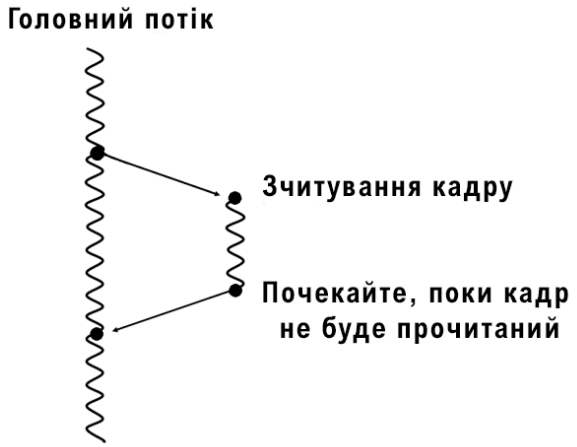

Рисунок 2.9 — Схема того, як повинно виконуватись зчитування

Р-кадр залежить від І або Р-кадру. Наступне відношення залежності стосується двох кадрів. Припустимо, що обидва фрейма вже прочитані і проаналізовані. Насправді ми працюємо не з кадрами, а з макроблоків, які є частиною кадру, але ми збираємося представити, що кожен кадр складається з макроблоків одного типу.

На цьому етапі у нас є І або Р-кадр і інший Р-кадр, який залежить від першого. Тепер ми збираємося спроектувати, що станеться, якщо є І-кадр і Р-кадр, щоб уникнути складності.

Обидва кадру можна кодувати одночасно, тому нам потрібно спочатку кодувати І-кадр, який першим увійшов у віртуальний конвеєр. Ідея полягає в тому, щоб уникнути очікування всього процесу кодування І-кадру, щоб уникнути потоків без будь-якої роботи. Отже, ми подумаємо, як працює ця залежність.

Дивлячись на рис. 2.4, ми бачимо, що не всі пікселі Р-кадру залежать від усіх пікселів І-кадру, насправді це залежить тільки від деяких з них. Тому варто

починати процес кодування кадру P якраз тоді, коли деяка точка кадру I вже закодована. Ідея полягає в тому, щоб розділити кадр на декілька частин, щоб виконати цю синхронізацію в різних кадрах кілька разів, щоб уникнути попадання в пікселі кадру, які мають залежності.

Можна подумати про те, щоб відзначити, які пікселі залежать від інших. Проблема з цим дизайном полягає в тому, що ці залежності пов'язані з переміщенням пікселів, а не з їх зміною, тому потрібно буде зберігати перетворення і зміщення для кожного залежного пікселя, який потенційно може становити більше трьох кварт від загальної кількості пікселів в кадрі. . Ще одна проблема - необхідність перевірки кожного пікселя, що може вплинути на кінцеву продуктивність.

Отже, ідея полягає в тому, щоб розробити свого роду алгоритм водоспаду, тобто: якщо у нас є послідовність кадрів, починаючи з одного кадру I, за яким слід кінцеве число кадрів P, то можна кодувати, наприклад, 10% I-кадру, поки не почнеться кодування першого P-кадру, який у нас буде в віртуальному конвеєрі.

Другий кадр, який є першим кадром P, в кінцевому підсумку закодує 10% свого зображення. На цьому етапі цей процес кодування буде чекати, поки I-кадр не досягне 20% від процесу кодування, але третій кадр, поставлений в чергу (тобто другий P-кадр, який залежить від першого P-кадру), почне процес кодування .

Після повторного кодування у нас буде 3 кадри з 30, 20 і 10% від всього процесу кодування. Знову ж таки, четвертий кадр може початися, тому що залежить від того, який в даний час займає 10% процесу кодування.

Така поведінка може тривати вічно, але можна бачити, що якщо ми використовуємо 10% зображення в якості значення синхронізації, тоді ми зможемо мати тільки десять кадрів, кодованих одночасно. Якщо ми подумаємо про паралелізм, ідея полягає в тому, щоб встановити цей відсоток з урахуванням того, скільки потоків ми використовуємо. Таким способом не варто встановлювати 10%, наприклад, якщо у нас всього 2 потоку, найкраще буде 50%, щоб уникнути максимально можливої синхронізації. На рис. 2.10 показано, як буде виконуватися віртуальний конвеєр при використанні поясненого алгоритму.

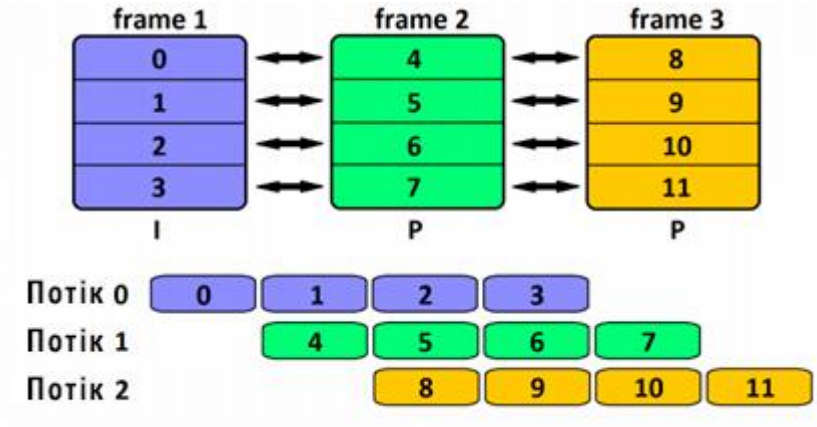

Рисунок 2.10 — Приклад алгоритму паралелізма на основі стандарту H.264

B-кадр залежить від попереднього I або P-кадру і наступного I-кадру. Ця залежність дуже схожа на попередню, але кадр залежить не тільки від попереднього кадру, але і від наступного.

Таким чином, ідея полягає в тому, щоб почати кодування попереднього I або P-кадру і наступного I-кадру одночасно. Кадр B не почне процес кодування до тих пір, поки не буде досягнутий відсоток від двох інших кадрів. Як і раніше, у нас будуть свого роду контрольні точки, де B-кадр повинен чекати, поки два інших кадру не отримають цей відсоток закодованих пікселів.

Єдина зміна - це порядок, в якому насправді кодуються кадри. Це означає невелику модифікацію віртуального конвеєра. Тепер кадри, які повинні бути закодовані до B-кадру, повинні бути спочатку закодовані, тому нам потрібно перемістити їх у віртуальний конвеєр. На рис. 2.11 показано, як модифікується реальний конвеєр.

Після цієї модифікації конвеєра залежність перетворюється в щось дуже схоже на залежність між P-кадром і його попередником. Єдина відмінність між цим кадром P і кадром B полягає в тому, що при перевірці можливості продовження кодування обов'язково перевіряти обидва кадру, від яких залежить цей кадр, і не починати процес кодування, поки обидві залежності не будуть задоволені.

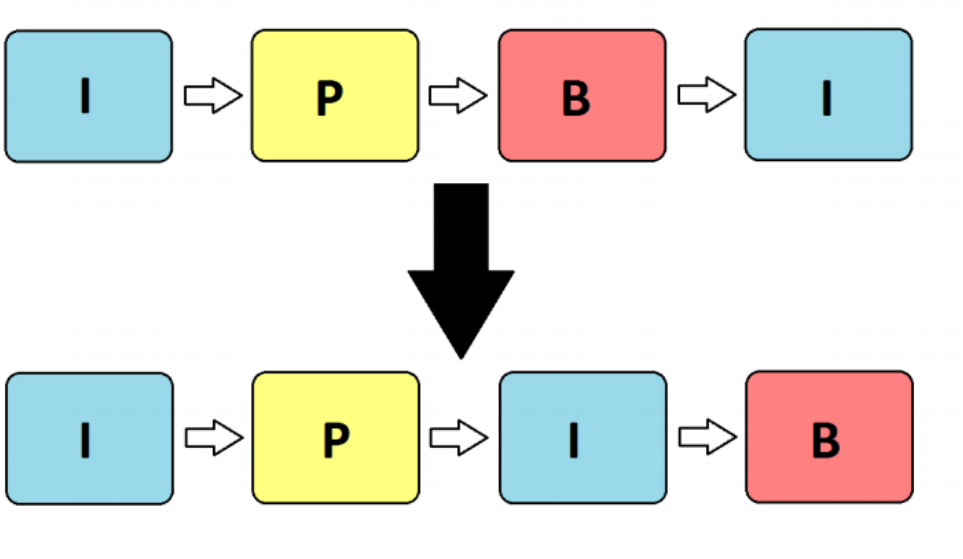

Рисунок 2.11 — Віртуальний конвеєр, отриманий після переупорядковування кадрів при кодуванні B-кадру

3 РОЗРОБКА ТА ТЕСТУВАННЯ АЛГОРИТМУ Й ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОПРАЦЮВАНЯ БАГАТОКАНАЛЬНИХ ВІДЕОСИГНАЛІВ

3.1 Програмна реалізація

У цьому розділі ми збираємося пояснити, як був реалізований вже описаний дизайн. Оскільки вже реалізована паралельна версія, яка дбає про створення декількох структур даних для належної ізоляції кожного процесу кодування.

Змінений вихідний код буде надано для кожної модифікації. Звичайно, тут буде показаний не весь код програми, тому будуть представлені тільки порушені проблеми. Весь представлений код знаходиться у додатку Б.

3.1.1 Зчитування кадрів

Читання кадру необхідно виконати до початку процесу кодування кадру. Щоб прочитати фрейм, ми створюємо нову задачу кожен раз, коли фрейм потрібно прочитати, це один раз за фрейм.

Обробка кожного кадру, включаючи його читання, включена в наступний код (цикл кодування):

```
for ( i_frame = 0 , i_file = 0; b_ctrl_c == 0 && ( i_frame < i_frame_total ||
i_frame_total == 0) ; )
{
        int res = 0;
# ifndef HAVE_OMPSS
        res = p_read_frame ( &pic , opt ->hin , i_frame + opt -> i_seek ) ;
# else
        # pragma omp task out ( pic ) label ( p_read_frame )
        res = p_read_frame ( &pic , opt ->hin , i_frame + opt -> i_seek ) ;
        # pragma omp taskwait on( pic )
# endif
```

```
if(res )
        break ; // кадр не читається належним чином, вихід
pic . i_pts = ( int64_t ) i_frame * param -> i_fps_den ;
if( opt -> qpfile )
        parse_qpfile ( opt , &pic , i_frame + opt -> i_seek );
else
{
        /* Не змінюйте параметри */
        pic . i_type = H264_TYPE_AUTO ;
        pic . i_qpplus1 = 0;
}
i_file += Encode_frame ( h, opt ->hout , &pic );
i_frame ++;
/* оновленя рядку стану (до 1000 разів на вхідний файл) */
if( opt -> b_progress && i_frame % i_update_interval == 0 )
{
        int64_t i_elapsed = h264_mdate () - i_start ;
        double fps = i_elapsed > 0 ? i_frame * 1000000. / i_elapsed : 0;
        double bitrate = ( double ) i_file * 8 * param -> i_fps_num / (
                ( double ) param - > i_fps_den * i_frame * 1000 );
        if( i_frame_total )
        {
                int eta = i_elapsed * ( i_frame_total - i_frame ) / ((
                        int64_t ) i_frame * 1000000) ;
                sprintf ( buf , " h264 [%.1 f %%] %d/%d frames , %.2 f fps ,
                        %.2 f kb/s, eta %d :%02 d :%02 d",
                                100. * i_frame / i_frame_total , i_frame ,
                                        i_frame_total , fps , bitrate ,
                                eta /3600 , ( eta /60) %60 , eta %60 );
        }
```

```
            else
            {
            sprintf ( buf , " h264 %d frames : %.2 f fps , %.2 f kb/s", i_frame ,
            fps , bitrate );
}
            fprintf ( stderr , "%s \r", buf +5 );
            SetConsoleTitle ( buf );
            fflush ( stderr );
        }
}
```

Ми бачимо, що для кожного кадру відеофайлу додаток зчитує кадр (рядок 8), виконує кілька перевірок і налаштовує кадр для його кодування і, нарешті, викликає функцію Encode frame в рядку 22, яка фактично виконує кодування кадру. Після цього в рядку 25 знаходиться умовний оператор, який виводить інформацію про процес кодування кадру, якщо користувач встановив детальний прапор під час запуску програми. Ініціалізація структури даних, яка представляє фрейм (тобто змінну pic), не відображається, тому що вона поза циклом.

Всі зміни, які були зроблені в циклі, містяться в рядках з 4 по 10. Як видно, ми використовували умовну групу, щоб перевірити, чи визначено HAVE_OMPSS чи ні. Цей макрос встановлюється під час компіляції на той випадок, якщо ми хочемо скомпілювати OmpS-версію додатка. Якщо він визначений, буде виконуватися код рядків з 7 по 9, а в іншому випадку - рядок 5. Рядок 5 була вихідним кодом.

У рядку 7 завдання оголошується з використанням синтаксису OmpSs. Ми оголошуємо один вихід і встановлюємо мітку завдання, щоб поліпшити візуалізацію трасування в найближчому майбутньому. Після оголошення завдання чекаємо, поки ця задача не закінчиться.

На даний момент це не недолік продуктивності, тому що час, який ми витрачаємо на читання з вхідного файлу, незначний. Про те, чому ми створюємо завдання, коли ми чекаємо завершення, просто тому, що нам потрібно забезпечити

деякі умови в задачі, а створення її в цей момент згодом знизить складність. Крім того, в задачах може бути створено більше потоків, так що врешті-решт це не послідовне виконання.

Представлення коду функції зчитування кадру:

```
static void read_frame_thread_int ( thread_input_arg_t *i )
{
        i-> status = i->h-> p_read_frame ( i->pic , i->h- > p_handle , i- >i_frame );
}
        int read_frame_thread ( h264_picture_t *p_pic , hnd_t handle , int i_frame
        )
{
        thread_input_t *h = handle ;
        UNUSED void * stuff ;
        int ret = 0;
        if( h-> next_frame >= 0 )
        {
# ifndef HAVE_OMPSS
                h264_pthread_join ( h- >tid , & stuff );
# else
                # pragma omp taskwait
# endif
                ret |= h-> next_args -> status ;
                h-> in_progress = 0;
        }
        if( h-> next_frame == i_frame )
        {
                XCHG ( h264_picture_t , *p_pic , h-> pic );
        }
        Else
```

```
        {
        ret |= h-> p_read_frame ( p_pic , h-> p_handle , i_frame ) ;
        }
        if( !h-> frame_total || i_frame +1 < h- > frame_total )
        {
                h-> next_frame =
                h-> next_args -> i_frame = i_frame +1;
                h-> next_args -> pic = &h-> pic ;
# ifndef HAVE_OMPSS
                h264_pthread_create ( &h- >tid , NULL , ( void *)
                        read_frame_thread_int , h-> next_args ) ;
# else
                # pragma omp task label ( read_frame_thread_int )
                read_frame_thread_int (h-> next_args );
# endif
                h-> in_progress = 1;
        }
        else
                h-> next_frame = -1;
        return ret;
}
```

Дивлячись на код, можна помітити, що насправді є дві функції. Перший (тобто read_frame_thread_int) фактично викликає функцію, яка була викликана. Це означає, що функція служить тільки для рекурсивного виклику вашої власної функції явно.

Інша функція, звана потоком кадру читання, насправді виконує читання кадру. Ця функція намагається фактично прочитати всі кадри. Ідея полягає в тому, щоб почати читання всього фрейму і після цього продовжити виконання програми. Для цього ми створюємо завдання для кожного кадру, виконуючи паралельне читання вхідного файлу, щоб уникнути втрати продуктивності.

Цікава частина коду стиснута між рядками 33 і 39. Як ви можете бачити, ми знову використовуємо умовну групу, щоб розділити вихідну реалізацію, яка використовує потоки pthread, від нашої, яка використовує OmpS. У разі, якщо макрос HAVE_OMPSS визначено, ми створюємо нову задачу, яка буде читати наступний кадр.

3.1.2 Кодування кадрів

Це найбільша модифікація, яка була зроблена в коді для створення нашої версії OmpS. Спочатку ми додали створення основного завдання, яка власне кодує кадр. Функція, яка дійсно це робить, називається h264_slices_write. Ця функція викликається для іншої, званої h264_encoder_encode, яка готує безліч структур даних, обов'язкових для кодування кадру.

Щодо функції h264_slices_write, то вона викликає інші для кодування кадру. Це ті функції, які поділяють кадр на макроблоки, аналізують їх, щоб вибрати кращий тип (тобто тип I, P або B) для кожного з них, і виконують кодування кожного макроблоку кадру. У наступному фрагменті коду показано створення завдання і код, завершальний процес кодування одного кадру.

Представлення коду створення завдання кодування кадру:

```
int h264_encoder_encode ( h264_t *h,
        h264_nal_t ** pp_nal , int * pi_nal ,
        h264_picture_t * pic_in ,
        h264_picture_t * pic_out )
{

        ...

        /* Запис кадру*/
        if( h-> param . i_threads > 1 )
        {
```

```
# ifndef HAVE_OMPSS
            h264_pthread_create ( &h- > thread_handle , NULL , ( void *)
                h264_slices_write , h );
# else
            # pragma omp task out (*h) label ( h264_slices_write )
            h264_slices_write (h);
# endif
            h-> b_thread_active = 1;
        }
        else {
            h264_slices_write ( h );
        }

        ...

        h264_encoder_frame_end ( thread_oldest , thread_current , pp_nal ,
            pi_nal , pic_out ) ;
        return 0;
}
```

Ми бачимо, що завдання створюється з функції h264_encoder_encode, як було пояснено раніше. Змінена частина укладена в рядки з 12 по 17. Ми використовуємо умовну групу, щоб перевірити, чи ми використаємо OmpS чи ні, і якщо так, ми створюємо завдання, яка буде виконувати h264_lices_write. Це завдання буде мати вихідне значення і настроюється мітку, яка дозволить нам поліпшити файли трасування, які ми створимо пізніше.

Нарешті, в рамках тієї ж функції ми закриємо кодировщик кадрів, це означає, що ми очистимо всі дані, які не будуть використовуватися знову, щоб уникнути витоків пам'яті. Код функції h264_encoder_frame_end наступний:

Представлення коду закриття кодувальника кадрів:

```
static void h264_encoder_frame_end ( h264_t *h, h264_t *
```

```
        thread_current ,
                h264_nal_t ** pp_nal , int *
                        pi_nal ,
                h264_picture_t * pic_out )
{
        int i, i_list ;
        char psz_message [80];

        if( h-> b_thread_active )
        {
 # ifndef HAVE_OMPSS
                h264_pthread_join ( h- > thread_handle , NULL );
# else
                # pragma omp taskwait on (*h)
# endif
                h-> b_thread_active = 0;
        }
        if( !h-> out . i_nal )
        {
                pic_out -> i_type = H264_TYPE_AUTO ;
                return ;
        }

        ...

}
```

Ця функція чекає, поки завдання, щойно створена для кодування кадру, що не буде завершена. Це корисно, тому що ми постійно перевіряємо, закодований чи вже кадр, щоб виконати деякі операції, що вимагають цього умови. Так що це єдиний спосіб переконатися, що один кадр вже закодований. Це дозволяє нам

очищати структури даних або повторно використовувати їх.

Тепер ми збираємося заглянути всередину функції h264_slices_write. Наступний код показує його вміст і важливий код функції h264_slice_write, яка викликається з іншого:

Представлення коду функції h264_slice_write:

```
static int h264_slices_write ( h264_t *h )
{
        int i_frame_size ;

        ...

        h264_stack_align ( h264_slice_write , h );
        i_frame_size = h-> out . nal [h- >out.i_nal -1]. i_payload ;

        ...

        h-> out . i_frame_size = i_frame_size ;
        return 0;
}

static void h264_slice_write ( h264_t *h )
{
        int i_skip ;
        int mb_xy , i_mb_x , i_mb_y ;
        int i, i_list , i_ref ;

        ...

        while ( ( mb_xy = i_mb_x + i_mb_y * h->sps -> i_mb_width ) < h- >sh.
```

```
        i_last_mb )
{
        int mb_spos = bs_pos (&h-> out.bs) + h264_cabac_pos (&h->
        cabac);

        if( i_mb_x == 0 ) {
                h264_fdec_filter_row ( h, i_mb_y );
        }

        /* load cache */
        h264_macroblock_cache_load ( h, i_mb_x , i_mb_y );

        /* аналізуємо параметри
        * Слой I: обираємо режим I_4x4 або I_16x16
        * Слой P: обираємо режим між P або внутрішнім (4 x4 or 16
        x16 )
        * */
        h264_macroblock_analyse ( h );

        /* encode this macroblock -> be careful it can change the
                mb type to P_SKIP if needed */
        h264_macroblock_encode ( h );

        h264_bitstream_check_buffer ( h );

        ...

        /* save cache */
        h264_macroblock_cache_save ( h );
```

```
            ...

            h264_ratecontrol_mb ( h, bs_pos (&h-> out .bs) + h264_cabac_pos
                (&h-> cabac ) - mb_spos );

            ...

        }

        ...

        h264_fdec_filter_row ( h, h->sps -> i_mb_height ) ;

        ...

}
```

У рядку 7 ви можете побачити, що викликається функція h264_slice_write. Насправді цей виклик є для нас найбільш важливою частиною функції h264_slices_write, тому що решта код призначений для оновлення зображення на дисплеї в разі, якщо ми використовували функцію візуалізації додатки на льоту.

Отже, дивлячись на функцію h264_slice_write, ми бачимо, що після виконання інших дій, таких як підготовка змінних, в якийсь момент виконання перейде в оператор циклу. Цей цикл дозволяє виконувати одні й ті ж операції для кожного макроблоку кадру. Заглянувши в оператор циклу, ми побачимо кілька викликів функцій, які важливі для пояснення того, як працює відеокодер.

Найперша рядок - h264_fdec_filter. Ця функція в основному оновлює статус процесу кодування для цього кадру, щоб дозволити іншим задачам (які будуть кодувати інші кадри) мати можливість перевірити, чи можуть вони продовжити його виконання (тобто синхронізацію). Отже, перше, що нам потрібно зробити, це обновити поточний статус процесу кодування, після цього ми завантажимо

макроблок, а потім проаналізуємо цей макроблок за допомогою функції h264_macroblock_analyse. Саме під час виконання цієї функції ми будемо перевіряти стан кадру або кадрів, від яких залежить поточний кадр. У разі, якщо стану цих кадрів недостатньо для початку аналізу макроблоку, ми почекаємо, поки ці кадри не закодують достатню кількість макроблоків, а потім продовжимо аналіз кадру.

Є й інші функції, які кодують макроблок, зберігають його і оновлюють деякі дані, які будуть використовуватися згодом. Звичайно, перед виходом з h264_slice_write обов'язково оновити статус процесу кодування цього кадру. Це робиться повторним викликом функції h264_fdec_filter_row.

Тепер ми збираємося вивчити функцію h264_fdec_filter_row, щоб побачити, як оновлюється статус. Це важливо, тому що там є певний код, який був змінений, щоб змусити працювати версії OmpS. Найважливіші функції полягають в наступному:

Представлення коду функції h264_fdec_filter_row:

```
static void h264_fdec_filter_row ( h264_t *h, int mb_y )
{

    ...

    if( h-> param . i_threads > 1 && h- >fdec -> b_kept_as_ref )
    {
# ifndef HAVE_OMPSS
        h264_frame_cond_broadcast ( h->fdec , mb_y *16 + ( b_end ?
        10000 : -( H264_THREAD_HEIGHT << h->sh. b_mbaff ) ) ) ;
# else
        # pragma omp atomic
        h->fdec -> i_lines_completed = mb_y *16 + ( b_end ? 10000 : -(
            H264_THREAD_HEIGHT << h->sh. b_mbaff ));
```

```
# endif
        }

        ...

}
```

Як можна побачити, це оновлення виконуватимуться лише тоді, коли кількість потоків більше одного. Це означає, що якщо ми запускаємо додаток тільки з одним потоком, ми не будемо виконувати будь-яку синхронізацію тільки тому, що вона взагалі не потрібна.

Крім цього, умовна група була використана для отримання нашою версією OmpS. Ми бачимо, що оновлення статусу полягає тільки в оновленні однієї змінної, до якої згодом будуть звертатися інші завдання. З цієї причини цей доступ повинен бути захищений, щоб уникнути стану гонки. Ми використовуємо атомарний оператор OmpSs для захисту доступу до змінної.

А тепер подивимося, як з цієї змінної звертаються до функцій h264_macroblock_analyse. Доступ до цієї змінної фактично здійснюється у функції, що викликається з h264_macroblock_analyse з ім'ям h264_mb_analyse_init. Обидві функції показані в наступному фрагменті коду. Надаються тільки важливі частини коду.

Предствалення коду функцій h264_macroblock_analyse і h264_mb_analyse_init:

```
void h264_macroblock_analyse ( h264_t *h )
{
        h264_mb_analysis_t analysis ;
        int i_cost = COST_MAX ;
        int i;

        ...
```

```
        h264_mb_analyse_init ( h, & analysis , h->mb. i_qp );

        /* - - - Робимо аналіз - - -*/

        ...

}

static void h264_mb_analyse_init ( h264_t *h, h264_mb_analysis_t *a,
        int i_qp )
{
        int i = h-> param . analyse . i_subpel_refine - (h->sh. i_type ==
                SLICE_TYPE_B );

        ...

        /* I: Intra part */

        ...

        /* II: Inter part P/B кадрів */
        if( h->sh. i_type != SLICE_TYPE_I )
        {
                int i, j;
                int i_fmv_range = 4 * h-> param . analyse . i_mv_range ;
                int i_fpel_border = 5; // umh u nc ond it io na l radius
                int i_spel_border = 8; // 1.5 for subpel_satd , 1.5 for
                        subpel_rd , 2 for bime , round up
```

```
/ * Обчислюємо максимально допустимий діапазон для повноекранного відео (FMV) * /

...

if( h->mb. i_mb_x == 0)
{
    int mb_y = h->mb. i_mb_y >> h->sh. b_mbaff ;
    int mb_height = h->sps -> i_mb_height >> h->sh. b_mbaff ;
    int thread_mvy_range = i_fmv_range ;

    if( h-> param . i_threads > 1 )
    {
        int pix_y = (h- >mb. i_mb_y | h->mb. b_interlaced ) *
        16;
        int thresh = pix_y + h-> param . analyse .
            i_mv_range_thread ;

        for ( i = (h->sh. i_type == SLICE_TYPE_B ); i >= 0;
        i -- )
        {
            h264_frame_t ** fref = i ? h- > fref1 : h-> fref0;
            int i_ref = i ? h-> i_ref1 : h-> i_ref0 ;
            for ( j =0; j< i_ref ; j++ )
            {
# ifndef HAVE_OMPSS
                h264_frame_cond_wait ( fref [j],
                thresh );
# else
                while ( fref [j]- >
```

```
                                        i_lines_completed < thresh )
                                        {
                                        # pragma omp taskwait
                                    }
# endif
                                    thread_mvy_range = H264_MIN (
                                    thread_mvy_range , fref [j]- >
                                    i_lines_completed - pix_y ) ;
                                }
                            }

                            ...

                        }

                        ...

                    }
# undef CLIP_FMV

                        ...

                    }

                    ...

            }
}
```

У рядку 9 ми бачимо, що h264_mb_analyse_init викликається, і він починає

виконання. Потім важлива частина міститься в рядках 56 і 62. Ми можемо бачити там іншу умовну групу, і в разі, якщо HAVE_OMPSS визначено, ми перевіримо, чи достатньо статусу фрейму, який у нас є, щоб дозволити нам продовжити. . Якщо немає, ми виконаємо Taskwait, щоб перевірити його пізніше. Як тільки перевірка буде позитивною, ми продовжимо аналіз макроблоку. Ця перевірка буде виконуватися для кожного кадру, від якого у нас буде залежність.

Це не найкращий варіант реалізації цієї синхронізації в рамках моделі програмування OmpS, оскільки основна проблема цієї реалізації - необхідність синхронізувати різні завдання, які не видно один одному. Модель програмування OmpS орієнтована на встановлення вхідних і вихідних залежностей, але в нашому додатку нам буде потрібно синхронізація зі станом однієї змінної, тому створити цю залежність за допомогою операторів OmpS не так просто.

Одним з можливих рішень буде використання дозорного (тобто змінної, яка нічого не робить, крім створення підроблених відносин залежності), але ми вирішуємо цього не робити. Причина в тому, що використання часових в OmpS не рекомендується, так як ви можете зіткнутися з проблемами, якщо поліпшите час виконання.

Останньою модифікацією додатки було додавання черзі завдань при завершенні процесу кодування для кожного кадру. Це обов'язково, так як вам потрібно оновити деякі параметри і звільнити деякі ресурси, щоб використовувати їх згодом. Код виконуваної функції наступний.

Представлення коду функції h264_encoder_frame_end:

```
static void h264_encoder_frame_end ( h264_t *h, h264_t *
        thread_current ,
                h264_nal_t ** pp_nal , int *
                        pi_nal ,
                h264_picture_t * pic_out )
{
        int i, i_list ;
```

```
        char psz_message [80];

        if( h-> b_thread_active )
        {
# ifndef HAVE_OMPSS
                h264_pthread_join ( h- > thread_handle , NULL );
# else
                # pragma omp taskwait on (*h)
# endif
                h-> b_thread_active = 0;
        }
        if( !h-> out . i_nal )
        {
                pic_out -> i_type = H264_TYPE_AUTO ;
                return ;
        }

        h264_frame_push_unused ( thread_current , h-> fenc );

        ...

        /* - - - Оновлення стану кодувальника - - - */

        ...

}
```

Як видно в рядках з 10 по 14, оголошена умовна група, і в разі, якщо макрос HAVE_OMPSS визначено, ми просто виконуємо завдання для змінної h. Це означає, що ми будемо чекати тільки завершення завдання h264_slices_write цього процесу кодування. В рамках функції виконується більше роботи, але єдина

важлива частина для нас - це та, про яку ми вже говорили.

3.2 Тестування роботи програми

Перед запуском програми треба перемістити файл з програмою до папки з відеофайлами. Для запуску завдання на кодування файлів треба відкрити консольну програму, яку користувач будемо використовувати для ущільнення (рисунок 3.1).

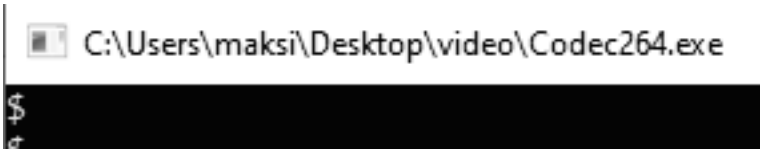

Рисунок 3.1 — Запуск консольної програми для ущільнення відеофайлів

Для використання програми потрібно команду наступного формату: --output <outputfile> <inputfile>. Для одночасного виконання ущільнення делькох файлів (а також для демонстрації їх одночасної обробки), краще за все буде вставка попередньозаписаних команд (рисунок 3.2).

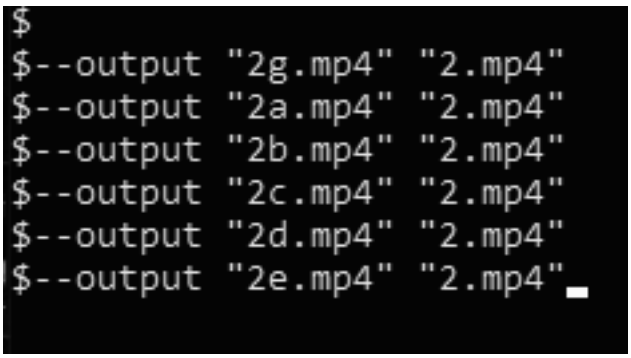

Рисунок 3.2 — Використання команд для багатоканального ущільнення декількох файлів

Для варіювання якості на різних кадрах для досягнення найкращої якості послідовності при заданому бітрейті потрібно вказати команду наступного формату: -- bitrate <integer> (рисунок 3.3).

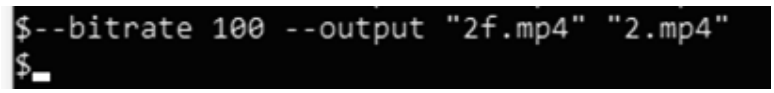

Рисунок 3.3 — Використання команди бітрейту

При подання завдань на обробку файлів створюються окремі консолі для кожного завдання. В відкритих вікнах можна побачити інформацію про якість вхідного файлу, скільки залишилось часу до кінця обробки, швидкість обробки кадрів, а також число оброблених кадрів (рисунок 3.4).

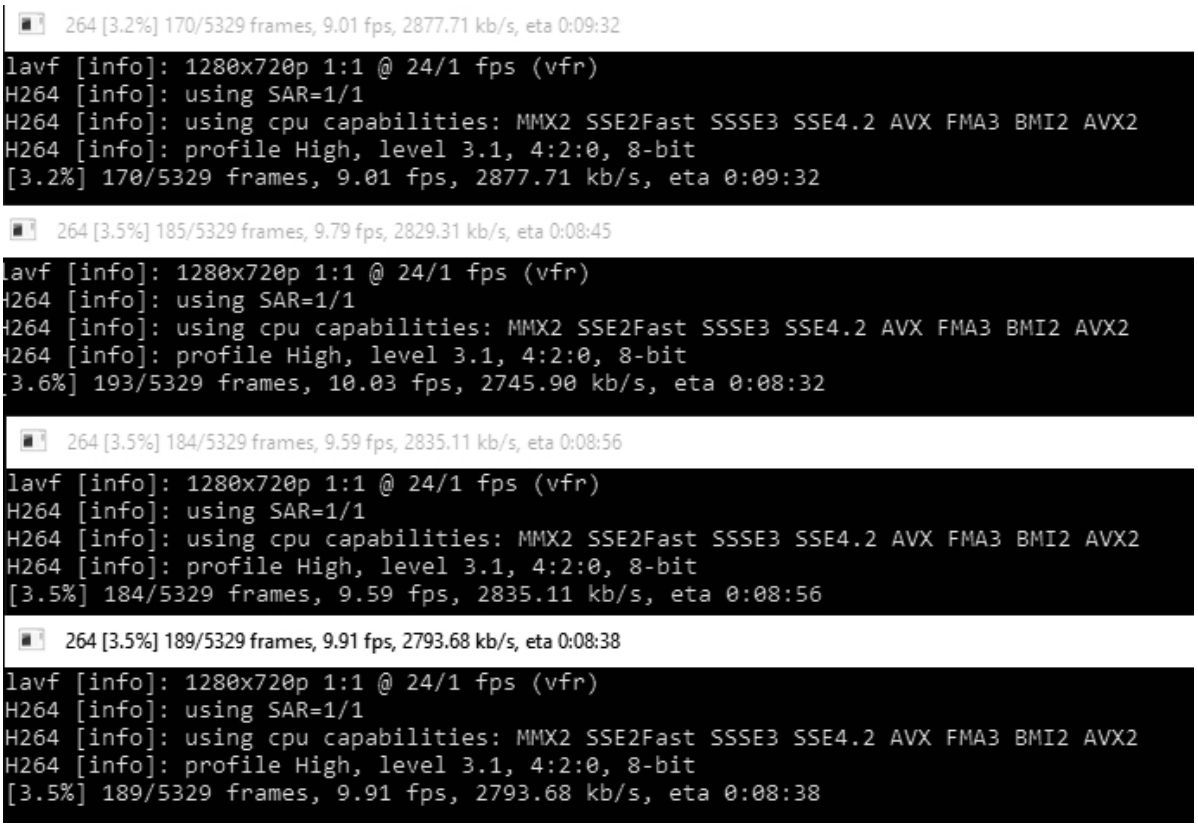

Рисунок 3.4 — Виконання декількох завдань в окремих консолях

Також варто замітити, що чим більше кількість файлів відправлених на обробку, тим повільніше проходить їх ущільнення, оскільки обробка відбувається одночасно, а не послідовно.

Після закінчення обробки фали будуть знаходитись в папці зі всіма об’єктами. В залежності від якості ущільнення кінцевий файл буде відрізнятись по розміру, тобто чим сильніше відео було ущільнення (нище бітрейт), тим менше воно буде по розмірам (рисунок 3.5). Також роль грає якість вхідного файлу: якщо воно вже було ущільнено, то різниця між початковим і кінцевим не буди значною.

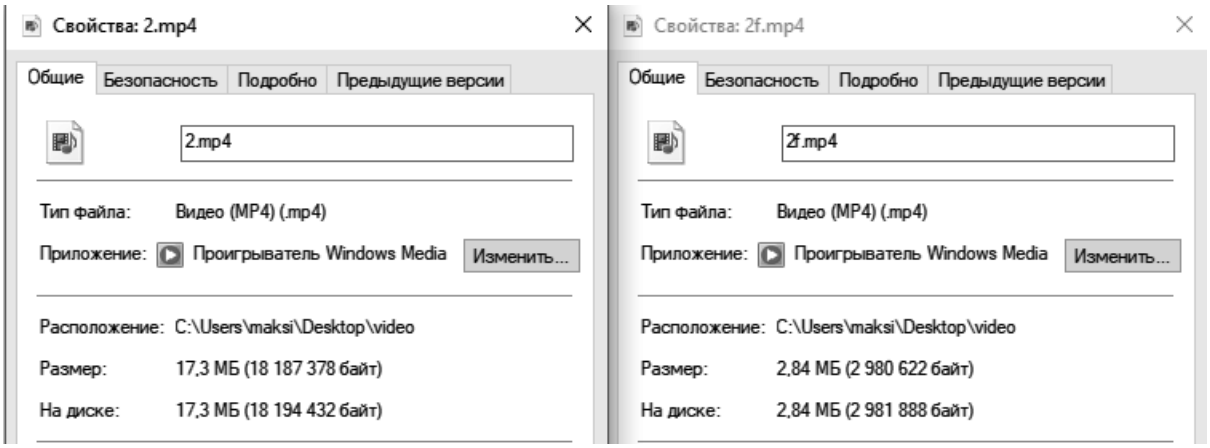

Рисунок 3.5 — Властивості вхідного та кінцевого відеофайлів

4 РОЗРАХУНОК ЕКОНОМІЧНОЇ ДОЦІЛЬНОСТІ СТВОРЕННЯ ПРОГРАМИ УЩІЛЬНЕННЯ БАГАТОКАНАЛЬНОГО ВІДЕОЗОБРАЖЕННЯ ДЛЯ СИСТЕМ КОМП'ЮТЕРНОГО МОНІТОРИНГУ

Проведення науково-дослідної роботи будь-якого типу завжди вимагає певних витрат. Витрати на виробництво та реалізацію продукту увесь час повинні зменшуватись, оскільки у цьому й полягає прогрес будь-якого виробництва. Якщо витрати не зменшуються, то ніяка науково-технічна розробка не буде реалізована на практиці, оскільки така розробка не буде більш ефективна, аніж існуючі на ринку аналоги.

На основі економічних розрахунків, можна довести економічну доцільність та ефективність впровадження результатів, що були отримані в результаті виконаних науково-дослідних робіт у виробництві, тобто здійснити комерціалізацію наукових розробок.

Саме цим завданням присвячений даний розділ магістерської кваліфікаційної роботи і він передбачає виконання таких етапів робіт (рис. 4.1):

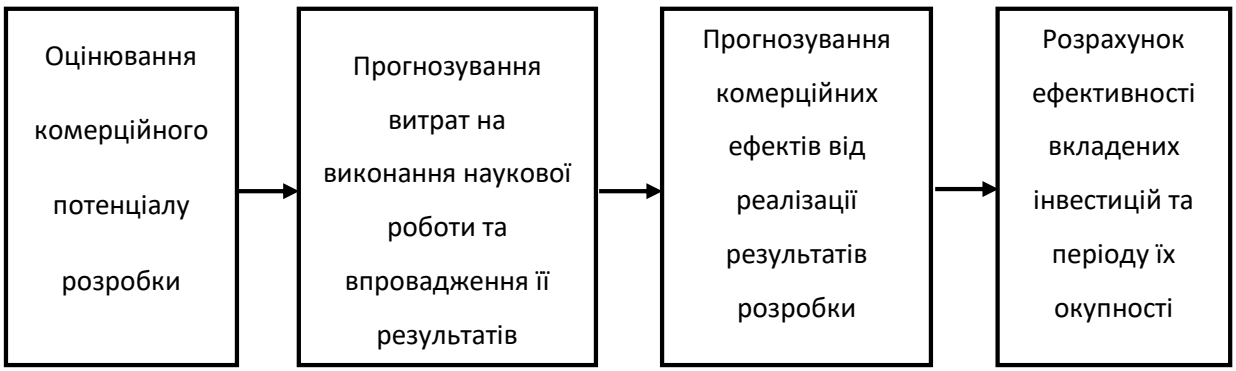

Рисунок 4.1 — Складові економічної частини магістерської кваліфікаційної роботи

На такі складові буде поділено економічну частину даної магістерської роботи. Усі подальші економічні розрахунки, будуть висвітлені у згаданих підрозділах економічної частини.

4.1 Технологічний аудит розробки

В даному підрозділі буде проведено оцінювання комерційного потенціалу розробки, створеної в результаті науково-технічного дослідження. За результатами оцінювання робиться висновок щодо напрямів організації в майбутньому та її впровадження з врахуванням встановленого рейтингу.

Оцінювання комерційного потенціалу розробки будемо здійснювати за дванадцятьма критеріями, наведеними у таблиці 4.1.

Таблиця 4.1 – Оцінювання комерційного потенціалу розробки

Критерії оцінювання та бали (за 5-бальною шкалою)					
Критерій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Багато аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
Практична здійсненність					
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Кінець таблиці 4.1

Критерії оцінювання та бали (за 5-бальною шкалою)					
Критерій	0	1	2	3	4
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні.	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідно регламентні документи та велика кількість дозвільних документів на виробництво продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

На основі складеної таблиці ряд незалежних експертів, у нашому випадку керівник магістерської роботи та викладачі випускової кафедри поставили різні бали. Результати цього оцінювання комерційного потенціалу занесено до таблиці 4.2.

Таблиця 4.2 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	1 Крупельницький Л. В., к.т.н., доц. кафедри ОТ	2 Очкуров М. А., к.т.н., доц. кафедри ОТ	3 Семеренко В. П., к.т.н., доц. кафедри ОТ
	Бали, виставлені експертами:		
1	3	3	3
2	4	2	3
3	4	4	4

Кінець таблиці 4.2

4	3	4	2
5	3	4	3
6	3	3	3
7	4	3	2
8	4	4	3
9	3	2	3
10	3	3	3
11	4	4	3
12	3	3	4
Сума балів	$СБ_1 = 41$	$СБ_1 = 39$	$СБ_1 = 36$
Середньо-арифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{41 + 39 + 36}{3} = \frac{116}{3} = 38.7$		

За даними таблиці 4.2, а також згідно із рекомендаціями, що наведені в таблиці 4.3, можна зробити висновок, щодо рівня комерційного потенціалу розробки.

Таблиця 4.3 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 - 10	Низький
11 - 20	Нижче середнього
21 - 30	Середній
31 - 40	Вище середнього
41 - 48	Високий

Взявши до уваги, середньоарифметичну суму балів, $\overline{СБ} = 38{,}7$, що були виставлені експертами, можна стверджувати, що рівень комерційного потенціалу даної розробки - вище середнього.

Для порівняння властивостей було взято Total Video Converter. Дана програма має більш широкий спектр використання. В свою чергу нова розробка є вузько спеціалізованою, через що більш ефективною. Аналог розповсюджується по платній підписці у той час, як плата за користування програмою, яка є продуктом розробки, є одноразовою.

Таблиця 4.4 — Порівняння характеристик розробки із аналогом

Показники	**Розробка**	**Аналог**
Функціонал	7	9
Швидкодія	9	7
Надійність	9	6
Метод розповсюдження	8	6
Інтерфейс, простота використання	8	8

Просування продукту здійснюватиметься засобами цифрової дистрибуції. За допомогою аналітики можна буде спрямувати на цільову аудиторію – телерадіокомпанії цифрового телевізійного мовлення.

Новизною розробки є вдосконалення методу ущільнення відео зображення зі збільшенням потоків обробки, що дає змогу більш швидко обробляти інформацію та займати менше місця на носіях пам’яті. Дана характеристика позитивно впливатиме на конкурентоспроможність продукту.

Виходячи з результатів даного порівняння можна зробити висновок, що нова розробка є конкурентоспроможною, оскільки по деяким параметрам переважає один з найкращих аналогів на ринку.

4.2 Прогнозування витрат на виконання та впровадження результатів наукової роботи

У магістерській кваліфікаційній роботі розглядається програмне забезпечення для ущільнення багатоканального відеозображення, тому значна частина витрат - це витрати на розробку, а не на виробництво і відтворення. Звідси, й певна специфіка розрахунків.

Основна заробітна плата розробників, які працюють над проектом визначається за формулою 4.1:

$$Зо = \frac{М}{Тр} \cdot t \text{ [грн]}, \quad (4.1)$$

де М – місячний посадовий оклад розробника;

Тр – число робочих днів в місяці (T_p= 22дні);

t – число днів роботи розробника.

Над створенням розробки працювали керівник проекту та інженер-програміст , отже, виконаємо для них всі необхідні розрахунки:

$$Зо = \frac{12\,000}{22} \cdot 7 = 3\,818{,}20\ (грн),$$

$$Зо = \frac{10\,000}{22} \cdot 55 = 25\,000\ (грн).$$

Таблиця 4.5 – Заробітна плата

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
1 Керівник	12 000	545,45	7	3 818,20
2 Старший інженер-програміст	10 000	454,55	55	25 000
Всього				$\sum Зо = 28\,818{,}20$

Додаткова заробітна плата (Зд) всіх розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховується як (10…12)% від суми основної заробітної плати всіх розробників та робітників, тобто:

$$Зд = (10 \ldots 12\%) * Зо\ [грн], \qquad (4.2)$$

де Зо - основана заробітна плата.

$$Зд = \frac{10 \cdot 28\,818{,}20}{100} = 2\,881{,}82(грн).$$

Нарахування на заробітну плату (Нзп) розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою:

$$Нзп = 22\% \cdot (Зо + Зд)[грн], \quad (4.3)$$

$$Нзп = \frac{22 \cdot (28\,818{,}20 + 2\,881{,}82)}{100} = 6\,974\ (грн).$$

Амортизація обладнання, комп'ютерів та приміщень (А), які використовувались під час виконання даного етапу роботи.

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо.

У спрощеному вигляді амортизаційні відрахування (А) в цілому бути розраховані за формулою 4.4:

$$А = \frac{Ц \cdot На}{100} \cdot \frac{Т}{12}[грн], \quad (4.4)$$

де Ц - загальна балансова вартість всього обладнання, комп'ютерів, приміщень тощо, що використовувались для виконання даного етапу роботи, грн;

На - річна норма амортизаційних відрахувань. Для нашого випадку можна прийняти, що На = (10...25)%;

Т– термін, використання обладнання, приміщень тощо, місяці.

Всі розрахунки зводимо до таблиці 4.5.

Таблиця 4.6 – Амортизація обладнання та приміщень

Найменування обладнання, приміщень	Балансова вартість, грн.	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань, грн.
ЕОМ	10 000	20%	3	250
Приміщення	120 000	15%	3	450
Всього				700

У магістерській кваліфікаційній роботі розробляється програмний продукт, який для споживача буде поширюватися через інтернет, а саме через веб-сторінку. При розробці сторінки до послуг виробничого характеру сторонніх підприємств

можна віднести надавання послуги «Хостинг», а також направлення обраного доменного імені на сервери хосту, тому на протязі всього існування проекту за ці послуги, щорічно потрібно сплачувати абонентську плату.

Таблиця 4.7 — Послуги, що використовуються при виготовленні програми

Найменування комплектуючих (робіт, послуг)	Кількість, шт.	Ціна за одиницю, грн.	Сума, грн.
1. Послуга «Хостинг», шт.	1	300	300
2. Послуга «Доменне ім’я», шт.	1	150	150
Всього			450 грн.

Витрати на силову електроенергію $В_е$, якщо ця стаття має суттєве значення для виконання даного етапу роботи, розраховуються за формулою 4.6:

$$В_е = В \cdot П \cdot Ф \cdot К_п [грн], \quad (4.6)$$

де В – вартість 1 кВт електроенергії, грн.;

П – установлена потужність обладнання, кВт/год;

Ф – фактична кількість годин роботи обладнання, яке задіяне на виготовлення одного виробу, годин;

$К_п$ –коефіцієнт використання потужності, $К_п \leq 1$.

$$В_е = 3{,}35 \cdot 0{,}09 \cdot 230 \cdot 0{,}7 = 48{,}5 \text{ (грн)}.$$

Інші витрати охоплюють: загально виробничі витрати (витрати управління організацією, ремонт та експлуатація основних засобів, витрати на опалення, освітлення тощо), адміністративні витрати (проведення зборів, оплата юридичних та аудиторських послуг, тощо), витрати на збут (витрати на рекламу, перепідготовка кадрів) на інші операційні витрати (штрафи, пені, матеріальні допомоги, втрати від знецінення запасів тощо).

Інші витрати $І_в$ можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто:

$$І_в=(1..3)\cdot(З_о+З_р) \quad (4.7)$$

Отже, розрахуємо інші витрати:

$$І_в = 1 \cdot (28\,818{,}20 + 2\,881{,}82) = 31\,700 \text{ грн.}$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини (розділу, етапу) роботи – В.

$$В = 28\,818{,}20 + 2\,881{,}82 + 6\,974 + 700 + 450 + 113 + 31\,700 = 71\,637(\text{грн}).$$

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою:

$$ЗВ = \frac{В_{заг}}{\beta}[\text{грн}], \quad (4.8)$$

де β–коефіцієнт, який характеризує етап (стадію) виконання даної роботи. Оскільки, розробка знаходиться на стадії впровадження, то $\beta \approx 0{,}9$;

$В_{заг}$–загальна вартість всієї наукової роботи. У даному випадку $В_{заг} = В$.

$$ЗВ = \frac{71\,637}{0{,}9} = 79\,596{,}7(\text{грн}).$$

Отже, розрахований кошторис витрат на розробку програмного забезпечення для формування ознак об’єктів у цифровому зображенні складає 79 596,7 грн.

4.3 Прогнозування комерційних ефектів від реалізації результатів розробки

У даному підрозділі здійснено прогнозування, яку вигоду можна отримати у майбутньому від впровадження результатів даної наукової роботи.

Передбачається, що виконання наукової роботи та впровадження результатів по розробці програмного забезпечення для ущільнення багатоканального відеозображення займе 1 рік.

Основні позитивні результати від впровадження розробки очікуються протягом 3 років після її впровадження. Саме зростання чистого прибутку забезпечить підприємству (організації) надходження додаткових коштів, які дозволять покращити фінансові результати діяльності.

$$\Delta П_{\text{і}} = \sum_{1}^{n} (\Delta П_{я} \cdot N + П_{я} \Delta N)_i \, , [\text{грн}], \tag{4.9}$$

де $\Delta П_{я}$ - покращення основного якісного показника від впровадження результатів розробки у даному році;

N - основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN - покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_{я}$ - основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n - кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

В результаті впровадження результатів наукової розробки покращується якість програмного продукту, що дозволяє підвищити ціну його реалізації на 450 грн., а кількість потенційних користувачів ресурсу збільшиться: протягом першого року - на 150 шт., протягом другого року - ще на 105 шт., протягом третього року - ще на 50 шт.

Орієнтовно: реалізація продукції до впровадження результатів наукової розробки складала 1 шт., а прибуток, що його отримувало підприємство на одиницю продукції до впровадження результатів наукової розробки – 700 грн.

Спрогнозуємо збільшення чистого прибутку підприємства від впровадження результатів наукової розробки у кожному році відносно базового.

З

б

і

л

ь

б

і

л

ь

ш

е

н

н

я

ч

и

с

т

о

г

о

п

р

и

б

у

т

к

у

п

і

д

п

р

и

є

м

с

т

в

а

в

і

д

в

п

р

о

в

а

д

ж

е

н

н

я

р

е

з

у

л

ь

т

а

т

і

в

н

а

у

к

о

в

о

ї

р

о

з

р

о

б

к

и

у

к

о

ж

н

о

м

у

р

о

ц

і

в

і

д

н

о

с

н

о

б

а

з

о

в

о

г

о

:

$$\Delta П_1 = 700 + (700 + 450) \cdot 150 = 173\,200(\text{грн}).$$

З

б

і

л

ь

ш

е

н

н

я

ч

и

с

т

о

г

о

п

р

и

б

у

т

к

у

п

і

д

п

р

и

є

м

с

т

в

а

в

і

д

в

п

р

о

в

а

д

ж

е

н

н

я

р

е

з

у

л

ь

т

а

т

і

в

н

а

у

к

о

в

о

ї

р

о

з

р

о

б

к

и

у

д

р

у

г

о

м

у

р

о

ц

і

в

і

д

н

о

с

н

о

б

а

з

о

в

о

г

о

:

$$\Delta П_2 = 700 + (700 + 450) \cdot 105 = 121\,450(\text{грн}).$$

З

б

і

л

ь

ш

е

н

н

я

ч

и

с

т

о

г

о

п

р

и

б

у

т

к

у

п

і

д

п

р

и

є

м

с

т

в

а

в

і

д

в

п

р

о

в

а

д

ж

е

н

н

я

р

е

з

у

л

ь

т

а

т

і

в

н

а

у

к

о

в

о

ї

р

о

з

р

о

б

к

и

у

т

р

е

т

ь

о

м

у

р

о

ц

і

в

і

д

н

о

с

н

о

б

а

з

о

в

о

г

о

:

$$\Delta П_3 = 700 + (700 + 450) \cdot 50 = 58\,200(\text{грн}).$$

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахований комерційний ефект можливого впровадження розробки однак не є гарантією, що розробка буде впроваджена. Якщо збільшення прогнозованого прибутку від впровадження результатів наукової розробки є вигідним для підприємства, то це ще не означає, що вона зацікавить інвесторів. Основні показники, що визначають рентабельність фінансування розробки певним інвестором, є відносна і абсолютна ефективність вкладених інвестицій та термін їх окупності.

Розрахунок ефективності вкладених інвестицій передбачає проведення таких робіт.

Першим кроком розраховуємо теперішню вартість інвестицій PV, що вкладаються в наукову розробку. Такою вартістю, можна вважати прогнозовану

величину загальних витрат ЗВ на виконання та впровадження результатів НДДКР, розраховану нами раніше за формулою 4.8, тобто будемо вважати, що ЗВ = PV=79 596,7.

Другим кроком розрахуємо очікуване збільшення прибутку $\Delta П_i$, що його отримає підприємство (організація) від впровадження результатів наукової розробки, для кожного із років, починаючи з першого року впровадження.

Третім кроков для спрощення подальших розрахунків побудуємо вісь часу, на яку нанесемо всі платежі (інвестиції та прибутки), що мають місце під час виконання науково-дослідної роботи та впровадження її результатів.

Платежі показуються у ті терміни, коли вони здійснюються. Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, наведений на рис. 4.2.

Четвертим кроком розрахуємо абсолютну ефективність вкладених інвестицій $Е_{абс}$. Для цього користуються формулою:

$$Е_{абс} = (ПП - PV), \quad (4.10)$$

де ПП - приведена вартість всіх чистих прибутків, що їх отримає підприємство (організація) від реалізації результатів наукової розробки, грн;

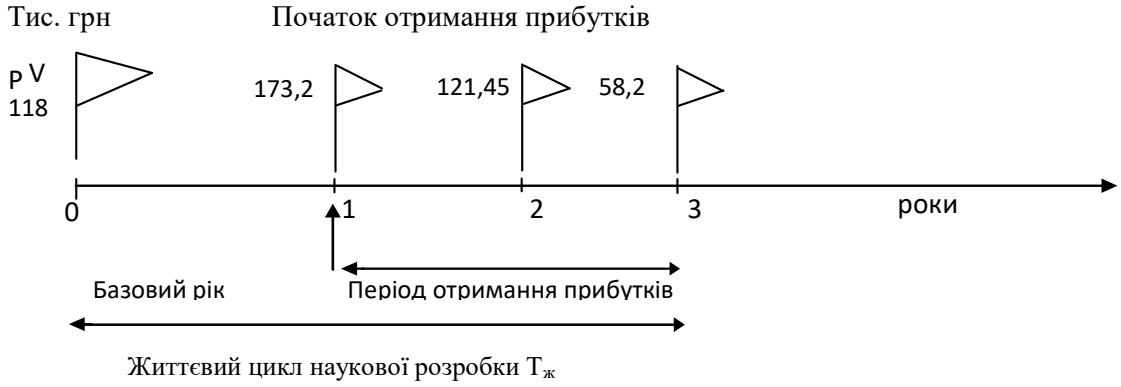

Рисунок 4.2 — Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

PV – теперішня вартість інвестицій PV = ЗВ= 79 596,7(грн).

У свою чергу, приведена вартість всіх чистих прибутків ПП розраховується

за формулою:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, [грн], \quad (4.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

Т – період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки «0».

Отримаємо:

$$ПП = \frac{173\,200}{(1+0{,}1)^1} + \frac{121\,450}{(1+0{,}1)^2} + \frac{58\,200}{(1+0{,}1)^3} = 171\,485{,}15 + 119\,056{,}96 + 56\,488{,}35 = 347\,030{,}46$$

Тоді $Е_{абс} = (347\,030{,}46 - 79\,596{,}7) = 267\,433{,}76$(грн).

Оскільки $Е_{абс} > 0$, то результат від проведення наукових досліджень та їх впровадження може принести прибуток, але це також ще не гарантує те, що інвестор зацікавиться у фінансуванні даної роботи.

П’ятим кроком розраховують відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$. Для цього користуються формулою:

$$E_в = \sqrt[Т_ж]{1 + \frac{E_{абс}}{PV}} - 1 \quad (4.12)$$

де $E_{абс}$ - абсолютна ефективність вкладених інвестицій, грн;

PV - теперішня вартість інвестицій PV = ЗВ, грн;

$Т_ж$ - життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[з]{1 + \frac{267\,433{,}76}{79\,596{,}7}} - 1 = 1{,}64 - 1 = 64\ \%$$

Далі, розрахована величина $E_в$ порівнюється з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатись не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$визначається за формулою:

$$t = d + f\ [\%], \quad (4.13)$$

де d - середньозважена ставка за депозитними операціями в комерційних банках; в 2019 році в Україні d = (0,19...0,22);

f - показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,1), але може бути і значно більше.

$$t = d + f = 0{,}20 + 0{,}1 = 0{,}30 = 30\%$$

Величина $E_в > \tau_{мін}$, інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених коштів у реалізацію наукового проекту за формулою:

$$T_{ок} = \frac{1}{E_в} [років], \quad (4.14)$$

$$T_{ок} = \frac{1}{0,64} = 1,56 \text{ роки.}$$

Оскільки $T_{ок}$ = 1,56 роки, то розробка являється доцільною.

4.5 Висновки економічного ґрунтування

У даному розділі магістерської кваліфікаційної роботи здійснено розрахунки, які доводять прибутковість та ефективність впровадження нового продукту.

Проведено оцінювання комерційного потенціалу розробки. На основі компетентної думки експертів було сформовано систему критеріїв та за 5-ти бальною шкалою, виставлено бали по кожному з них. Виставлені бали, показують, що рівень комерційного потенціалу є вище середнього.

Розраховано витрати на розробку. Розрахований кошторис витрат на розробку склав 79 596,7 грн.

Були спрогнозовані комерційні ефекти від реалізації розробки, тобто який дохід, можна отримати у майбутньому від впровадження виконаної наукової роботи. Доведено, що розробка отримає вигоду від впровадження.

Розраховано основі показники, які визначають доцільність фінансування наукової розробки інвестором. Такими показниками є абсолютна та відносна ефективність вкладених інвестицій, а також термін їх окупності.

Обрахована абсолютна ефективність становить 267 433,76 грн, що свідчить про те, що інвестор буде зацікавлений у фінансуванні даної розробки. Відносна (щорічна) ефективність становить 64%, що більше мінімальної ставки дисконтування, що ще раз підтверджує зацікавленість інвестора.

Термін окупності вкладених коштів у реалізацію наукового проекту становить 1,56 роки, що означає, що вкладені кошти повернуться, приблизно, через 19 місяців.

Отже, можна стверджувати, що фінансування даної розробки є доцільним.

ВИСНОВКИ

Основний результат дипломної роботи полягає у створенні спеціалізованого програмного забезпечення для комп’ютерної системи цифрового телевізійного мовлення, задачею якого є ущільнення багатоканального відеозображення.

В роботі, згідно з поставленими задачами, виконано наступне.

В першому розділі проаналізовані методи ущільнення відеозображення, їх порівняння та обрано стандарт для реалізації поставленої задачі.

В другому розділі розроблено специфічний метод на основі стандарту H.264 та який враховує специфіку ущільнення багатоканального відеозображення.

В третьому розділі розробити і проаналізувати програмне забезпечення для опрацювання багатоканальних відеосигналів на основі моделі програмування OMPSs, а також зроблено тестування програми.

В четвертому розділі обґрунтовані техніко-економічні показники і ефективність виконаної розробки в порівнянні з аналогами.

Отже, всі задачі, що ставились на початку роботи вирішені, а мету роботи досягнуто.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Крупельницький Л. В. Комп’ютерна система моніторингу телевізійного мовлення / Л.В. Крупельницький, М.В. Жилін, В. В. Самко // Інформаційні технології та комп’ютерна інженерія. – 2019.

2. Сэломон Д. Сжатие данных, изображения и звука. М.: Техносфера, 2006. – 368 с.

3. Ричардсон Я. Видеокодирование H.264 и MPEG-4 – стандарты нового поколения. - М. – Техносфера. – 2005. – 368 с.

4.Тропченко А.Ю., Тропченко А.А. Методы сжатия изображений, аудиосигналов и видео: Учебное пособие – СПб.: СПбГУ ИТМО, 2009. – 108 с.

5. Ватолин Д., Ратушняк А., Смирнов М., Юкин В. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. - М.: ДИАЛОГ-МИФИ, 2002. - 384 с.

6. Тропченко А.Ю., Тропченко А.А. Методы сжатия изображений, аудиосигналов и видео: Учебное пособие – СПб.: СПбГУ ИТМО, 2009. – 108 с.

7. Цифровая обработка телевизионных и компьютерных изображений. Под ред. Зубарева Ю.Б. и Дворковича В.П. – М.: Международный центр научной и технической информации, 1997. – 487 с.

8. Гонсалес Р., Вудс Р. Цифровая обработка изображений. // Пер. с англ.- М.– Техносфера. – 2006. – 1072 с.

9. Wiegand T., Sullivan G. J. and others. Overview of the H.264\AVC Video Coding Standard. IEEE transactions on circuits and systems for video technology, vol.13, 2003, p.560-576.

10. Тропченко А.Ю., Курносенков И.Н. Анализ современных стандартов сжатия видеоданных// Научно-технический вестник СПбГУ ИТМО. – Вып. 32. – СПб. – СПбГУ ИТМО. – 2006. – с.17-21.

11. E. H. Land and J. J. McCann. Lightness and retinex theory. Journal of the Optical Society of America, 61(1):1-11, January 1971.

12. B. K. P. Horn. Determining lightness from an image. Computer Graphics and Image Processing, 3(4):277-299, December 1974.

13. H. G. Barrow and J. M. Tenenbaum. Recovering intrinsic scene characteristics from images. Academic Press, New York, NY, 1978.

14. A. Blake. Boundary conditions for lightness computation in mondrian world. Computer Vision, Graphics, and Image Processing, 32(3):314-327, 1985.

15. J.-M. Morel, A. B. Petro, and C. Sbert. A PDE formalization of retinex theory. Image Processing, IEEE Transactions on, 19(11):2825-2837, November 2010.

16. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. Numerical Recipes 3rd Edition: The Art of Scientic Computing. Cambridge University Press, New York, NY, 3 edition, 2007.

17. Ray Bernard. H.264 and I-frames, P-frames and B-frames [Електронний ресурс] / Ray Bernard // SIW. – 2020. – Режим доступу до ресурсу: https://www.securityinfowatch.com/video-surveillance/article/21124160/real-words-or-buzzwords-h264-and-iframes-pframes-and-bframes-part-2.

18. BSC. The OmpSs Programming Model [Електронний ресурс] / BSC // BSC Programming Models. – 2020. – Режим доступу до ресурсу: https://pm.bsc.es/ompss.