НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

«На правах рукопису»
УДК _004.056.5____

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Спосіб ентропійного кодування відео на базі використання розширених наборів SIMD інструкцій»

Виконав (-ла):
студент (-ка) VI курсу, групи ІМ-з11мп
Бойко Тимур Петрович _____

Науковий керівник:
доц., к.т.н.,
Русанова Ольга Веніамінівна _____

Консультант з нормоконтролю:
проф., д.т.н.,
Жабін Валерій Іванович _____

Рецензент:
доц. кафедри ІСТ, к.т.н.,
Шимкович Володимир Миколайович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2022 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій СТІРЕНКО

«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Бойко Тимуру Петровичу

1. Тема дисертації « Спосіб ентропійного кодування відео на базі використання розширених наборів SIMD інструкцій », науковий керівник дисертації доц., к.т.н., доц. Русанова Ольга Веніамінівна., затверджені наказом по університету «4097-с від 08.11.2022»

2. Термін подання студентом дисертації 12 грудня

3. Об’єкт дослідження програмні засоби ентропійного кодування відео

4. Вихідні дані ентропійне кодування відео

5. Перелік завдань, які потрібно розробити: дослідити архітектуру та апаратні реалізації сучасних процесорних систем; дослідити метод ентропійного кодування в AV1; вдосконалити програмну реалізації ентропіного кодеку у відеокодеку AV1 на базі розширеного набору інструкцій SIMD AVX-512.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: представлено у вигляді презентації

7. Орієнтовний перелік публікацій: Спосіб ентропійного кодування відео на базі розширеного набору інструкцій SIMD AVX-512 / Т.П.Бойко, О.В.Русанова// Проблеми інформатизації та управління-2022.-№2(70)-С.10-18. https://doi.org/10.18372/2073-4751.70.16841

8. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завданняприйняв
Нормоконтроль	*проф., д.т.н., Жабін В.І.*		

9. Дата видачі завдання _____1 вересня_____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1.	*Затвердження теми роботи*	1.09.2022	
2.	*Дослідження літератури та документації*	2.09.2022-21.09.2022	
3.	*Огляд існуючих рішень, аналіз теоретичних методів*	22.09.2022-4.10.2022	
4.	*Вдосконалення способу ентропійного кодування на базі SIMD, що зменшує час ентропійного кодування відео на процесорних мікроархітектурах із AVX-512*	5.10.2022-23.10.2022	
5.	*Проєктування програми*	24.10.2022-26.10.2022	
6.	*Програмна реалізація алгоритму та інтеграція у репозиторій AV1*	27.10.2022-19.11.2022	
7.	*Тести,бенчмаркінг, виправлення помилок*	20.11.2012-29.11.2022	
8.	*Передзахист*	30.11.2022	
9.	*Захист*	22.12.2022	

Студент Тимур Бойко

Науковий керівник Ольга Русанова

АНОТАЦІЯ

Магістерська дисертація присвячена пришвидшенню роботи програмного забезпечення: ентропійного кодування відео AV1 на базі SIMD. Представлена версія програмного забезпечення, що написана в ході роботи є вдосконаленням існуючого скалярного алгоритму ентропійного кодування AV1 для процесорних мікроархітектур із підтримкою SIMD AVX-512 –використовуючи паралелізм на рівні даних, написана на мові Ассемблер.

Дана версія ентропійного кодування дозволяє пришвидшити процес декодування відео AV1 під час перегляду або стрімінгу відео та задіяти усі можливості мікроархітектур на базі SIMD AVX-512 .

Ключові слова: ентропійне кодування, кодування відео, SIMD, AVX-512

ABSTRACT

The master's thesis is devoted to the acceleration of software: SIMD-based entropy encoding of AV1 video. The presented version of the software, written during the work, is an improvement of the existing AV1 scalar entropy coding algorithm for processor microarchitectures with AVX-512 SIMD hardware support - using parallelism at the data level, written in the Assembler language.

This version of entropy coding allows you to speed up the AV1 video decoding process while watching or streaming video and use all the capabilities of microarchitectures based on SIMD AVX-512.

Keywords: entropy coding, video coding, SIMD, AVX-512

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Спосіб ентропійного кодування відео на базі розширених наборів інструкцій SIMD

студентом: Бойко Тимуром Петровичем

Робота складається із вступу та трьох розділів. Загальний обсяг роботи: 142 аркуша тексту, 42 ілюстрації, 43 таблиці. При підготовці використовувалася література з 61 різного джерела.

Актуальність. Величезну популярність набув он-лайн перегляд відео-файлів на персональних комп'ютерах та інших портативних пристроях, особливо на майданчику Youtube. Youtube планує до 2025 року перекодувати більшість файлів у формат відео-кодеку AV1, який є найбільш вимогливим до процесорної системи із існуючих відеокодеків. Дедалі зростає якість відео та потужність процесорних систем користувача, додаються нові розширення їх архітектуру. Традиційні центральні процесори обмежені у своїй ефективності, тому й обмежена якість відео, яке може програватись на пристрої користувача із таким процесором. Альтернативою підвищення продуктивності програм у традиційних процесорних системах є використання розширення архітектури у вигляді аппаратного набору SIMD-команд.

Критичним місцем (hot spot) у кодуванні відео є частина вихідного коду відеокодеку, яка відповідає за ентропійне кодуваггя. В сучасних відеокодеках використовують бінарний арифметичний кодек для ентропійного стискання, проте його особливістю є погана здатність до будь-якого типу розпаралелювання. Для розв'язання цієї проблеми Mozilla розробила відеокодек Daala, який вже має програмні реалізації для процесорів із підтримками розширеного набору інструкцій SSE, AVX, AVX2. Проте, по-перше, їх можна покращити, по-другк, останні тренди свідчать про поширення процесорних систем із підтримкою розширеного набору інструкцій AVX-512, що надає ще більші можливості для

прискорення ентропійного кодування, що і потребує подальшого дослідження та розробки нових програмних реалізацій.

Мета і завдання дослідження. Метою магістерської роботи є зменшення часу кодування відео – ентропійного кодування з використанням можливостей процесорів із розширеним набором інструкцій типу AVX-512 шляхом удосконаленнявихідного коду на мові Assembler на базі SIMD типу AVX-512.

Для досягнення поставленої мети визначено та вирішено наступні завдання:

- дослідження архітектури та апаратної реалізації сучасних процесорних систем;
- класифікація та узагальнення архітектур комп'ютерних систем та принципів їх побудови;
- розробка та спеціалізація програмної реалізації ентропіного кодеку у відеокодеку AV1 для процесорів із підтримкою розширення набору інструкцій AVX-512;
- ілюстрація алгоритму роботи відеокодеку, ентропійного кодування як його складової та демонстрація бенчмарків отриманих результатів.

Об'єкт дослідження – процес ентропійного кодування – діапазонний мультисимвольний арифметичний кодер Daala відеокодеку AV1 та програмна реалізація його алгоритму.

Предмет дослідження – методи ентропійного кодування відео на базі мультипроцесорних систем на базі розширеного набору інструкцій SIMD

Методи досліджень. Для досягнення завдань, що були поставлені в магістерській дисертації, використано методи індукції, дедукції, імітаційного моделювання.

Наукова новизна результатів, що були отримані, полягає у:

- вперше додано вихідний програмний код на мові Assembler на базі SIMD для процесорів із підтримкою розширеного набору інструкці SIMD типу AVX-512;

- досліджено продуктивність коду на базі SIMD для процесорів із підтримкою розширеного набору інструкці SIMD типу AVX-512 у порівнянні із кодом для процесорів без SIMD-розширень, з розширеним набором інструкцій SIMD типу SSE (128 біт), типів AVX й AVX2 (256 біт) та досліджено продуктивність ентропійного кодування відео та завантаженість мікроархітектури.

Результати даного дослідження дозволють застосувати новий вихідний код на мові Assembler на базі SIMD AVX-512 із публічним репозиторієм для кодування відео Google AOM AV1 для побудову проекту та виконавчого файла для кодування відео на процесорних системах із підтримкою розширеного набору інструкцій SIMD типу AVX-512. Вихідний виконавчий файл дозволяє та зменшити час виконання кодування відео відеокодеком AV1, завантаженість мікроархітектури під час роботи відеокодеку AV1 (наприклад, звичайний перегляд відео на Youtube).

Особистий внесок здобувача. Магістерська дисертація виконувалась самостійно, є самостійною роботою що відображає особистий підхід автора та та прикладні результати щодо вирішення задачі спеціалізації та покращення продуктивності ентропійного кодування на базі SIMD для відеокодеку AV1. Визначення мети, завдань дисертації розглядалося з науковим керівником.

Практична цінність. Отримані результати можуть використовуватися у майбутніх дослідженнях за напрямками:

- вдосконалення та спеціалізація програмного коду алгоритмів ентропійного кодування для спеціальних процесорних систем;
- аналіз та дослідження змін у процесі виконання коду ентропійного кодування на спеціальній процесорній системі;

Публікації:

Спосіб ентропійного кодування відео на базі розширеного набору інструкцій SIMD AVX-512 / Т.П.Бойко, О.В.Русанова// Проблеми інформатизації та управління-2022.-№2(70)-С.10-18. https://doi.org/10.18372/2073-4751.70.16841

Ключові слова

Ентропійне кодування, адаптивне кодування, арифметичне кодування, CABAC, ентропійне кодування, відео-кодек, Daala, SIMD, AVX-512

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ І ПОЗНАЧЕНЬ

ЦП – центральний процесор

АЛП – арифметично-логічний пристрій (або ALU)

ILP – Instruction Level Parallelism (Паралелізм на рівні інструкцій)

GPU – Graphical Processing Unit (графічний обчислювальний пристрій)

SIMD – Single Instruction Multiple Data (Одинична інструкція – множинні данні)

AVX – Advanced Vector Extension (додаткові векторні розширення)

FMA – Fused Multiply-Add (Змішане (одночасне) множення-додавання)

Кодек – програма для КОдування / ДЕКодування інформації

ОС – операційна система

CISC – Complex Instruction Set

RISC – Reduced Instruction Set

IF – instruction fetch (завантаження інструкцію)

ID – instruction decode (декодування інструкції)

IE – instruction execute (виконання інструкції)

MEM – завантаження даних інструкцією (від англ. – memory)

WB – write back (зворотній запис)

ADD – додавання

SUB – віднімання

MUL – множення

DCT – дискретно-косинусне перетворення

IDCT – зворотнє (inverse) дискретно-косинусне перетворення

VLC – метод кодів змінної довжини

CABAC – метод адаптивного арифметичного кодування

CPI – (instructions per cycle) — кількість інструкцій за такт

ISA – (instruction set architecture) –архітектура системи команд

X86 – архітектура процесорів, стандарт набору команд, вперше розроблений Intel на базі мікропроцесора Intel 8086. Підтримується усіма процесорами Intel та AMD

ВСТУП

Починаючи з останнього десятиліття спостерігається швидке зростання інтересу до перегляду відео в інтернеті на різноманітних пристроях користувача, а також суттєве збільшення користувачів мережі Youtube та значного підвищення вподобань щодо якості – наприклад популяризація формату 4К. Тому серверні можливості зберігання відео користувачів у мережі інтернет будуть потребувати суттєвого розширення якщо не підвищувати рівень стиснення відео та потребують суттєвих затрат. Також користувач обмежений пропускної здатності інтернет-мереж.

Як інтернет-мережі клієнта, так і серверні сховища компаній відео-хостингу перебувають на межі, а тому все більше підіймається питання оптимізації та провадження нових алгоритмів для зростання якості відео, зниження завантаженості серверів компаній та мереж користувачів. Все це зробило необхідністю впровадження нового формату стиснення відео – AV1, що майже на 50% знижує бітовий потік стисненого відео за тієї ж якості. Такі компанії як Google, Mozilla, Cisco, Netflix, Qualcom, IBM організував пул компаній-дослдників, мета яких створення належної інфраструктури, вдосконалення та популяризації відео-кодеку AV1. Також за допомогою перекодовують вже існуючі відео-файли на своїх серверах у формат AV1, що потребує величезних обчислювальних потужностей, серверних процесорних систем дані компанії.

Найбільш проблемною частиною відео-кодеків, що має низькі або вістуні можливості для паралельної обробки, є ентропійне стиснення – що є фінальним узагальненим методом прибирання (стиснення) даних. Сучасні процесорні системи що існують на ринку зробили можливим написання використання програмних реалізацій відео-кодеків на рівні крупно-зернистого паралелізму, проте це не вплинуло на можливості розпаралелювання ентропійного стиснення. Опрацювання інтересу до даної проблему почалося із 2010-х років коли Я.Дуда та компанія Google запропонували діапазонно-мультисимвольну версію ентропійного кодеру, яку спочатку використали в експерементальному VP10,

потім як есперементальну у відеокодеку Daala, а вже потім ентропійний кодер Daala став частиною відеокодук AV1. В данному контексті його і буде розглянуто в даній роботі.

Одним із ключових рішень, що сприяв розв'язанню проблеми «ланцюжка» став перехід від бінарного до мультисимвольного арифметичного кодування. Дане рішення гірше масштабується на процесори без набору розширень SIMD-інструкцій, проте суттєво покращує продуктивність коду процесорних систем із набором розширень SIMD-інструкцій. І чим більше довжина векторного регістру таких систем, тим більше даних здатний обчислювати енкодер за одиницю часу.

Важливим фактором що блокує пропорційне масштабування ефективності ентропійного кодеру щодо довжини векторного регістру є незмінний розмір шини сучасних систем – 128 біт. Також, фактором, що блокує поширення ентропійного стиснення Daala із застосуванням є відносно велика (але така, що постійно скорочується частина старих пристроїв без підтримки розширення інструкцій набору SIMD, або із обмеженою підтримкою типу MMX, SSE2 тощо, розміром векторних регістрів 128 біт) та новизна та невелика частка пристроїв із підтримкою розширеного набору інструкцій SIMD (в т.ч типу AVX-512 – із розміром векторних регістрів 512 біт та нових інструкцій). На даний час код переважно оптимізований для мультипроцесорних систем із підтримкою розширеного набору інструкцій типу AVX2 (розміром векторних регістрів 256 біт).

Поведінка сучасної мультипроцесорної системи із підтримкою розширеного набору інструкцій SIMD (AVX-512) у процесі виконання коду ентропійного стиснення Daala є недостатнє дослідженим, оскільки фактори, що блокують її ефективність – сталий розмір шини процесорної системи у 128 біт, споживання енергії системою при застосування інструкцій над 512-бітними регістрами, конкуренція пристроїв із апаратною реалізацією кодеку AV1.

Майже усі сервені процесорні системи підтримують розширеного набору інструкцій SIMD, а останні покоління тип AVX-512. Поява доступних сучасних пристроїв (Intel Ice Lake, Tiger Lake) та відсутня спеціалізація програмної

реалізації для зазначеної мультипроцесорної системи із підтримкою нового розширеного набору інструкцій SIMD типу AVX-512 у процесі виконання коду ентропійного стиснення Daala. Це має бути не просто розширення довжини векторних регістрів SIMD, але суттєве використання нових інструкцій – оскільки тип AVX-512 має суттєво більший набір логічних команд у порівнянні із SSE/AVX2.

Тому, тема магістерської роботи, а саме дослідження та написання спеціалізованої програмної версії коду для для мультипроцесорної системи із підтримкою розширеного набору інструкцій SIMD (в т.ч. типу AVX-512) є надзвичайно актуальною. В існуючих дослідженнях, ці проблеми досліджуються щоб знизити кількість витраченої на кодування відео енергії, максимально ефективно використовувати інфраструктуру мережі та мати якісний стабільний відео-зв'язок та сприяти подальшому перегляду алгоритмів, які в деяких випадках є незмінними десятиліттями.

РОЗДІЛ 1
SIMD В СУЧАСНОМУ ЦЕНТРАЛЬНОМУ ПРОЦЕСОРІ

В Розділі 1 буде детально розглянуто мікроархітектуру сучасного центрального процесора в контексті продуктивності виконання програмного коду і оцінено нові сучасні його можливості – суперскалярність, суперконвеєрність, позачергове виконання (out-of-order execution). Буде розглянуто місце сучасного центрального процесору згідно класифікації Фліна та Фон-неймана.

У п.1.2 буде розглянуті гібридні архітектури процесорних систем та місце SIMD у поєднанні із класичними архітектурами.

У п.1.3 буде детально розглянуто можливості програмної й апаратної реалізації додаткових розширень у вигляді SIMD-розширень. Розглянуто синтаксис та можливості SIMD команд у контексті їх впливу на продуктивність коду.

1.1. Розширений набір інструкцій SIMD в мікроархітектурі сучасних ЦП

Мікроархітектура процесора це схема розташування та схема з'єднань регістрів, арифметично-логічного пристрою, кінцевих автоматів, банків пам'яті та інших функціональних юнітів, що необхідні для реалізації цієї архітектури [1].

Термін “мікроархітектура” в інформаці слід відрізняти від імплементації - апаратним дизайном, що імплементує архітектуру. Іноді під мікроархітектурою центрального процесора розуміють обчислювальну архітектуру плюс апаратну архітектуру процесора (на кремнії), (функціональну структуру застосовуваних технологічних рішень).

Хоча архітектура та реалізація — окремі поняття, але вони пересікаються - архітектура прямо впливає на якість імплементації.

Обчислювальна мікроархітектура — більш вужче поняття і представляє набір доступних регістрів, лічильник та набір команд. Це викликає необхідність підтримки розробниками версій ПЗ окремо для кожної з найбільш популярних архітектур. Це прямо впливає на скомпільований асемблерний та, як наслідок,

машинний код. Тобто, в залежності від мікроархітектури центрального процесора, для роботи на різних платформах, програміст має забезпечити декілька скомпільованих версій програми для кожної цільової архітектури. Наприклад, Microsoft, у звязку із тотальним поширенням RISC архітектури (процесорів ARMv8), у співробітництві із Qualcomm вирішила створити окрему версію ОС Windows 10 та Windows 11 для процесорів дизайну ARM [1].

Найбільш популярними архітектурами назарі є CISC, RISC, VLIW. Поділ процесорів на ті, які наслідують в чистому вигляді RISC, CISC або VLIW архітектури залишився в минулому, оскільки теперішні імплементації процесорів переймають рішення із усіх зазначених архітектур.

VLIW — це архітектура, де декілька інструкцій в одному блоці декодуються разом і виконуються паралельно, але без перевірки на наявність залежностей між ними: впорядкування інструкцій для забезпечення паралелізму на рівні інструкцій (ILP) здійснюється повністю компілятором. VLIW архітектура наслідує принцип MIMD (розглянуто нижче). VLIW архітектура покликана забезпечити простішу та дешевшу апартну імплементацію. Поки що імплементації архітектури VLIW для центрального процесора не попуялрні : все ще потребують високої кількості транзисторів, хоча компілятори для VLIW-процесорів вже не примітивні як в минулому. Однак в найближчому майбутньому щільність транзисторів та подальший розвиток методів компіляції для паралелізму на рівні інструкцій зроблять їх конкурентними до RISC та CISC архітектур.

CISC (complex instruction set) — це архітектура, в якій процесор має використовувати окремий мікрокод (мікропрограму) для виконання кожної окремої з обмеженого набору команд. Але мікропрограми мають відмінну довжину та потребують складної імлементації електронних ланцюгів (тразисторних функцій) для вибірки та виконання. З початку появи CISC-архітектури набір команд та їх скаладність постійно зростає, оскільки зростають можливості для їх імлементації. Але дослідження використання процесорів показало, що більше 80% часу виконуються менше 20% із всього сету команд. Тому в наш час імплементації CISC архітектури адаптовані до виконання 20%

найбільше використовуваних простих мікропрограм (команд). CISC до сих пір є найбільш популярною архітектурою персональних компютерів.

RISC (reduced instruction set) — архітектура, в якій процесор виконує команду за один такт. Всі команди виконуються апаратним забезпеченням без інтерпретації мікрокоду. Відсутність інтерпретації макрокоманд, що спрощують імплементацію процесора та підвищують швидкість його роботи. Імплементація архітектури RISC мають меншу площу на кристалі, що дозволяє мати більше регістрів — у сучасних імплементаціях більше 100 регістрів. Тому процесор на RISC архітектурі має на третину менше звернень до пам'яті, що підвищує швидкість виконання програми.

Сучасні програми містять велику кількість незалежних інструкцій, тому в сучасних імплементаціях CISC/RISC архітектур додано можливість незалежним інструкціям виконуватись одночасно, що називають паралелізмом на рівні інструкцій (ILP). ILP паралелізм досягається за допомогою апаратної імплементації техніки “позачергове” виконання (Out-of-Order execution, OoO), що дозволяє процесору динамічно змінювати порядок виконання інструкцій [2].

Суттєвим недоліком процесорів на архітектурі VLIW та перевагою CISC та RISC процесорів є те, що паралелізм на рівні інструкцій не під час компіляції, а під час виконання програми спеціальною апаратурою (Рис. 1.1).

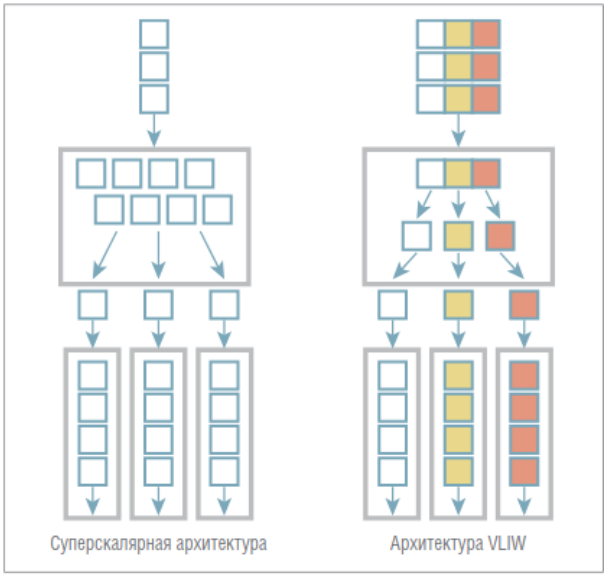

Рис. 1.1 Порівняння суперскалярних (RISC/CISC) та VLIW архітектур[3]

VLIW-процесори працюють краще на дуже регулярному коді адаптованому до довгих векторних обчислень, тоді як архітектури із апаратною реалізацію ILP - коли структура коду змінна: на частинах коду, що важко векторизувати [2].

Хоча VLIW-процесор і має коротший конвеєр (Рис. 1.1), але головною проблемою стає складний компілятор: розробка, підтримка, подальше вдосконалення, а також використання таких процесорів для масового сегменту, а в більшості випадків — для рядових задач — неможливість повного завантаження даними VLIW-архітектури [4].

Доречно зауважити, що терміни “VLIW” та “SIMD” не пов'язані між собою поняття, хоча їх досить часто помилково ототожнюють. Для уточнення цих понять, коротко нагадаємо класифікацію систем за Флінном [5].

Одиночний потік інструкцій одиночний потік даних (SISD) — система, яка не використовує паралелізм ні на рівні інструкцій, ні на рівні потоку даних. Один виконавчий блок декодує один потік команд з пам'яті і генерує відповідні сигнали для процесорного елемента для обчислення наступного елементу із потоку даних (одна операція за шаг). Прикладами системи SISD є класичні однопроцесорні системи до 2000-х років (аж до Pentium 4, до появи та Core2Duo).

Одиночний потік інструкцій декілька потоків даних (SIMD) — система, що одночасно застосовує одну інструкцію до декількох окремих потоків даних.
SIMD системи є синхронними програмними комп’ютерними системами дрібнозернистого паралелізму [6]. Виконавчий блок декодує один потік команд з памяті, але генерує відповідні сигнали для процесорного елемента для одночасної паралельної обробки різних вхідних даних. У 1972 Флін поділив SIMD системи на ті, що мають процесор масивів (паралельний процесорний блок має власну пам’ять і файловий регістр), конвеєрний процесор (кожен паралельний процесор зчитує дані з регістрового файлу, окремо обробляє фрагменти вхідних даних і пише результати назад в цей же регістровий файл) — саме сюди відносять NEON, SSE, AVX та асоціативний процесор (кожен паралельному блок незалежно вирішує на основі вхідних даних, виконувати чи ігнорувати сигнал обчислення) — так званий masked-SIMD.

Множинний потік інструкцій одиночний потік даних (MISD) — система, в якій процесорні елементи обробляють один і той же потік даних, але після кожного незалежного обчислення узгоджують результат обчислень. Це нестандартна система, яка використовується з метою гарантування відмовостійкості — наприклад, система управління польотом SpaceShuttle [8].

Множинний потік інструкцій, множинний потік даних (MIMD) — система, в якій наявно декілька незалежних процесорів, які одночасно виконують різні інструкції над різними даними. До MIMD архітектури відносяться багатоядерні суперскалярні процесори та системи розподілених обчислень.

Класичні CISC та RISC архітектури відносяться до SISD систем, проте їх багатоядерні суперскалярні нащадки, а також VLIW архітектура (хоча вони і значно відрізняються), за класифікацією Флінна відносяться до категорії MIMD систем, однак за більш пізньою класифікацією Дункана багатоядерні суперскалярні системи відносяться до гібридної MIMD-to-SIMD системи. Такий тип системи розріняють й Русанова О., Корочкін В. – гібридні програмні комп’ютерні системи, що є комбінацією різних архітектур – SIMD, MIMD, VLIW, конвеєру [6]. Схематично це представлено на рисунку 1.2.

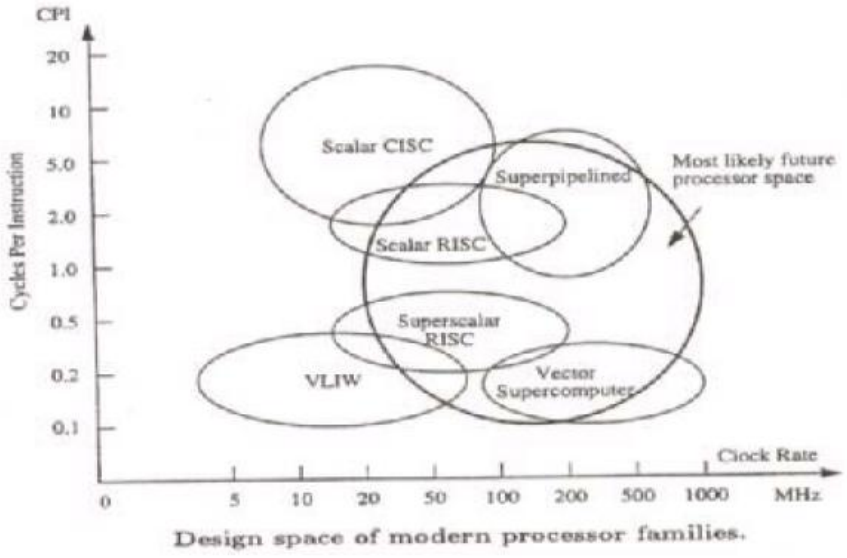

Рис 1.2 Сучасна архітектура центрального процесора [7]

Визначимо місце цільової системи згідно класифікації систем у Таблиці 1.1

Таблиця 1.1

Критерії класифікації процесорної системи

Критерій	Результат	Опис
Синхронність	асинхронна/ синхронна	Міжпроцесорна взаємодія – асинхронна, на рівні процессорного елемента – синхронна (кожний з процесорів має елементи векторного процессору)
Зернистість	крупно /дрібно	Міжпроцесорний паралелізм – крупний, на рівні підсистеми – дрібний (кожний з процесорів має елементи векторного процессору – синхронно виконує багато операцій за такт)
Тип памяті	роздільна (shared)	Усі процесорні елементи розділяють оперативну память, та кеш память третього рівня, при тому мають власний кеш 1 і 2
Тип звязаності	паралельна (сильна)	Процесорні елементи системи - ідентичні, та є конвеєрними процесорами
Типом управління	керована потоком команд	Суперскалярна система із динамічним конвеєром керована потоком із кешу інструкцій
Тип паралельної обробки	гібридна	гібридна MIMD-to-SIMD система.
Тип взаємозв'язку	статична	Зв'язок між процесорними елементами - через канали
Тип симетрії	симетрична	Процесорні елементи системи з однаковими правами доступу до пам'яті

Джерело: Корочкін О.В., Русанова О.В. Н72 Паралельні та розподілені обчислення[6]

В цій роботі розглянуто роботу програми та її оптимізацію до SIMD системи у вигляді існуючого розширення до ядра багатопроцесорної системи CISC архітектури - Intel SIMD Advanced Extension 512. Дункан Р. таку систему відносить до гібридних MIMD/SIMD систем, які виконують асинхронну обробку вхідних даних, як і MIMD-системи, але в цій MIMD-системі є процесорні елементи, що є підсистемою, яка працює синхронно у вигляді SIMD-системи Гібридні MIMD/SIMD системи еластичні, оскільки можуть адаптуватись до обробки даних конкретної прикладної задачі. Такі процесори ще називають потоковими процесори, і зараховують до них процесори з Intel Pentium III які пітдтримували технологію SSE (Streaming SIMD Extensions) [9, с.240, с.285].

SIMD в даному випадку є внутрішною частиною апаратної імплементації та безпосередньо доступний через архітектуру набору інструкцій (ISA) (рис. 1.3).

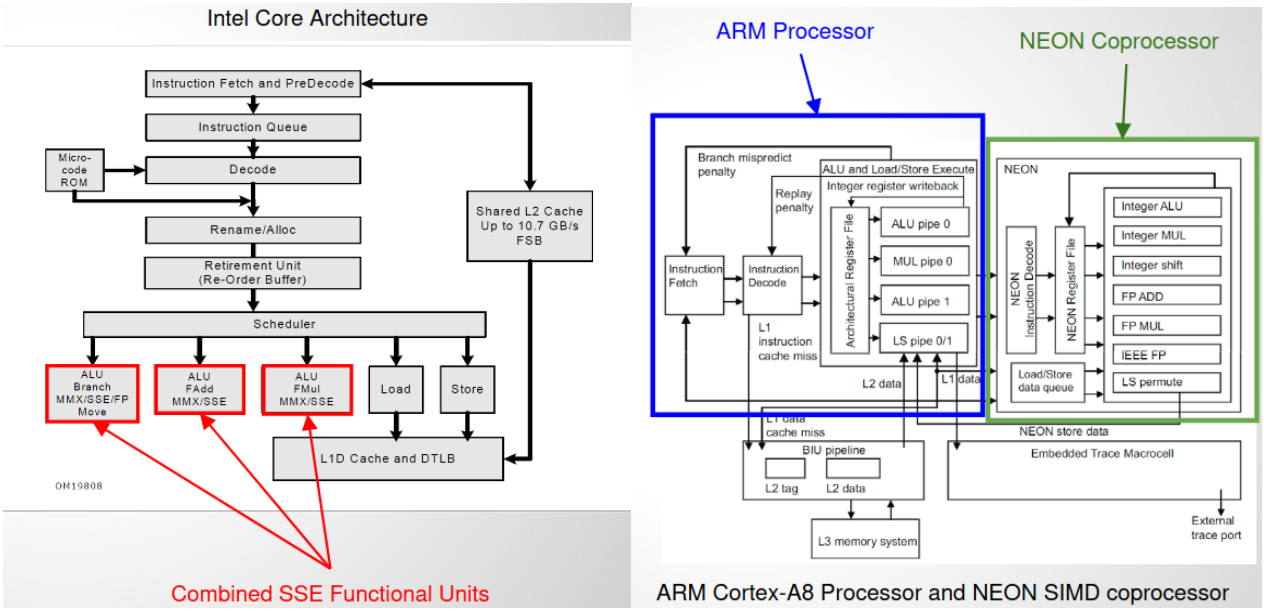

Рис. 1.3 Сучасні апаратні реалізації процесорної системи на базі архітектури CISC та RISC [23]

Intel та ARM додали до класичних архітектур CISC та RISC розширення SIMD як розширення ISA в якості опції. Однак для забезпечення принципів роботи RISC архітектури ARM додав розширення NEON у вигляді співпроцесора, який відповідав за обробку векторних інструкцій (Рис. 1.3). У Intel спочатку було додано апаратне забезпечення у виді співпроцесора для роботи з числами з плаваючою комою, до відповідного процесорами x86 (маркування чіпів з таким розширенням закінчувалось на «87»), колись було опціональним у версії x86 (AMD64). Потім до x86_64 було додано і стала обовязковою підтримка SSE2. Процесори архітектури «RISC» пройшли той же шлях - векторні інструкції NEON спочатку були опціональним розширенням процесорів ARM, але в невдовзі став обовязовим до набору інструкцій ARMv8-A. Ці інструкції SIMD займають кілька циклів і декодуються в декілька мікроінструкцій навіть на процесорах ARM (Рис.1.3).

У імплементації Intel Skylake планувальник має буфер на 97 інструкцій, які відправляє на 7 виконавчих блоків, що прив’язані до портів 0-7: Це три АЛП для роботи із цілими числами, два АЛП для роботи із дійсними числами та векторними командами та окремі блоки для збереження й завантаження. Буфер

дозволяє планувальнику динамічного аналізувати та міняти існуючий порядок потоку команд для максимально можливого завантаження усіх АЛП (порт0 – порт7) [10]. Для повного завантаження таож використовується і передбачувач розгалужень – апаратний модуль процесора, який намагається вгадати, яким шляхом піде розгалудження (наприклад, if–else) [11]. Ця техніка називається - спекулятивне виконання – коли ще до остаточного відома виконується найбільш вірогідна гілка. Якщо ж пізніше стане відомо, що передбачення було помилковим, тоді спекулятивно виконані або частково виконані інструкції відкидаються, і конвеєр починає спочатку з правильного відгалуження, що зазнає затримки. Час, який витрачається даремно в разі помилкового прогнозування гілок, дорівнює кількості етапів у конвеєрі від етапу отримання до етапу виконання – у сучасних мікропроцесорах це 20 тактів. Тому, для довгих конвеєрів потрібні більш складні передбачувачі розгалужень [12].

В сучасних процесорах інструкції SSE/AVX2/AVX512 виконуються атомарно на тих же АЛП, що й операції над числами з плаваючою точкою. У ранніх процесорах із SSE (наприклад, початок 2000-х років - Intel Pentium III) декодер інструкцій конвертував 128-розрядні інструкції у дві 64-бітних інструкції, що потім виконувались паралельно (рис.1.4) [20, с. 439].

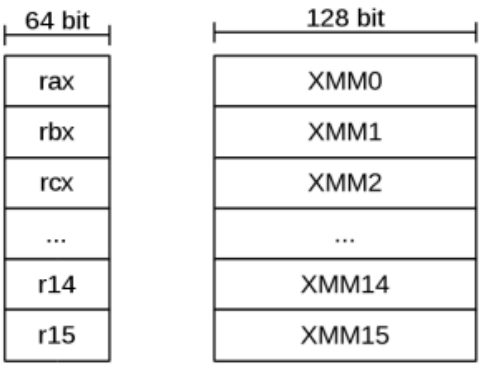

Рис.1.4 Регістри загального призначення та векторні регістри згідно Intel x86 calling conventions [20]

До того ж, у сучасних процессорах (Intel Skylake) для довгих та часто зустрічаємих команд виділено окремі процесорні одиниці (PU - processing units), що забезпечує високий рівень паралелізму на рівні інструкцій та максимально

використовує АЛП у рядових користувальницьких програмах. Наприклад, на рисунку 1.5 показаний набір процесорних одиниць мікроархітектури Intel Core 2 (2006 р.) , що мав окремі одиниці для обробки SSE-команд перемішування (1 од.), цілісного множення/ділення (1 од.), дійсних чисел (1 од.), зберігання та завантаження у пам'ять (3 од.), загального АЛП для цілісних чисел (1 од.)

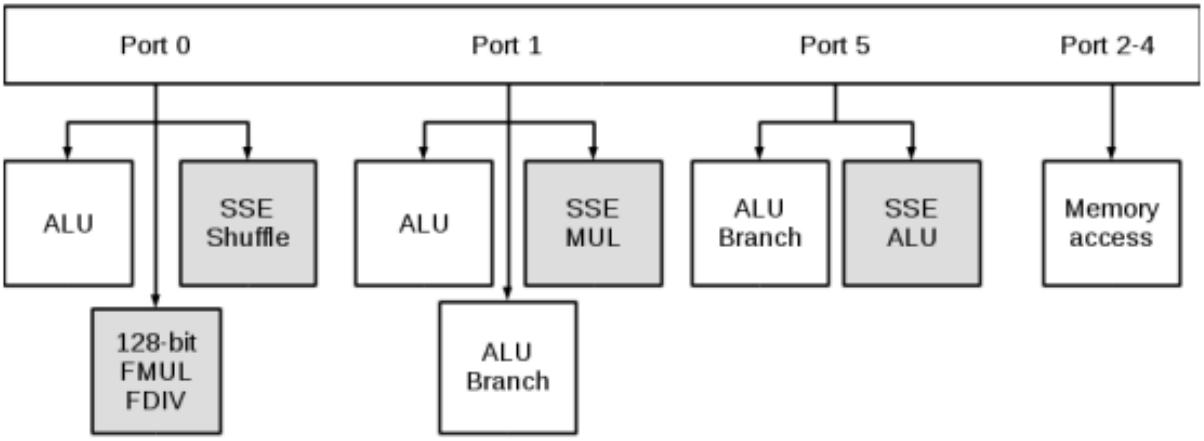

Рис. 1.5 Виконавчийпристрій Intel Core 2*[1] [10]

У кожному процесорі такої системи є одночасно як АЛП, що виконує дії над операндами послідовно (порозрядно), так і АЛП, що виконує дії над розрядами паралельно (одночасно), причому. По виду чисел, що обробляються, АЛПздатні виконувати операції над двійковими, двійково-десятковими, числами з крапкою, що плаває та фіксованою крапкою. У випадку двійково-десяткових кожндясяткова цифра записуєтьсячотирма двійковими розрядами. Кожне АЛП складається із суматора відповідного типу (паралельного, послідовного), блока управління обчисленням та регістрами [9, с.106 - 113]. АЛП, що здійснює паралельні обчислення SIMD працює на виконавчій одиниці SIMD, що приєднана до портів 0,1,5, що містять 4 паралельні суматори одинарної точності. Також, виконавча одиниця є конвеєрним - 4 затримки і 1/такт.

[1]* *ALU — з англ. АЛП скалярний арифметичний пристрій, FMUL/FDIV — SSE пристрій множення та ділення чисел з плавачою крапкою, SSE MUL — SSE ціле чисельне множення, SSE ALU — SSE пристрій простих арифметичних цілочисельних операцій [10].*

Конвеєр представляє собою окремий блок процесора - набір схем окремого обладнання, що організовані у конвеєр. Кожний ступінь конвеєру виконує одну частину (IF/ID/IE/MEM/WB) окремої команди одночасно. На межі кожного такту, результати одного етапу передаються до наступного. На рисунку 1.6 показано схему суперскалярного процесора, який приймає та виконує дві інструкції (2-way) одночасно (сучасні процесори зазвичай 4-way). Декодер через внутрішню шину даних отримує 2 інструкції із кешу інструкції за раз. Конвеєр має регістровий файл, з'єднаний із шістю портами для зчитування 4 операндів на вхід і зворотнього запису 2 операндів за 1 цикл. Конвеєр містить 2 АЛП і пам'ять із 2 портами прийому даних, тобто може виконувати 2 інструкції одночасно (Рис. 1.6).

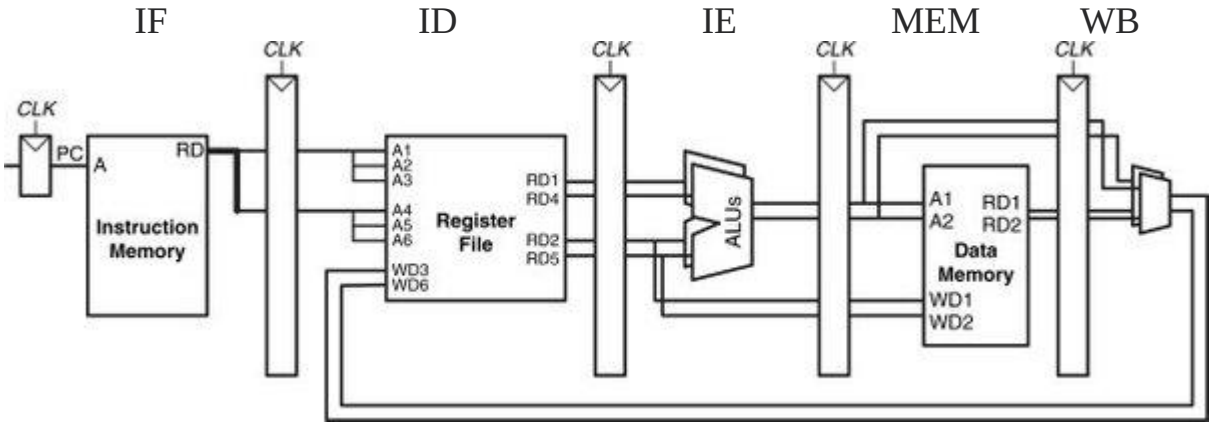

Рис. 1.6 Схема суперскалярного суперконвеєру [13, с. 439]

В Процесорі Intel 486 було імплементовано 5-етапний (ступеневий) конвеєр інструкцій. На етапі вибірки інструкцій (IF) одиниця попередньої вибірки інструкцій заповнює 32-байтну чергу попередньої вибірки інструкцій блоками інструкцій по 16 байт, звідки вибирається конкретна інструкція. Закодовані команди, так й мікроінструкції обробляються в два етапі на етапі декодування (ID). На етапі виконання (IE) виконуються як операції АЛП (арифметичні), так і операції доступу до кеш-пам'яті. Визначення ефективної адреси – на етапі декодування, а доступ до памяті – на етапі виконання, тому повторне використання результатів інструкції можливо в тому ж такті. Такий час (у циклах), доки результат команди стане доступним для використання як операнд у наступній інструкції називається – затримкою операції (latency). Проте якщо

наступна інструкція буде використовувати регістр з результатом попередньої – потрібен додатковий новий цикл (оскільки ефективна адреси визначається на етапі декодування ID). На останньому етапі конвеєра (WB) здійснюється запис назад в регістр.

5 ступеневей конвеєр інструкцій здатний виконати 5 інструкцій IA32 за один цикл, хоча довгі інструкції потребують декодування на мікроінструкцій і потребують 2 і більше циклів. Процессор Intel 486 витрачав приблизно 1,95 циклів на інструкцію (CPI) [14],що було в 2.5 рази менше ніж в Intel 386 [15].

Сучасні процесори Intel – суперскалярні (4-way), із динамічним суперконвеєром довжиною 14-19: використовують усі вище вказані техніки - позачергове виконання інструкцій, спекулятивне виконання, перейменування регістрів. В суперскалярних суперконвеєрних процесорів з параметрами (IPC = m, op latency = n) час циклу складає 1/m порівнянно до скалярного конвеєра, і виконують n інструкцій за цикл (Рис. 1.7). В такому випадку говорять, що ILP необхідний для повної утилізації архітектури складає n*m інструкцій [14, с .189].

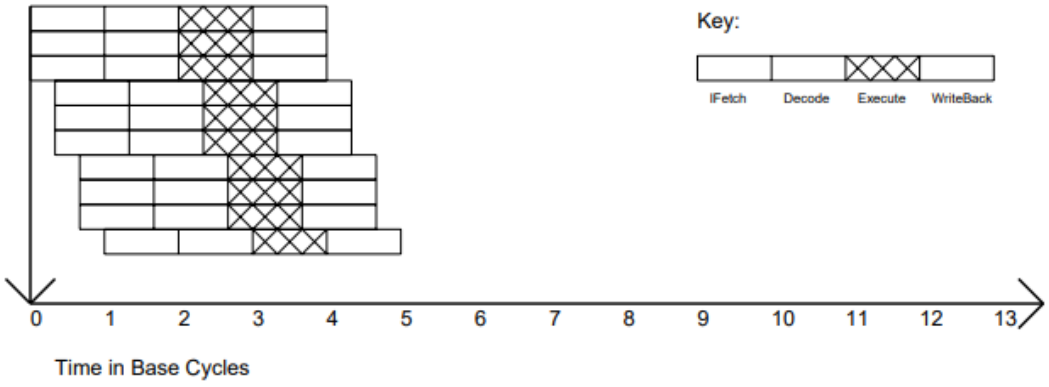

Рис. 1.7 Суперскалярний суперконвеєр (3, 3) [14]

Суперскалярний конвеєр шириною m може завантажити m інструкції з кешу інструкцій за цикл. Суперскалярний конвеєр на відміну від скалярних конвеєрів має вхідний буфер, що завантажує m інструкцій одночасно для динамічної (на льоту) буферизації інструкцій. Для мінімізації затримки інструкцій у суперскалярному конвеєрі інструкції можуть обходити призупинену провідну

інструкцію, що змінює порядок виконання інструкцій, як тільки їх операнди стають доступними, такий конвеєр, із можливістю позачергового планування інструкцій є динамічним конвеєром. В динамічниому конвеєрі позачергове виконання імплементовано через буфер інструкцій на декілька входів/виходів, через який інструкції можуть входити/виходити у оптимальному порядку (На Рисунку 1.8 – це Dispatch buffer).

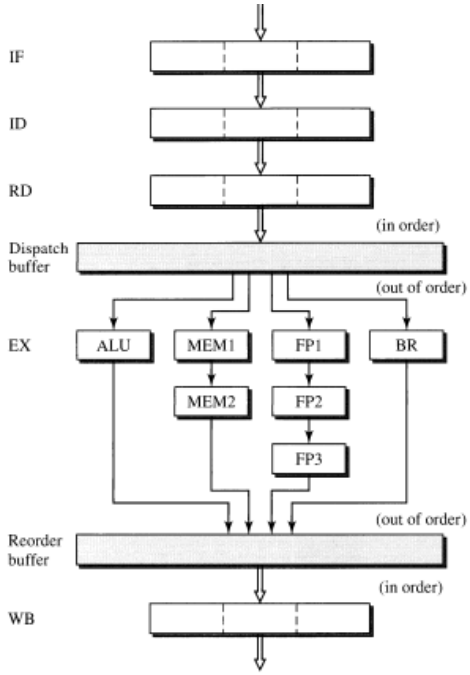

Рис. 1.8 Динамічний конвеєр із довжиною 3 елементи Intel P6*[2] [14]

Для архітектури CISC має місце навантаження на декодер інструкцій суперскалярного конвеєра, оскільки декодер ще й транслює інструкції у мікрооперації (низькорівневі RISC-подібні примітиви, μops) для можливості виконання на АЛП [16] у їхній роботі про високопродуктивневіднімання (HPS). За даними Intel, в середньому одна інструкція IA32 конвертується 1,5-2,0 μопсів (Intel P6). Схематично одиниці вибірки та декодування показані на рисунку 1.9.

[2] *де IF (Instructionfetch) -отримує інструкції з 64-байтової префетченої черги, ID-1 (instructiondecode) - переводить інструкцію в адресу мікрокоду, ініціює генерацію адреси та доступ до пам'яті. ID-2 Звертається до пам'яті мікрокодів, переводить μops для блоку виконання, EX (execution) - виконання операцій АЛП та операцій доступу пам'яті. WB (Write-back) - записує результату назад в регістр[14, с.189].*

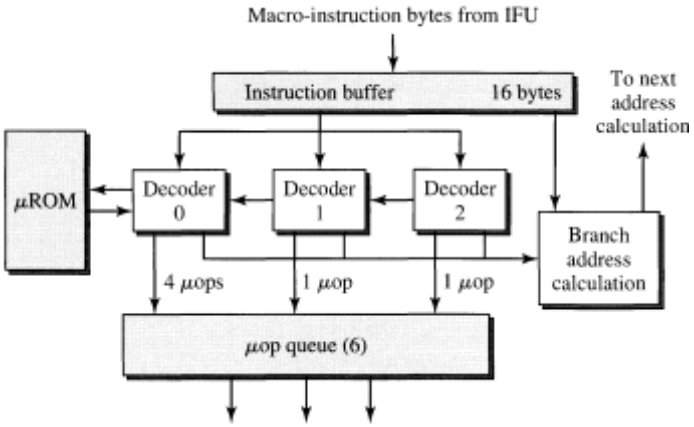

Рис. 1.9 Пристрої вибірки та декодування суперскалярного конвеєра моделі Intel P6 [14]

У кожному машинному циклі з буферу інструкцій відправляється 16 байтів у чергу інструкцій, а декілька паралельні декодерів (0-2) паралельно декодують ці байти із черги інструкцій. Декодер 0 може декодувати усі типи інструкції, тоді як декодери 1-2 можуть декодувати лише прості інструкції Починаючи з архітектури Haswell введена повна конвеєризація AVX інструкцій (з розкладом на μops).

Для повної утилізації можливостей мікроархітектури в сучасних процесорах імплементовано апаратну одиницю «перейменування регістрів (RFR)» — це частина конвеєра для розв’язання залежностей даних поміж інструкцій, що перейменовує визначене компілятором ім’я регістру (16 є явними в архітектурі набору інструкцій) на нове із фізично апаратно доступних 168, що й усуває та розпізнає справжні залежності між інструкціями [17]. Це не порушує конвенції ISA, тому апаратура динамічно змінює програмні номери регістри на фізичні через PRF [18, с. 40].

Архітектура x86_64 забезпечує 16 128-бітних регістрів, які доступні для інструкцій SSE. Рисунок 2.2 порівнює їх із звичайними георадарами. Однак слід зазначити, що незважаючи на те, що на рівні збірки видно лише 16 регістрів, набагато більше регістрів (наприклад, 168 регістрів SSE-AVX в Intel Haswell) реалізовано фізично та може використовуватися для перейменування. Зокрема, починаючи з мікроархітектури Intel NetBurst, значення SSE зберігаються в

окремому файлі регістру, який використовується спільно між SSE та операціями з плаваючою комою.

Одиниця виконання відправляє на відповідний порт АЛП для обчислення за кодом операції (мікрооперації), які виконує над вхідними операндами у регістрах-пересилання, та поміщує результат в регістр-приймач.

За типом дії над вхідними операндами: Такий тип ALU відноситься до блочних ALU, різні АЛП виконують обчислення над різними типами даних, а за типом зв'язку – безпосередньо пов'язаними регістра-пересилання та регістра-приймача.

Починаючи з архітектури Skylake (2017) AVX2 тепер має регульоване живлення до цього AVX2 не мали регулювання живлення AVX2, попередні процесори з AVX2 були схильні до витоку енергії. Intel отримав вмикнати/вимикати AVX2, коли процесорний елемент виконує код без AVX2. У архітектурі Skylake AVX2 блок має «розігрітись» щоб інструкції могли виконуватися на максимальній швидкості (10 000 циклів) – або за допомогою лише одного виконання однієї інструкції AVX2 за 10000 циклів до реального використання AVX2.

1.2. Метрики виміру продуктивності коду на мові Assembler на базі SIMD

Продуктивність системи залежить від багатьох факторів, які було зазначено у Розділі 1 : тактова частота, кількість циклів на інструкцію, кількість АЛП та багато інших. Але не один з цих факторів не є визначальним. Єдиний обєктивний спосіб виміряти продуктивність системи — порівняти час виконання необхідної програми на цій системи.

Час виконання програми за обчислюється за формулою 1.1:

$$Execution\ Time = (k_{інструкцій})*(k_{циклів}/інструкцію)*(k_{секунд}/цикл) \quad (1.1)$$

Кількість секунд на цикл (такт) – це час T_c тактового сигнала (між імпульсами), який залежить від частоти процесору - обернено пропорційний його тактовій частоті.

CPI - (cycles per instructions) – середня кількість тактів, що необхідні процесору для виконання інструкції. CPI обернено пропрційно показнику IPC – (instructions per cycle) — кількість інструкцій за такт (цикл). Зазвичай, вираховується із припущенням — без врахування роботи з підсистемою памяті [5, с. 938]

Як було зазначено у п. 1.1 сучасний процесор наявна можливість векторизації — можливість паралельно із скалярними АЛП застосовувати й векторні АЛП (по моделі SIMD — одинарна команда — множинні дані) , тобто одна команда виконується на даними усіх операндів регістрів, до яких вона застосована. Майже усі сучасні процесори Intel мають підтримку SSE (ширина регістру - 128 біт) та AVX-2 (256 біт), останні покоління — AVX-512 (512 біт), що, наприклад, дозволяє процесору одночасно виконувати обчислення над 64 одиниць по 8 біт або 8 одиниць по 64 біт одночасно.

При вимірюванні продуктивності векторизованого коду використовують такі метрики вимірювання ефективності коду, враховуючи особливості сучасних процесорів:

- затримка (latency) виконання векторних операцій відносно скалярних;

- пропускна можливість (throughput) - скільки векторних інструкцій можуть виконатись одночасно.

Затримка (latency) – час, необхідний системі, для обробки одиниці інформації від моменту її входження до моменту виходу із системи. *Затримкою операції* є кількість тактів, необхідних процесору від моменту надходження команди до АЛП до моменту завершення обчислень (коли результат виконання буде доступний для використання іншою командою — в регістрі).

Пропускна здатність системи (throughput) – кількість одиниць інформації, що може обробити система за одиницю часу [19]. Деякі автори ([20],[21]) визначають поняття “пропускна здатність” як пропускна здатність інструкції за цикл, деякі ([22], [23], [24]) - як пропускну здатність циклів за інструкцію. Тому Intel визначає пропускну здатність як кількість тактів очікування процесора, доки порт, що виконує цю інструкцію, буде вільним для її повторного прийняття, а

Фогнер А. як очікувану кількість тактів на інструкцію для серії таких самих незалежних інструкцій в межах даного потоку (взаємна здатність).

У даній роботі використоно тлучачення пропускною здатності, що вимірюється в циклах за інструкцію (reciprocal throughput).

Пропускна здатність підвищується можливістю обробки декількох одиниць інформації паралельно — паралелізмом. Паралелізм буває просторовим (використовується N>1 копій апаратних одиниць для одночасного виконання N завдань) та часовим (поділ команд на N ступенів для конвеєрної обробки).

Наприклад, у системі із N копій АЛП з паралелізмом на рівні інструкцій пропускна здатність дорівнює N/L. У системі із динамічним конвеєром із N ступенів, але із однієї копією АЛП пропускна здатність теоретично також може бути рівна N/L, але тільки в тому випадку, якщо всі операції конвеєра рівні L, інакше — N/L_{max}. Як було зазначено у п.1.1 команди в сучаних процесорах архітектури CISC мають 2 стадії вибірки інструкції (на другій робиваються на мікрооперації), тому можна казати що $N/L_{max} \sim N/L$ (мова йде про незалежні одна від одної команди).

Пропускною здатністю є кількість векторних інструкцій, що можуть здійснюватись системою паралельно — дорівнює кількості векторних АЛП в одиниці виконання. Цей показник часто використовують не стосовно системи, а стосовно інструкції (у системі із N=3, L=3, наприклад, пропускна здатність команди буде 0.33 — означає система може одночасно обчислити три команди векторного додавання vpadd).

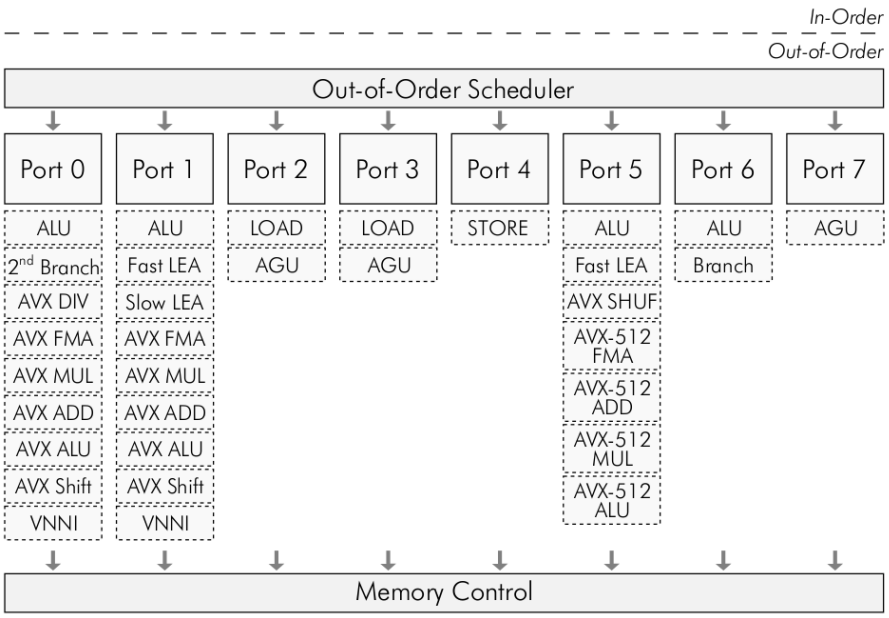

Рис. 1.10 Схема організації апаратних логічних пристроїв Intel [23]

Кожна інструкція (або серія мікрооперацій) може бути виконана лише на релевантному їй порті. Для представлення інструкції можна використати запис: 3*p_abc+1*p_de, що означає, що команда складається із 4 мікрооперацій μops, 3 з яких можуть виконатись на портах a,b, c, а 1 — на портах d, e.

Тиск на порти

Тиск на порти - це відносна кількість тактів де програма зупинялась не із-за обчислень чи звернень до ядра ОС Причинами такої призупинки є взаємозалежності між інструкціями програми, чи виконання великої послідовності інструкцій певною команди. Однією із технік, що зменшує тиск на порти є векторизація циклів - скорочує послідовності інструкцій за рахунок SIMD (у сучасних компілятор виконується автоматично)[24].

Нехай P — множина портів процесору, а U — множина мікроінстукцій команди I. Нехай ports : U → 2^P — таке відображення, що ports(u) — множина портів із функціональною одиницею FU, що здатна до виконання мікрооперації u. Завантаження портів для атомарно одиничної інструкції μ можна виразити за допомогою запису $P_{usage}(p_{combination}) = \{\mu \in U \mid ports(u) = p_{combination}\}$, тобто $p_u(pc)$

позначає кількість мікрооперацій інструкції I, чиї функціональні одиниці знаходяться в портах в pc [26].

Профайлер операційної системи Linux – программа perf за допомогою використання performance hardware counters процесору надає можливість аналізу таких показників тиску на порти та ефективного використання портів за допомогою лічильника процесора «UOPS_EXECUTED.CORE» (далі :

$$Cycles\_\{N\}\_Port\_Utilized = \sum_{\mu ops\ executed\ on\ \{Pn\}} / \sum_{cycles} \quad (1.2)$$

$$Cycles\_All\_Ports\_Utilized = \sum_{\mu ops\ executed\ on\ \{P0...PN\}} / \sum_{cycles\ x\ N} \quad (1.3)$$

$$Кількість\ «вкрадених»\ циклів = N*\sum_{cycles} - \sum_{\mu ops\ executed}$$

Джерело: складено автором на основі [60]

Загальну ефективність використання портів певною функцією обчислюється як відношення загальної суми тактів функції до суми добутків затримки кожної інструкції на її відносну пропускну здатність, тобто фактичної до теоретично можливої пропускної здатності.

Відносна кількість кеш-промахів (% тактів)

Запити до даних, що відсутні у кеш-пам’яті, перенаправляються до наступного рівня в ієрархії памяті (з L1 до L2, з L2 до L3, з L3 до глобальної памяті DRAM із дуже великою затримкою). Показник кеш-промахів вказує частку тактів з кеш-промахами до числа усіх тактів, що належали програмі. Найбільш великими є час очікування на виконання промахів до останнього рівня кешу L3 - такі такти втрачаються на очікування запиту і завантаження даних із глобальної памяті. Зменшення кеш-промахів можливе шляхом зменшення набору даних, використанні апаратної та програмної попередньої вибірки (prefetch), або переписування коду без блокуючих інструкцій.

Блокуючі інструкції

Інструкція, що виконається раніше за наступні інструкції (тобто блокує їх виконання). До блокуючи не відносяться інструкції операційної системи, інструкції з затримкою нуль, інструкцію зупинки та інструкції, що впливають на потік керування на основі значення регістра. Блокуючими є також інструкції де операндами є регістри різних типів (наприклад між векторним та загального

призначення), різної довжини (між XMM та YMM, ZMM), а також інструкції між регістрами та памятю.

Піковий рівень завантаження та тактова частота системи

Зауважимо, при використанні усіх портів із векторними АЛП (0, 1, 5) одночасно призводить до зниження тактової частоти процесора та, як наслідок продуктивності роботи програми, що може досягати до 30% в залежності від моделі процесора.

Вплив частоти залежить від ширини операндів команди та типу команди. Intel розрізняє 3 рівні тактової частоти процесора (ліцензії) на яких можуть обчислюватись певні команди: L0 (найбільш швидка), L1 і L2 (найбільш повільна). Ліцензія L0 — базова заявлена тактова частота процесора на якій здійснюється обмежений перелік команд із скалярними або короткими векторними операндами (128 біт – SSE команди). L1 – нижча тактова частота ніж у L0 – частота на якій процесор виконує інструкції набору AVX (256 біт) та AVX2 (256 біт). L2 є ліцензією на якій здійснюються команди набору AVX-512 із задіяними найдовшими операндами 512 біт. Точні тактові частоти процесора для кожної ліцензії ще потрібно корегувати на кількість процесорів системи, що задіяні одночасно для виконання одного типу ліцензії. Наприклад, для центрального процесора Intel Xeon 5100 предствлено на Рис. 1.11:

Mode	Base	Turbo Frequency/Active Cores													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Normal	2,200 MHz	3,200 MHz	3,200 MHz	3,000 MHz	3,000 MHz	2,900 MHz	2,900 MHz	2,900 MHz	2,900 MHz	2,700 MHz	2,700 MHz	2,700 MHz	2,700 MHz	2,600 MHz	2,600 MHz
AVX2	1,800 MHz	3,100 MHz	3,100 MHz	2,900 MHz	2,900 MHz	2,700 MHz	2,700 MHz	2,700 MHz	2,700 MHz	2,300 MHz	2,300 MHz	2,300 MHz	2,300 MHz	2,200 MHz	2,200 MHz
AVX512	1,200 MHz	2,900 MHz	2,900 MHz	2,500 MHz	2,500 MHz	1,900 MHz	1,900 MHz	1,900 MHz	1,900 MHz	1,600 MHz	1,600 MHz	1,600 MHz	1,600 MHz	1,600 MHz	1,600 MHz

Рис.1.11 Тактова частота в залежності від кількості активних ядер процесорної системи і типу інструкцій, що використовуються [25]

Відносне зниження тактової частоти для ліцензій L1 і L2 збільшується в залежності від числа процесорів, що виконують команди таких ліцензій. Якщо на 1-2 активних процесорах зниження тактової частоти для виконання ліцензій L1 складає 3% та 10%, то для трьох процесорів – вже 7%, та 20%, від L0, то для 14

активних процесорів зниження досить суттєве – 15% і 38%. Очевидно, що частота виконання команд «найвищої» ліцензії буде тактовою частотою для усіх команд «нижчих» ліцензій.

Інструкції, що активують ту чи іншу ліцензію залежить від ширини їх операндів та «важкості» - апаратної імплементації обчислення такої команди.

Width Light Heavy

Scalar L0 N/A

128-bit L0 L0

256-bit L0 L1*

512-bit L1 L2*

*soft transition (see below)

До важких інструкцій належать SIMD-інструкції, що виконуються на АЛП для дійсних чисел (FP/FMA) – з плавуючою крапкою (одинарної та подвійної точності) - ті, що виконуються на АЛП для множення, на АЛП для ділення, а також на АЛП для обчислення аккумулятивних інструкцій (множення із накопиченням, тощо). Всі інші SIMD-інструкції належать до легких інструкцій .

Переходи

Деякі інструкції мають м’який перехід, деякі - жорсткий перехід. Жорсткий перехід викликає зупинку ЦП та перехід новий частотний режим майже відразу (декілька тактів). М'який перехід наступає якщо важкі інструкції виконуються із неповною пропускною здатністю (відповідні порти не завантажені). Але по мірі зростання тиску на порту (росту пропускної здатності) понад дозволений процесором поріг, наступає перехід до наступної ліцензії. Тобто центральний процесор здійснює поступове зменшення тактової частоти в залежності від рівня тиску на порти (пропускної здатності) в процесі виконання важких інструкцій.

1.3. AVX-512 розширеного набору інструкцій SIMD: огляд

Набори SIMD-інструкцій є невід’ємною складовою функціональності сучасного центрального процесору. SIMD- інструкції надають можливість

швидших обчислень, якщо одну й ту ж інструкцію (або всю чергу інструкцій) можна застосувати до декількох елементів даних (векторів) одночасно. Набір інструкцій підтримується апаратно спеціально відведених у центральному процесорі АЛП, які працюють на окремих портах одиниці виконання (EU) на стадії виконання (IE). Усі сучасні процесори підтримують операції над векторами розміром 128 біт (SSE), більш просунуті – 256 біт (AVX2), серверні та деякі спеціалізовані процесори – 512 біт (AVX-512), що дозволяє обчислювати до 16 чисел з плаваючою крапкою (типу float32_t) або 64 8-бітні цілі (типу uchar) одночасно. Причому кожний новий тип розширення включає підтримку попереднього. Сучасні SIMD-розширення, окрім додавання, множення та інших базових арифметичних операцій над векторами, підтримують великий набір інструкцій: починаючи від перестановок елементів даних (смуги або lane) у векторному регістрі аж до тернарних операторів запрограмованих розробником. AVX-512 – не тільки про збільшення кількості розрядів одночасної обробки до 512. AVX-512 додав нові набори інструкцій та можливості оперувати над новими наборами даних (зсув над 128-бітним числом, тощо).

Інструкції наборів AVX-512BW, AVX-512DQ підтримують деякі нові інструкції для (8- та 16-розрядних та 32- та 64-розрядних цілих, скорочення від англ. byte, word (BW) double, quadword (DQ). Інструкції набору AVX-512VL підтримують довжини вектора 128 біт та 256 біт для більш інструкцій над довшими операндами у 512 розрядів (variable length). Інструкції набору AVX-512PF підтримують попередню вибірку даних для інструкцій scatter та gather (розглянуто нижче). Інструкції набору AVX-512ER використовують додаткову логіку для обчислень експоненти з базою двійка, оберненого квадратного кореня, степеню тощо. Всі вказані у даному абзаці набори інструкцій не потребують використання у даному дослідженні, та доступні лише у серверних процесорах. Тому Всі вказані у даному абзаці набори інструкцій не будуть розглянуті.

AVX-512F (від англ. F – «fundamental») набір інструкцій, що в основному розширив 256-розрядні AVX інструкції для роботи з 512 розрядними регістрами, та додав маскування, округлення, обробку винятків. AVX-512F є обов’язковим для

усіх імплементацій архітектури, що пітримують регістри довжиную 512 біт. У розширенні AVX-512F представлено найбільш широкий набір SIMD-інструкцій, який і буде розглянуто у цьому пункті.

Розвиток розширених наборів SIMD-інструкцій в мікроархітектурі Intel від MMX (2000р.) до AVX2 (2015р.) показано на рис 1.12.

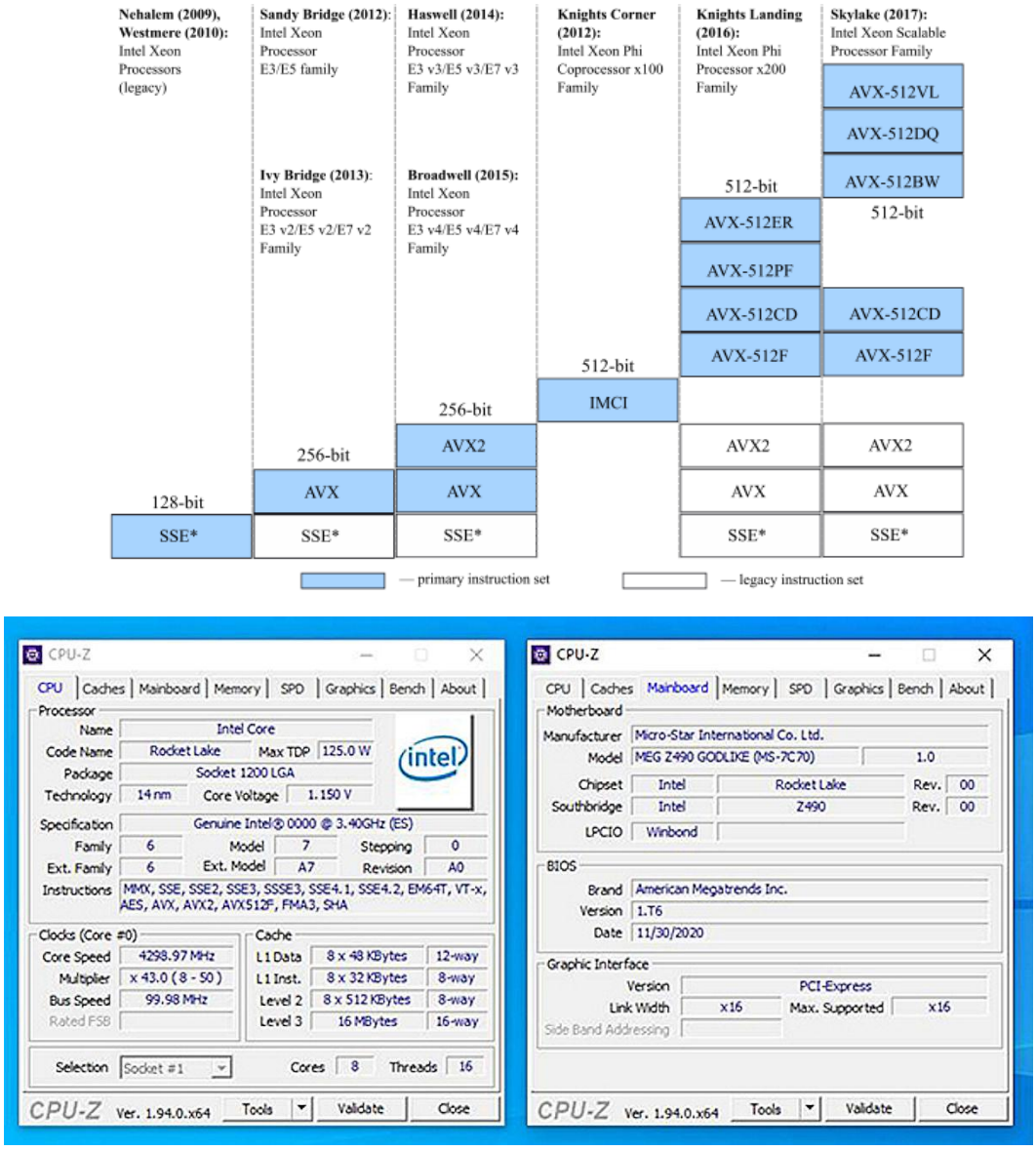

Рис. 1.12 Процесор Intel i5-11400 (Rocket Lake) : набір інструкцій

Останні моделі процесорів Intel 11-го та 12-го покоління (2020-2021 р.) мають підтримку деяких наборів інструкції AVX-512. Підтримка усіх можливих наборів SIMD-інструкцій реалізована у серверних процесорах Intel Xeon.

AVX-512F: Маскування. Нагадаємо, що маска — це певний набір значень, що діє як фільтра елементів даних. Процес застосовування бітової маски називають «маскуванням» оскільки одну частину елементів даних він відкриває, а іншу частину – приховує, а також змінює. Операції, що використовують бітові маски – перестановка елементів у векторі, перемішування двух векторів, векторизація циклів із операторами if/else. У порівнянні з AVX2 в AVX-512 маскування вдосконалено - в AVX-512 використовуються окремі маскові регістри __kmask, що не пов’язані із векторними регістрами. На відміну від AVX2 , де маскування здійснювалось лише над 128-розрядними половинами 256-розрядних регістрів, у AVX-512 маскування можливе над повною довжиною регістру, а також додано підтримку усіх типів даних (8-, 16, 32-, 64-, 128-, 256- розрядних смуг 512-розрядного регістру). З точку зору продуктивності коду векторний код, що використовує маски став швидшим (Рис 1.13): маскування в AVX-512F не потребує використання результату проміжного регістру (Рис 1.13) та другої операції із цим проміжним результатом)

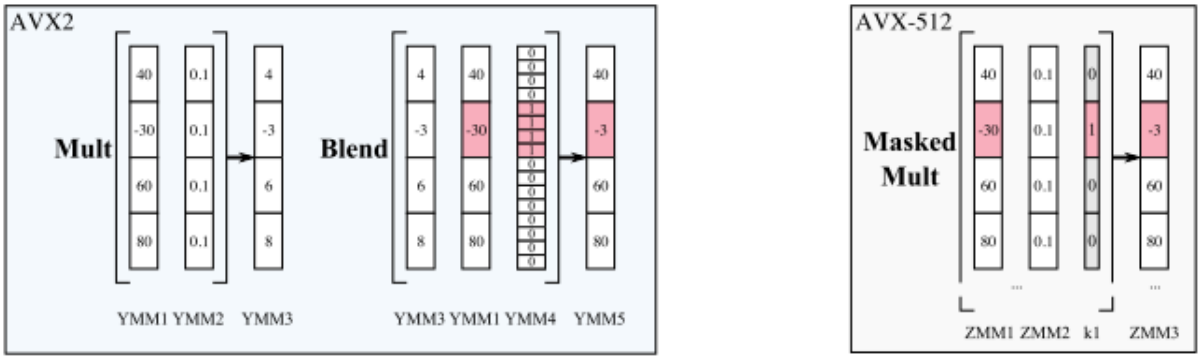

Рис. 1.13 Операція перемішування із застосуванням маски (AVX2, AVX-512) [24]

На Рис. 1.13 Показано, що для маскованого множення AVX2 потребує 2 операції: звичайне векторне множення (vpmul) та змішування (vpblend), AVX-512 потрібно лише одна атомарна операція множення із k-регістром. AVX2 потрібно 5 ymm регістрів, AVX-512 – 3 ymm регістра +1 k-регістр.

AVX-512F : зтискання та розширення. Було додано інструкції . vcompress та vexpand. Інструкція зтискання потребує 2 аргументи, по перше елементи даних з регістру, по друге бітову маску, на основі значень якої, результат записується у

вихідний регістр. Інструкція розширення зчитує елементи даних з вхідного регістра і переміщує їх у вихідний регістр у смуги відповідні до значень бітової маски. Такі інструкції в наборі AVX2 потребували 2 операцій – перемішування та запис із застосуванням додаткового регістру для проміжного результату.

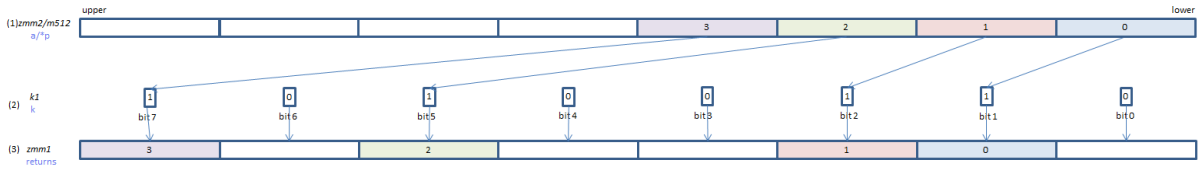

Рис. 1.14 Приклад роботи команди перемішування vexpand [27]

На Рис 1.14. показано роботу команди vexpand – завантажуються тільки ті із 16 послідовних елементів даних, відповідне значення яких у 16-бітовій масці дорівнює 1. Порозрядно бітова маска перевіряється – при значенні 1 елемент даних відповідно до номеру розряду маски переміщується у відповідну смугу регістру результату.

AVX-512 : просунуте перемішування (shuffle).

Для транспонування матриць AVX-512F дозволяє використати менше інструкції: AVX-512 інструкції vperm2q роблять перестановки елементів даних ZMM-регістрів та переміщують результат у вихідний регістр. За чим слідує змішування результатів перестановки над 2 векторами та їх вирівнювання. Для AVX2 було необхідно більше повільних операцій (із більшою затримкою): декілька vmovups, vinsertf128 та vunpckhpd. Інструкція vpblendmq перемішує елементи даних з двох регістрів ZMM за даними бітової маски із регістру kmask, і valignd об'єднує результати у вихідний регістр (Рис. 1.15).

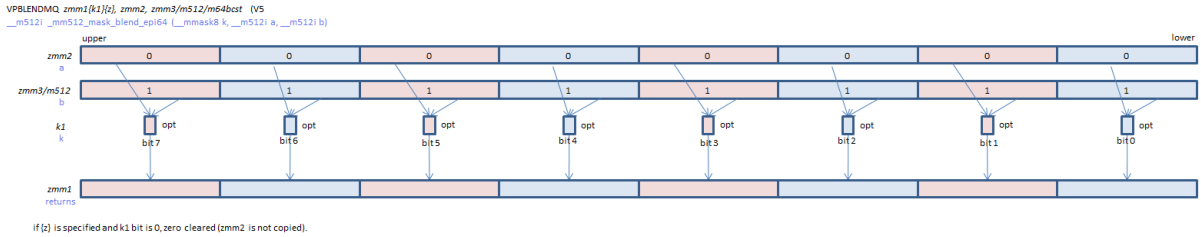

Рис. 1.15 Приклад роботи команди перемішування vplendmq [27]

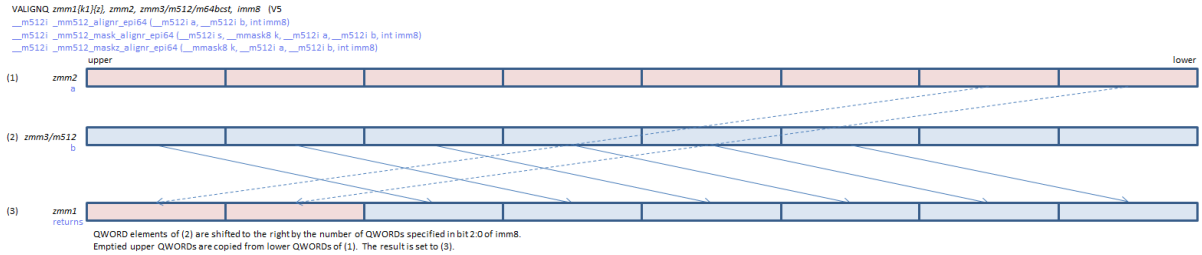

Рис. 1.16 Приклад роботи команди вирівнювання valignq [27]

AVX-512F: інструкції збирання та розсіювання (vpgather та vpscatter). Операції збирання дозволяють швидше апаратно завантажувати у ZMM регістр елементи даних із несуміжних локацій у пам'яті із з інтервалом між цими елементами N, наприклад векторизувати цикл for(i = 0; i < 10000; i++) { X[i] = Y[i*{N}]; } (Рис.). Операції посіву дозволяють швидше апаратно зберігати дані із ZMM регістру у несуміжні локації пам'яті з інтервалом N, тобто for(i = 0; i < 10000; i++) { X[i*{N}] = Y[i]; } (Рис. 1.17)

AVX-512F апаратна трансляція (broadcasting) – нові інструкції клонують один скалярний елемент даних по усіх смугах векторного регістру апаратно. У AVX2 трансляція скалярного елементу данних у вектор емулювалась - здійснювалось за допомогою однієї проміжної операції та явного використання додаткового векторного регістра (Рис. 1.18)

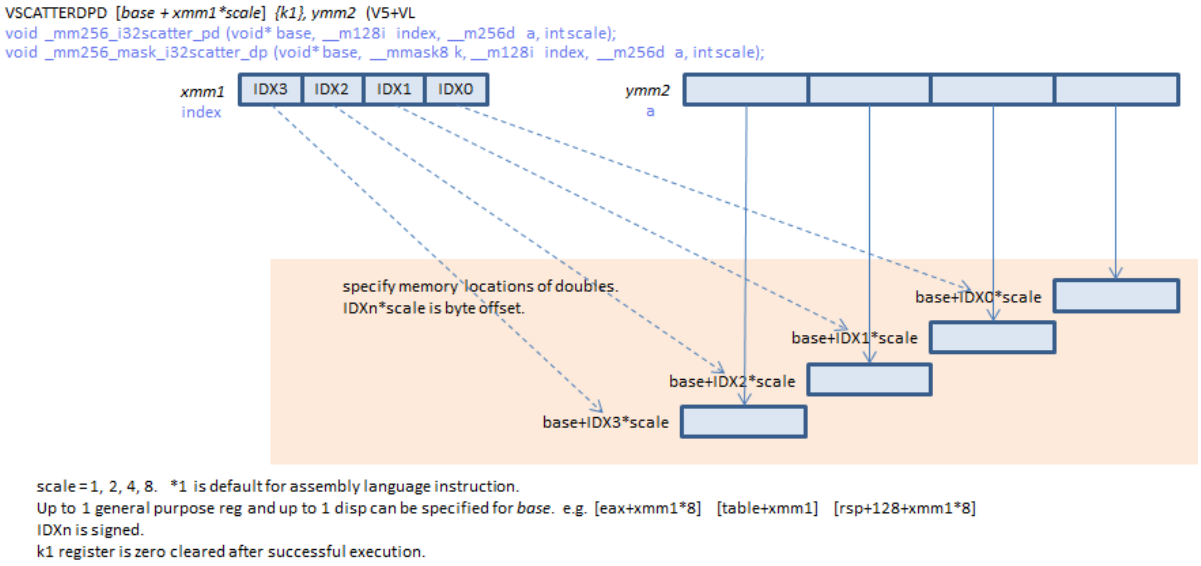

Рис. 1.17 Приклад роботи команди вирівнювання _mm_mask_i32scatter_pd [27]

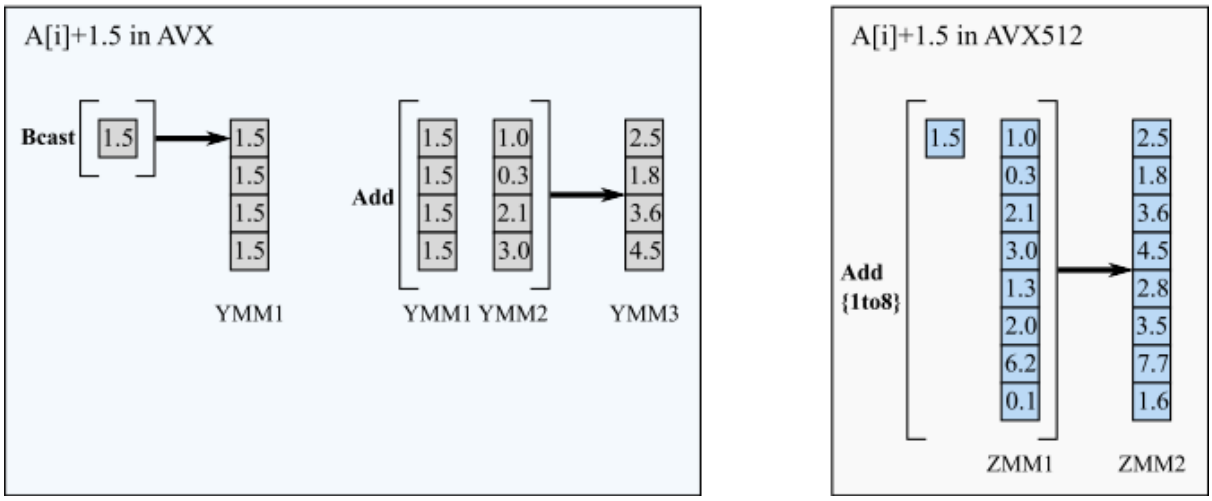

Рис. 1.18 Операція трансляції скалярного елемента у векторний регістр (AVX2, AVX-512) [24]

На рис. 1.18 показано економію тактів процесора за рахунок нових інструкцій при додаванні скалярного елементу даних до усіх смуг векторного регістру одночасно – у AVX2 було необхідна ще одна інструкція по передзавантаженню цього елемента у додатковий регістр.

AVX-512F: інструкції тернарної логіки. Оператор тернарної логіки приймає три вхідних векторних регістра (тому тернарний) та таблицю-маску. Інструкція формує тріплети із відповідних розрядів трьох вхідних регістрів (2^3=8 варіантів) та порівнює кожен тріплет із табличним значенням (8 можливих) та копією значення відповідного біта із таблиці у біт векторного регістру-приймача. Аналогів у попередніх наборах SIMD- інструкцій не існує (Рисунок 1.19).

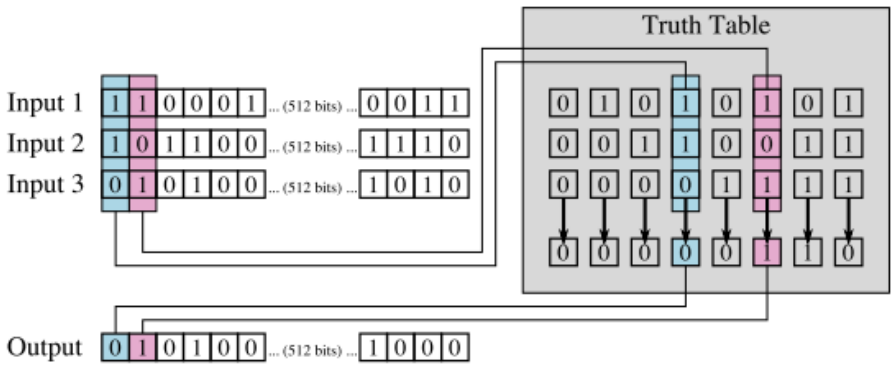

Рис. 1.19 Інструкція тернарної логіки _mm512_ternarylogic_epi64 (AVX2, AVX-512) [24]

AVX-512CD (від англ. CD – Conflicts Detect) набір інструкцій, що дав змогу безконфліктної векторизації циклів із можливою взаємозалежністю елементів даних типу M[N[i]] *=2, де якщо N[i]==N[j] j!=i, існує необхідність попередніх обчислень. Така операція апаратно розділена на 3 етапи: 1) завантаження і значень Ni у ZMM-регістр, інструкція vpconflict визначає векторні смуги із однаковим N[i] (конфліктні). 2) до безконфліктного набору застосовується векторне множення 3) до конфліктних даних застосовується послідовне (скалярне) множення (Рис.1.20).

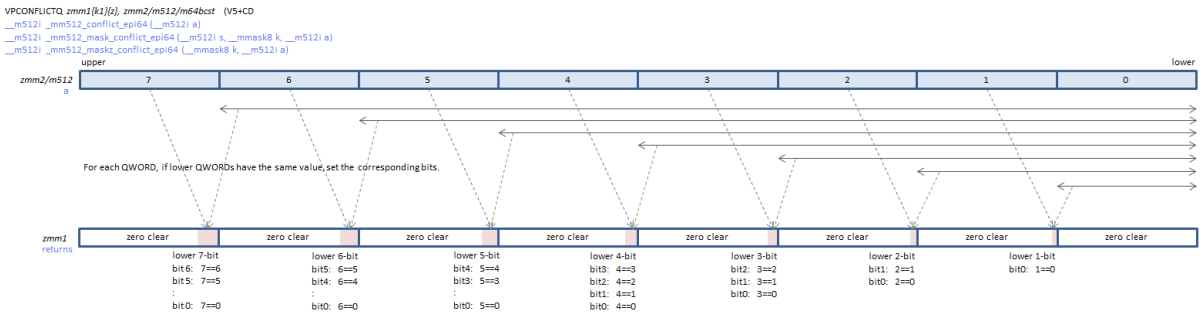

Рис. 1.20 Інструкція визначення конфліктних даних [27]

AVX-512F : контрольоване округлення. Інструкція дозволяє обирати бажаний режим округлення не залежно від типу даних (тобто долати встановлено апаратно режим округлення), що передається як додатковий аргумент інструкції. Також можливе відключення округлення за допомогою спеціального значення параметра.

Висновки до розділу 1

Архітектура і апаратна реалізація – це різні поняття, але апаратна реалізація базується на одній або декількох архітектур або їх елементів. Наприклад апаратних реалізації сучасних процесорів є поєднанням архітектури CISC та RISC (але з переважаючою долею окремою) – наприклад, у апаратних реалізаціях процесорів ARM присутні деякі «довгі інструкції» у декілька тактів, що виносяться в окремий конвеєр. І, навпаки, у процесорах з аппаратною реалізацію архітектури x86 – присутні деякі набори коротких (за 1 цикл) інструкцій. Також сучасний процесор завдяки технологічним вдосконаленням у апаратних реалізаціях зазвичай містить додаткові обчислювальні не скалярні блоки – такі модулі є додатковими розширення у вигляді окремої одиниці на кристалі для обробки довгих паралельних векторних інструкцій та за класифікацію Фліна є SIMD-інструкціями. Також переважна більшість сучасних центральних процесорів є мультипроцесорними системами. Тому сучасний центральний процесор можна назвати мультипроцесорною системою, де одна одиниця виконання (ядро) є типом SISD-системою із розширеннями у вигляді SIMD-системи. До того ж SIMD-інструкції та класичні інструкції інтегровані в один конвеєр та мають змогу відправлятись на спільні арифметичні одиниці (для SIMD-команд і команд для обчислення чисел з плаваючою точкою).

Можливості програмної та апаратної реалізації додаткових розширень у вигляді SIMD-розширень суттєво доступні для користувача апаратно і програмно у будь-якому компіляторі. Суть SIMD базується на можливості одночасної паралельної обробки елементів, що не залежать один від одного – наприклад 16 елементів по 32 біти – тому в залежності від апаратної реалізації додаткового розширення SIMD-команд вони здатні явно пришвидщити виконання програми до 64 раз, а в деяких випадках із-за специфічних апаратних реалізацій деяких команд – і більше. Але зазвичай є деякі обмеження у використанні таких інструкцій: можливості користувача розпаралелити алгоритм,й апаратно такі довгі інструкції можуть вимагати пониження тактової частоти системи до 2 разів для їх обробки.

РОЗДІЛ 2
ЕНТРОПІЙНИЙ КОДЕК DAALA ЯК ЧАСТИНА ВІДЕОКОДЕКУ AV1

У Розділі 1 було розглянуто особливості архітектури сучасних процесорів, які за замовчуванням мають архітектурне розширення SIMD та те, що ці розширення доступні програмісту у більшості компіляторів із підключенням відповідних прапорців ISA та застосуванням спеціальних інструкцій що генерують SIMD-команди центральному процесору – інтрінсіків. Це розширення грає ключову роль у роботі користувальницьких програм, суттєво зменшує час виконання програми. У Розділі 2 розглянуто загальну схему роботу відеокодеку (програми для стисенння відео-файлів) та детально розглянуто його окремий модуль - ентропійний стиснювач. Показано наскільки критичним стало наявність розширення SIMD у написанні та використанні сучасних програм-відеокодеків.

2.1. Короткий огляд схеми відео кодування та місце ентропійного кодування

Слово «кодек» є складовим двох слів кодер (стиснювач) і декодер (розтиснювач) – «Ко-Дек».Кодер (або енкодер) перетворює вихідні дані в стиснуту форму, що займає зменшену кількість бітів, перед передачею або зберіганням, а декодер перетворює стиснену форму назад у подання вихідних відеоданих.Програму що складається із кодеру та декодеру називають КОДЕК (enCOder/DECoder) [28, с.25].

Підмножини кольоровостей (color planes)

Відео зберігається як сукупність компонент кольору. Камера захоплює кольорове світло і розділяє його на основні компоненти - червоний, зелений, синій кольори (відповідно компоненти $R, G, B \in [0,255]$). Натомість згідно єдиного стандарту ITU-R BT.601 [29] дані зберігаються у форматі YCrCb, де Y-скравість, Cr та Cb – колірність (червоний/зелений), що перетворюється згідно (2.1). У файлі ці компоненти зберігаються як послідовність даних типу unsigned char.

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ 0.439 & -0.368 & -0.071 \\ -0.148 & -0.291 & 0.439 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} \quad (2.1)$$

Якщо для кожного просторового місця кадру присутні 3 зразка кожної з компонент, то компоненти є закодованими в «4:4:4», якщо компоненти яскравості та кольоровості субдискретизуються як 2 до 1 по горизонталі – то «4:2:2», а якщо як по горизонталі, так і по вертикалі – то «4:2:0». Хоча кодування по схемі «4:2:0» має найменшу візуальну точність, сааме «4:2:0» є кодуванням за замовчуванням у більшості додатків [30] (Рис.2.1).

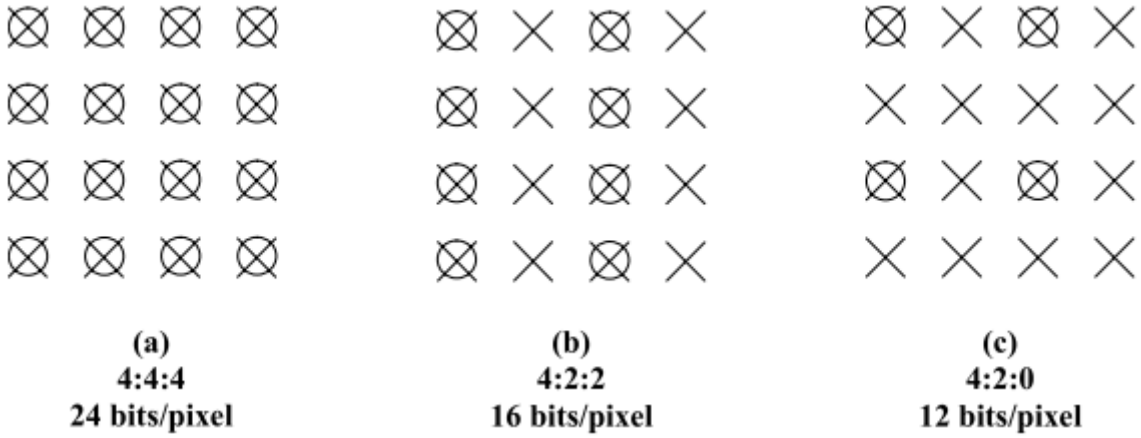

Рис. 2.1 Формати відео у контексті компоненти кольоровості [30]

Доцільність використання формату «4:2:0» підтверджено наочно – візуальна система людини (HVS) менш чутлива до компонент кольоровості у порівнянні із компонентами яскравості [31]. Тому компоненти кольоровості зберігають частково – на ділянку зображення розміром 2 на 2 точки – по 1 компоненту кольоровості (Рис 2.1, (c)).

Стиснення розміру відео (або кодування відео)

Стиснення (кодування) – це процес зменшення обсягу даних. Стиснення (кодування) відео — це процес перетворення відео у форматі послідовного набору пікселей у відмінний формат за певним алгоритмом. Наприклад, «необроблене» цифрове відео у форматі FullHD має бітрейт (потік бітів на секунду) у 216 Мбіт на 1 секунду (1920x1080x3x8біт на 30 кадрів/с), що ускладнює його мовлення по мережі – методи стиснення відео дозволяють у сотні разів зменшити бітрейт відео.

Методи стиснення відео поділяються на методи із втратами (with lose) і методи без втрат якості відео (loseless). Далі розглянуто методами стиснення з втратами – стиснене відео є неточним відтворенням оригінального відео. Методи стиснення відео базуються на надлишковості даних у просторовій області, та на змінній часовій області. Просторова область — це набір пікселів у межах одного зображення. Змінна часова область – це набір послідовних зображень на певний момент часу. Методи кодування відео застосовують методи компенсації руху (motion compensation), що використовують часову надлишковість, що є головною відмінністю між кодуванням відео (наприклад, MPEG) та кодуванням зображень (наприклад, JPEG) [32].

Блок-схема на Додатку В описує узагальнений процес стиснення відео. На верхніх трьох блоках вказані етапи кодування, на нижніх чотирьох - етапи декодування відео, серед яких і післяопрацювання відео. Післяопрацювання є опціональним процесом – не є обов'язковою частиною методів стискання відео, що будуть зазначені нижче. Інтер-кодування, передбачення руху та ентропійне кодування описані нижче. В роботі розглянуто типи об'єктів, що входять у цифрову відео послідовність (Додаток В. Flow-chart діаграма звичайного відеокодування). Відеопослідовність є об'єктом вищого рівня ієрархії. Відеопослідовність складається з множини зображень (кадрів). Кожний кадр складається з розділів, що називаються фрагментами (tiles). Кожен "тайл" складається з рекурсивного набору макроблоків (рекурсивно означає, що окремий макроблок циклічно поділяється на субмакроблоки, які також поділяються поки не зменшаться до розміру 4x4 пікселі). Піксель (точка, семпл) є найменшою одиницею відеопослідовності. Схематично зображення показано на рисунку 2.1.

Внутрішньо-кадрове кодування (або інтра-кодування)

Інтракодування є кодуванням просторової надлишковості, тому є спільним методом як для кодування відео, так і для кодування зображень. Хоча інтеркодування (прогнозування руху у часовому просторі) має більшу вагу стиснення, інтракодування зображення не залежать від інших кадрів. Інтракодовані зображення є першим типом зображення відеопослідовності і є

корисними в кодуванні різких змін сцен та пошуку (посилання) вперед і назад. Процес внутрішньокадрового кодування складається із декорелюючого перетворення (дискретне косинусне перетворення – DCT) [33] та квантування коефіцієнтів перетворення.

Перетворення DCT є перетворенням, що змінює представляння інформацію зображення із двовимірного в одновимірне (в лінійну частотну область). Кожен відповідний DCT коефіцієнт у блоці 8x8 вказує на внесок іншої «базисної» функції DCT до вихідного блоку зображення. Основна функція найнижчої частоти (зверху зліва Рис. 2.1 а) має назву DC-коефіцієнта і представляє середню яскравість блоку.

Дискретне косинусне перетворення (DCT) здійснюється над X, блоком N × N вибірок - вибірками зображень або залишковими значеннями після передбачення, для отримання Y, блоку N × N коефіцієнтів. DCT та зворотній до нього IDCT можна записати в через одиничну матриці перетворення A. Прямий DCT Y, блоку вибірки N × N, має вигляд:

$$Y = AXA^T (2.1)$$

а зворотний IDCT має вигляд:

$$X = A^T YA^T (2.2)$$

де X — матриця пікселів, Y — матриця DCT-коефіцієнтів, а A — матриця перетворення N × N. Елементами матриці A є:

$$A_{ij} = cos\frac{(2j+1)y\pi}{2N}, \text{де} C_i = \sqrt{\frac{1}{N}}, (i=0), C_i = \sqrt{\frac{2}{N}}, (i>0) (2.3)$$

Дискретно-косинусне перетворення безпосередньо не стискає матрицю вибірок. Навпаки, наприклад, для області 8x8 пікселів (піксель - 8 біт) DCT створює відповідний блок із більш довгими коефіцієнтами DCT (коефіцієнт - мінімум 11 біт), для можливості зворотного перетворення. Стиснення стає можливим завдяки наочному факту – для більшості користувальницьких зображень блоки DCT-коефіцієнтів мають нерівномірний розподіл – DCT-перетворення концентрує енергію у низькочастотних коефіцієнтах, а більшість коефіцієнтів є близькими до нуля. Стиснення і досягається по-перше, шляхом

відкидання близьких до нуля коефіцієнтів, а друге через квантування та кодування приблизно однакових коефіцієнтів - описано нижче) [34, с.14].

Квантування значень коефіцієнтів (застосовується після дискретно-косинусного перетворення) – процес округлення до найближчого степеню числа два (зазвичай 2, 4, 8, 16 .. 64) – які іноді називають «мертвою зоною». Чим «ширша» мертва зона тим більше коефіцієнтів буде округлено до нуля – і відповідно стиснення буде більшим, але будуть великі втрати частоти коефіцієнтів та відносної якості декодованої картинки відповідно.

Зигзагоподібне сканування (застосовується після квантування) – перепорядкування зигзагоподібним шаблоном позицій коефіцієнтів у блоці, що у більшості випадків дозволяє отримати довші серії нулів, які можна більш ефективно кодувати. Сортування блоку DCT-коефіцієнтів, оскільки встановлено наочно – довгі серії нулів після DCT-перетворення та квантування розташовуються навскіс блоку.Після квантування та зигзагоподібного сортування ентропійний кодер перетворює коефіцієнти у ефективне двійкове представлення (роботу ентропійного кодеру буде детально розглянуто у п. 2.2).

Між-кадрове кодування (інтер-кодування)

Послідовні зображення у відеопослідовності співвідносяться одне до наступного – багато деталей залишаються і в наступному кадрі – часова надмірність. Замість кодування одного зображення послідовності, інтеркодування здійснює кодування тимчасової надмірності – формує прогноз наступного зображення з використанням попереднього, що було закодовано (опорного або еталонного зображення). Спочатку енкодер поточне зображення розділяє на макроблоки, а опорне зображення енкодер використовує для знаходження передбачення поточного зображення відносно опорного. Енкодер здійснює пошук найбільших макроблоків в опорному (посильному) зображенні, щоб створити оцінку поточного зображення, що він кодує і записує співвідношення у вигляді векторів руху. Енкодер (програма для кодування відео) у інтер-кодуванні зазвичай використовує значення помилки передбачення як метрику вибору найкращих макроблоків. У більшості енкодерів використовуються такі метрики помилки

передбачення – мінімальні значення середньоквадратичної помилки (Mean Square Errors – MSE) та суми абсолютних різниць (Sum of Absolute Difference – SAD) відповідно. Враховуючи два дискретних 2-D сигнали x_i і y_j розміру M x N, SAD(x, y) визначається як:

$$SAD(x,y) = \frac{1}{M \cdot N} \sum_{i=1}^{M} \sum_{j=1}^{N} |x_{ij} - y_{ij}| \quad (2.4)$$

Середньоквадратична помилка між двома сигналами x і y визначається як:

$$MSE(x,y) = \frac{1}{M \cdot N} \sum_{i=1}^{M} \sum_{j=1}^{N} (x_{ij} - y_{ij})^2 \quad (2.5)$$

Енкодер асоціює вектори руху з кожним макроблоком, щоб описати просторове розташування кожного макроблока в опорному зображенні відносно поточного зображення у двомірній системі координат. Процес оцінки передбачень для поточного зображення називається оцінкою руху (motion estimation) (дивись Рис.3). Якщо для передбачення використовується лише попереднє зображення – то така схема має назву «прогнозування вперед» (forward estimation), якщо використовується ще й майбутнє зображення – «двонаправлене» передбачення (рис 1.8). Після того, як для поточного зображення створено зображення передбачення з компенсацією руху (ще називають «картою векторів руху), залишок між зображенням передбаченням руху та фактичним зображенням кодується по схемі внутрішньогокадрового кодування. Для кожного інтеркодованого зображення передаються як вектори руху, так і внутрішньо кодоване залишкове зображення.

Кодування кожного макроблоку закінчується символом «кінець блоку» (end-of-block – EOB). Для ефективного зберігання коефіцієнтів нульові коефіцієнти відкидаються, зберігаються тільки їх кількість між ненульовими коефіцієнтами (серії нулів). Кінцем послідовності, що кодується, є останній символ, що не дорівює 0. А оскільки після зигзагоподібного сортування більшість ненульових коефіцієнтів знаходиться на початку послідовності, це дозволяє уникнути передачі більшості нульових коефіцієнтів, і зменшити розмір інформації, що закодовано (Рис. 2.2).

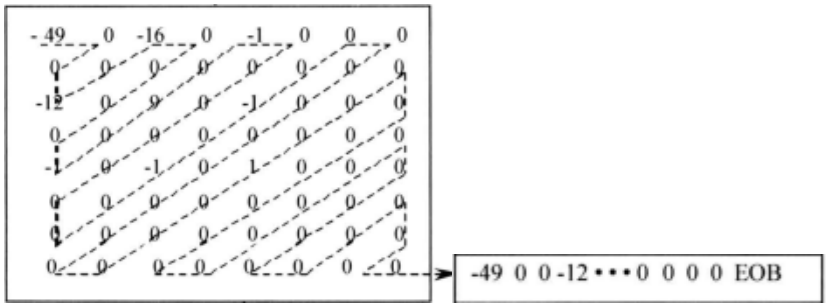

Рис. 2.2 Зигзаго-подібне сортування макроблоку даних у схемі відеокодеку [34]

У самому кінці схеми роботи декодеру (Додаток В) відбувається ентропійне кодування – стиснення результатів кодування інтра-, інтер-кодування, дискретно-косинусного перетворення та векторів руху, щоб суттєво полегшити вагу передачі та зберігання інформації, що закодована. Тобто, як зазначалось вище, кодування з прогнозуванням і DCT-перетворення усувають тимчасову та просторову надмірності послідовності зображень відеоряду, а ентропійне кодування, у свою чергу, додатково усуває ентропійну надмірність (надмірність у даних) із усуненою тимчасовою та просторовою надмірністю. Для ефективного представлення елементів множини бітів (квантованих DCT-коефіцієнтів, векторів руху відео послідовності) кожен елемент записується як коротший набір бітів. Дані кодуються без втрат через наявну надмірність в даних. Наочно встановлено, що у природному відео деякі елементи синтаксису зустрічаються частіше за інші, а тому ентропійний кодер використовує коротше бітове поле щодо елементів з високою ймовірністю, та довше бітове поле щодо елементів з низькою ймовірністю. Середня кількість бітів, що необхідна для кодування елементу даних є показником ентропії даних.

2.2. Основні методи ентропійного кодування

Методи ентропійного кодування змінної довжини

Методи, в яких кожний закодований символ має різну довжину, прийнято називати методами кодів змінної довжини (VLC). У свої працях Шеннон визначив

доцільну нижню межу для середньої довжини кодового слова [35]. Найпростіший VLC метод що задовольняє умові Шеннона був описаний Хаффманом [36]. Інший тип VLC - це код Голомба [37], що використовується в H.264. В Таблиці 2.1 показаний фрагмент таблиці кодів змінної довжини для кодування векторів руху відеокодеку MPEG-2. Зазвичай, короткі вектори руху є поширенішими за довгі вектори руху. Тому, меншим векторам руху призначаються короткі набори бітів (кодові слова), але в залежності контенту відео може бути навпаки. Нульовий рух представляє статичний об’єкт (область) відеопослідовності (напр. - фон, що не рухається).

Таблиця 2.1

Коди Голомба

Code Number	Code Symbol	Codeword
0	=>1	1
I	=>10	010
2	=>11	01l
3	=>100	00100
4	=>101	00101
5	=>110	00110
6	=>111	00111
7	=>1000	0001000

Джерело: Golomb S., “Run-length encoding”[37]

У кожному стандарті відеокодеку таблиці пар символ-кодове слово формується експериментальним шляхом розробниками чи авторами. Такі таблиці можуть бути такими, що оновлюються в процесі роботи енкодеру на основі сбору статистики для покращення рівня ентропії. У дещо старому стандарті MPEG-2 дві доступні таблиці VLC, які не оновлюються (статичні) тоді як в VP9 – їх багато і вони оновлюються через задану n-ну кількість кадрів (адаптивні).

У інформатиці та теорії інформації кодування Хафмана відомий як алгоритм ентропійного стиснення даних без втрат (lossless) [38]. Термін відноситься до використання кодів змінної довжини (VLC) для кодування вхідного символу, де VLC-таблицю отримують відповідно до оціночних ймовірностей вхідного

символу. Алгоритм полягає у записі меншої кількості бітів для кодування елементу даних із більшою ймовірністю [39]. Будь-яке кодове слово у VLC-таблиці не є префіксом будь-якого іншого [40], що гарантується деревом Хаффмана. Форма передачі VLC-таблиці потоці даних є неявною інструкцію, згідно якої декодер здатний побудувати цю VLC-таблицю [41]. в якій кожен символ є листом і коренем.

Як видно з таблиці 2.1, мінімальна кількість бітів, що може бути призначена одному вхідному символу – 1 (призначається символу 0). Тобто в середньому символу не можна призначити бітів менше 1 – що є основним обмеження кодування Хаффмана у порівнянні з іншими методами кодування. Цю проблему вирішує у т.ч. арифметичне кодування.

Методи арифметичного ентропійного кодування

Іншим класом методів ентропійного кодування є арифметичні [41]. Арифметичні методи теоретично ближче до нижньої межі Шеннона, і на відміну від VLC використовують нецілу кількість бітів на кодове слово (арифметичні методи розглянуто у п. 2.2).

Арифметичне кодування присвоює послідовність бітів не символу, а повідомленню - тобто, набору символів – тобто об’єктом кодування є повідомлення (набір символів) [43, с.146-162]. На відміну Хафмана, в арифметичному кодуванні не використовує статичну кількість бітів для кожного символу. Кількість бітів, необхідна для кодування символу - змінна, і залежить від ймовірності, що присвоюється цьому символу. Символи з нижчою ймовірністю потребують більше бітів, символи з вищою ймовірністю – менше бітів [44].

Ідея арифметичного кодування – призначити кожному символу інтервал. Рекурсивно, починаючи з інтервалу [0...1), наступний інтервал поділяють на декілька підінтервалів, довжина котрих пропорційна до поточної ймовірності відповідних символів [45].

Підінтервал закодованого символу рахується за інтервал для наступного символу. Результатом - є інтервал останнього закодованого символу [39]. Коротко програмна реалізація арифметичного кодування показана на Рис. 2.3.

```
BEGIN
low = 0.0; high = 1.0; range = 1.0;
while (symbol != terminator)
{   get (symbol);
low = low + range * Range_low(symbol);
high = low + range * Range_high(symbol);
range = high - low;      }
output a code so that low <= code < high;
END.
```

Рис. 2.3 Програмна реалізація арифметичного кодування [46]

Нехай маємо множину елементів { 0, 2, 14, 136 , 222 }. У таблиці 2.2 показона частота їх появи у множині та відповінй діапазон ймовірності [0..1).

Таблиця 2.2

Таблиця бінарного арифметичного кодеру (Boolean CABAC)

Символ	Ймовірність	Діапазон
0	0.63	[0,0.63)
2	0.11	[0.63,0.74)
14	0.1	[0.74,0.84)
136	0.1	[0.84,0.94)
222	0.06	[0.94,1.0)

Джерело: Chanda S., Rana U. – Evaluation of Huffman-Code and B-Code Algorithms for Image Compression Standards[47]

Графічне представлення роботи алгоритму арифметичного кодування із повідомленням у напрямку «зліва направо» показано на Рисунку 2.3. Як бачимо, перший діапазон ймовірності – [0,63..0,74) – оскільки кодується символ «2». Далі йде наступний символ «0» із табличною ймовірністю [0..0,63), який потрібно масштабувати в межах інтервалу [0.63..0,74] – маємо інтервал [0,63 + 0 .. 0,63 + (0,74-0,63)*0,63) = [0,63..0,69) і так далі інтервал рекурсивно звужується до обробки останнього символу (Рис. 2.4).

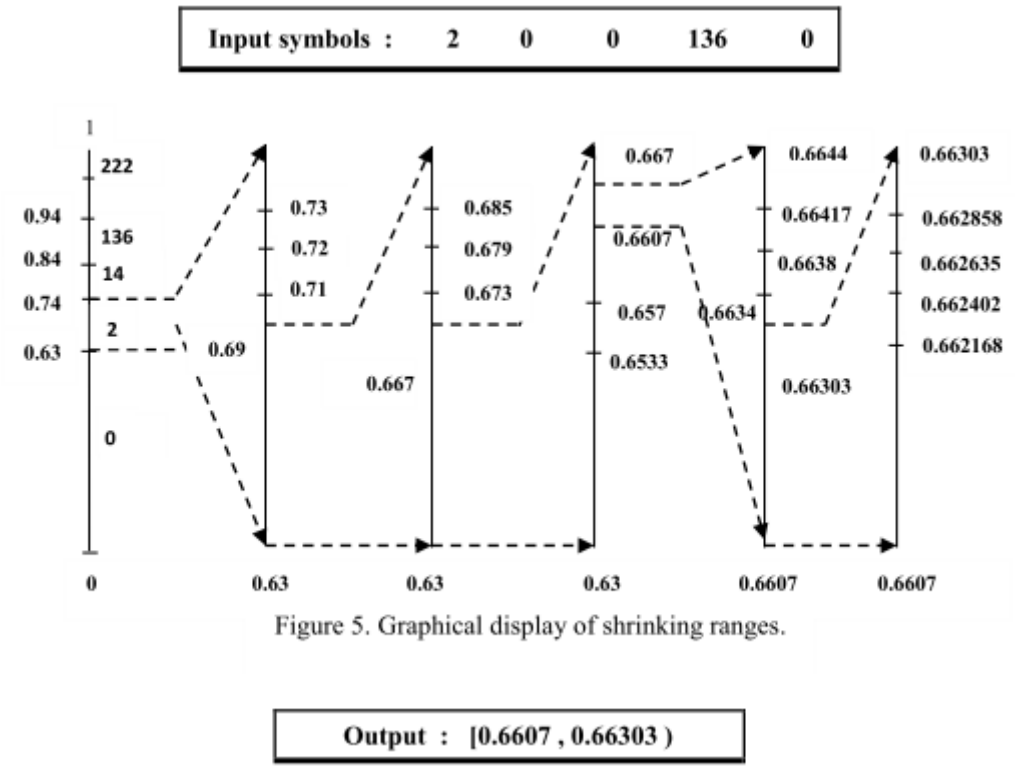

Figure 5. Graphical display of shrinking ranges.

Рис. 2.4 Основні кроки алгоритму арифметичного кодування набору символів { 2, 0, 0, 136, 0 } [47]

Результатом є закодований вихідний інтервал [0,6607, 0,66303). Послідовність бітів буде присвоєно числу яке буде знаходитись у вказаному інтервалі. Із реалізації алгоритмів бачимо, що складність програмної реалізації арифметичного кодування вище, ніж кодування Хаффмана – вона потребує більше циклів та більшу точність та роботу із дійсними числами (або додатковими циклами у цілочисельному ренормуванню).

Програмна реалізація алгоритмів кодування Хаффмана та арифметичного кодування за допомогою інструментів програмування Matlab наведено у Таблиці 2.3.

Таблиця 2.3

Кодування Хаффмана (зліва) та арифметичне (зправа)

```
%**********Start Huffman Coding
for time= 1:100
tic
k=0;
VECTOR-HUFF(1) = V(1);
   for l= 1:m
          a=0;
          for q=1:k
          if(VECTOR (l) == VECTOR-HUFF (q));
                a=a+1;
          end
          end
          if (a==0)
                k=k+1;
                VECTOR-HUFF(k) = V(l);
         end
    end
    for u=1:k
          a=0;
          for l=1:m
          if (V(l)== VECTOR-ARITH(u))
                a=a+1;
          end
                VECTOR-HUFF-NUM(u)= a;
          end
     end
     for i=1:k
     P(i)= VECTOR-HUFF-NUM (i)/(m1);
     end
 dict = huffmandict(VECTOR-HUFF,P);
 hcode = huffmanenco(VECTOR,dict);
 [f1,f2] = size(hcode);
 Compression ratio = b0/f2
 toc
end
```

```
%*********Start Arithmetic Coding
for time= 1:100
tic
k=0;
VECTOR-ARITH(1) = V(1);
   for l= 1:m
          a=0;
          for q=1:k
          if (V(l) == VECTOR-ARITH (q));
                a=a+1;
          end
          end
          if (a==0)
                k=k+1;
                VECTOR-ARITH (k) = V(l);
         end
    end
    for u=1:k
          a=0;
          for l=1:m
          if (V (l)== VECTOR-ARITH (u))
                a=a+1;
                Varith(l)=u;
          end
          VECTOR-ARITH-NUM(u)= a;
          end
    end
code = arithenco(Varith,VECTOR-ARITH-NUM);
[f1,f2] = size(code);
Compression ratio = b0/f2
toc
end
```

Джерело: Chanda S., Rana U. – Evaluation of Huffman-Code and B-Code Algorithms for Image Compression Standards [47]

2.3. Ентропійне кодування відео AV1

Останні відеокодеки (Рис. 2.5) кодують інформацію шляхом двійкового арифметичного кодування: кожен символ може приймати лише два значення.

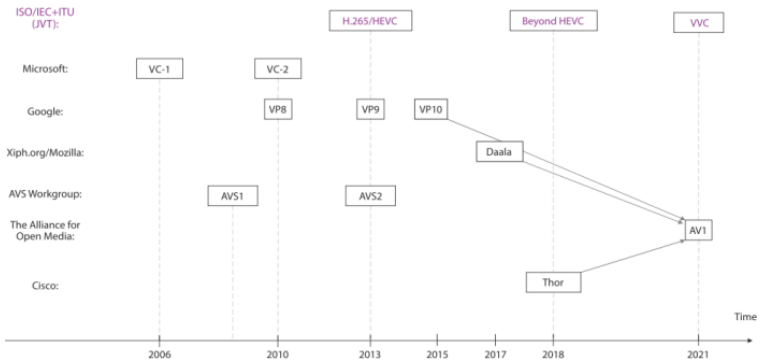

Рис. 2.5 Основні відеокодеки у світі у 2006-2021 рр.

Ентропійний енкодер Daala є діапазонним багатосимвольним ентропійним кодером та підтримує до 16 можливих значень на символ, що дозволяє кодувати менше символів [48]. Це еквівалентно паралельному кодуванню до чотирьох двійкових значень і зменшує послідовні залежності, дозволяючи апаратним реалізаціям використовувати нижчу тактову частоту, а отже, і меншу потужність.

Багатосимвольний ентропійний кодер не є інновацією Daala, схожий алгоритм використовувася в аудіо-кодеку компанії Opus. Однак, новацію він є у відеокодеках, зважаючи на те, що попередні відеокодеки обмежувалися двійковим (бінарним) арифметичним кодуванням. Хоча використання бінарного арифметичного кодування в стандартах H.264 і HEVC призвело до запатентованості бінарних відеокодеків, однак це не є основною причиною їх невикористання у Daala. Арифметичне кодування по своїй природі є послідовним, тому символи необхідно декодувати по одному (з одного потоку), що уповільнює процес декодування.

Алгоритм роботи діапазонного енкодеру використовує наближення без операцій ділення/множення, що називається кусково-цілочисельним відображенням [49, с.3-12]. Початкове наближення здійснює перегляд (підвищення) значень ймовірностей для символів наприкінці алфавіту і перегляд значень ймовірностей на початку. У Daala початкове наближення здійснюється обернено, оскільки нульові значення є символами із найвищим значеннями ймовірності. Переоцінка значень ймовірностей призводить до менших витрат на наближення, ніж недооцінка без зміни порядку алфавіту.

Кусково-цілочисельне відображення замінює операції множення та ділення звичайного арифметичного кодера на віднімання, операцію знаходження мінімального значення, та додавання. Так відображення дозволяє енкодеру ефективно працювати з довільним вхідним розподілом ймовірностей, без необхідності нормування розподілу до степенів двійки. У такій схемі Daala використовує звичайні арифметичні обчислення частот для моделювання розподілу ймовірностей до більших розмірів алфавіту, замість використання схем на основі таблиць, що використовує переважна більшість двійковими кодерів. У

програмній та апаратній реалізації кодеру оновлення значень набору ймовірностей займає декілька циклів - оновлюються 1-2 SIMD-інструкціями. Загальна кількість операцій оновлення у роботі кодеру становить близько 1%, що аналогічно CABAC [50, с.620], [51].

Кускове лінійне цілочисельне відображення

Якщо відомо, що в деякому інтервалі R/2 < t <= R (і, отже, величина R div t <=1), то можливе інше двочасткове відображення, в якому надлишок округлення розподіляється на більше одного цілого числа. Ця можливість врахована в рівнянні 3:

$$f(x,t,R) = \begin{cases} x & \text{If } x < d, \\ 2x - d, & \text{otherwise} \end{cases} \qquad (2.6)$$

де d = 2t - R – кількість значень в інтервалі [0, t), які виділені в одинарні одиниці в *R*, а *t-d* — кількість значень, які виділені подвійними одиницями в R.

Псевдокод на Рис. 2.6 ілюструє обчислення нових значень L’ і R’ з використанням цього підходу [54]. Цикл while на етапі 1 гарантує, що t і R знаходяться в правильному співвідношенні, щоб це наближення було дійсним.

1. While *t* <= *R/2* do
Set l < –2l && h < –2h && t <- 2 t
2. Set d <–2t - R
3. Set L’ <– L + max { l, 2l - d }
4. Set R’ max{ h, 2h - d } - max { l, 2l - d }
5. Renormalise to bring R’ into the required range

Рис. 2.6 Кускове наближення (енкодер) [54]

Розглянемо які значення можуть приймати R і t. Використаємо запис Моффата, нехай w — розмір слова конкретної архітектури, а 2^b — коефіцієнт масштабування, що використано для представлення L та R як цілих чисел, b <= w.

Якщо використовується побітова нормалізація, то R утримується в діапазоні $2^{b-2} < R \leq 2^{b-1}$. Величина t вважається частково нормованою у інтервалі $2^{f-1} < t < 2^f$, де $f \leq b-2$. Тобто, якщо взяти частину повного діапазону $1.0 = 2^b$, то величина R обмежується процесом ренормування до множини значень є [0,25..0,5], а t є [0,125 .. 0,25]. Таким чином, цикл while на кроці 1 на Рис.2.2

виконається максимум один раз – цей цикл рівносильний оператору if. Шаг 5 — цикл ренормування (цю операцію розглянуто пізніше).

В арифметичному декодері виконуються три відповідні операції. Спочатку має бути визначено значення x таке, що f (x,t,R) <D ≤ f(x+1,t,R), де D = V - L , а V є поточним «вікном» розміром b бітів у стисненому бітстрімі. Далі визначається інтервал [l, h), що містить значення x, цей інтервал визначає символ, що було записав енкодер на цьому кроці. В кінці, декодер налаштовує його значення R і D, щоб відобразити зміни, що енкодер записав в процесі кодування цього символу. Перші три кроки на Рис 2.6 показують обчислення зворотної функції відображення f^{-1} для величини V для знаходження цільового значення x. Потім шукається цільове значення (Крок 4) з використанням даних структури кумулятивних частот ([52], [53]), як тільки знайдено інтервал [l; h), декодер виконує операції звуження діапазону, відповідно таким, що зображено на Рис.2.7. Алгоритм роботи енкодера або декодера не містить операції множення та ділення [54].

1. While t <= R/2 do
   Set l <- 2l and h <- 2h and t <- 2t
2. Set d <- 2t - R
3. Set x min {D; (D + d)/2}
4. Determine the [l, h) interval for the source symbol corresponding to target x
5. Set R’ <- max{h, 2h - d} - max {l, 2l - d}
6. Set D’ <- max {l; 2l - d}
7. Renormalise to bring R’ into the required range

Рис. 2.7 Частичне наближення (декодування) [54]

Ентропійний кодек Daala використовує метод арифметичного кодування M-арних символів [51] для стиснення елементів синтаксису, де ціле число M ∈ [2, 14]. Кожен синтаксичний елемент Daala є символом певного алфавіту з N елементів, а контекст[3] складається з набору N ймовірностей (у вигляді кумулятивних функцій розподілу) разом із лічильником для полегшення швидкої ранньої адаптації. Таблиці розподілу є масивом із N елементів (N є [2..14])

[3]Контекст кадру - набір ймовірностей, що використовуються в процесі декодування [55]

ймовірностей із елементів довжиною 15 біт. Модель ймовірності оновлюється відповідно до закодованого символу: адаптація ймовірностей відбувається рекурсивно на основі фактору масштабування N – розміру алфавіту. Бітовий потік символів складають переважно DCT-коефіцієнти, вектори руху та режими передбачення, для кодування яких використано алфавіти розміром N≥2. Тому, у порівнянні із бінарним арифметичним ентропійним кодеком Daala забезпечує пропускну здатність приблизно у 2 рази.

Розглянемо M-арну випадкову величину, функція розподілу маси ймовірностей (PMF) у момент часу n визначається як

$$\overline{P}_n = [p|1(n), p_2(n), .., p_{M-1}(n), 1] \quad (2.7)$$

і кумулятивну функцію розподілу (CDF), яка має наступний запис:

$C_n = [\mathrm{c}_1(n), \mathrm{c}_2(n), \dots, \mathrm{c}_{M-1}(n), 1]$, де, $\mathrm{c}_k(n) = \sum_{i=1}^{k} p_{i_i}(n)$ (2.8)

Коли символ буде закодовано, отримаємо новий результат $k\epsilon\{1,2,..,M\}$. Далі оновлюється модель ймовірності :

$$\overline{P_n} = \overline{P}_{n-1}(1-\alpha) + \alpha\overline{e}_k \quad (2.9)$$

де $\overline{e}_k$ – вектор-вказівник, k-й елемент якого дорівнює 1, а інші – 0, α – коефіцієнт оновлення.

Для оновлення CDF, розглянемо $\mathrm{c_m}$(n) де m < k

$$\mathrm{c}_m(n) = \sum_{i=1}^{k} p_i(n) = \sum_{i=1}^{k} p_i(n-1)\,(1-\alpha) = \mathrm{c}_m(n-1)\cdot(1-\alpha). \quad (2.10)$$

Для випадків, де m ≥ k cases, маємо

$$1 - \mathrm{c}_m(n) = \sum_{i=1}^{k} p_i(n) = \sum_{i=1}^{k} p_i(n-1)\,(1-\alpha) = \big(1 - \mathrm{c}_m(n-1)\big)\cdot(1-\alpha). \quad (2.11)$$

З цього рівняння випливає наступне рівняння, де m+1>k . Переставляємо доданки, отримаємо:

$$\mathrm{c}_m(n) = \mathrm{c}_m(n-1)\cdot\alpha\big(1 - \mathrm{c}_m(n-1)\big) \quad (2.12)$$

Підсумовуючи, кумулятивна функція розподілу CDF оновлюється як:

$$\mathrm{c}_m(n) = \begin{cases} \mathrm{c}_m(n-1)\cdot(1-\alpha), m < k \\ \mathrm{c}_m(n-1)\cdot\alpha\big(1 - \mathrm{c}_m(n-1)\big), m \geq k \end{cases} \quad (2.13)$$

Daala зберігає ймовірності M-арних символів у формі множини значень кумулятивної функції розподілу розміром 2..14, у машинному представленні елементи множини масштабуються на величину 2^{15} для зберігання у вигляді числа цілочисельної точності.Арифметичне кодування використовує ці CDF-масиви безпосередньо під час стиснення символів [40]. Коефіцієнт оновлення ймовірності адаптується на основі відносної частоти появи цього символу в кадрі.

$$\alpha = \frac{1}{2^{3+I(count>15)+I(count>32)+min\,(log_2 M,2)}} \quad (2.14)$$

де I - подія має значення 1, якщо подія істинна, інакше 0.Це забезпечує швидшу адаптацію на початку кожного кадру.Моделі ймовірностей (масиви CDF) успадковуються із опорного кадру, номер якого передається в бітовому потоці.

Арифметичне кодування

В апаратному забезпеченні, критичним є пропускна здатність і величина основного множника, що перемасштабовує діапазон стану арифметичного кодування. Точність, яка необхідна для контролю діапазону ймовірностей, у програмній реалізації може бути без втрат замінено округленням від 16×15 біт (31 біт) до множника 8×9 біт (17 біт) [56]. Щоб покращити пропускну здатність на конкретній апаратній реалізації, ентропійний кодек Daala в програмній реалізації має працює так (не порушуючи математичну модель), щоб результат операцій множення був довжиною 16 біт. CDF-модель ймовірностей оновлюється і здійснюється з точністю 15 біт, але в ентропійному кодуванн лише 9 старших бітів подаються в арифметичний кодер, як показано на рис. 2.4. Нехай R - довжину інтервалу, на якій працює M-арний арифметичний кодер, а V - позначає комірки відповідного CDF-масиву.Обробка декодування зображена в Рис. 2.9 «Алгоритмі 1». Довжина масштабованого цілочисельного інтервалу R скорочується на 1/256 (аналогічно бітовому зсуву праворуч на 8 бітів) перед множенням, після чого результат операції кодера - добуток (R >> 8)× f вписується у 16 біт. Алгоритм, по якому працює скалярна функція ентропійного кодування представлено автором на Рис. 2.8.

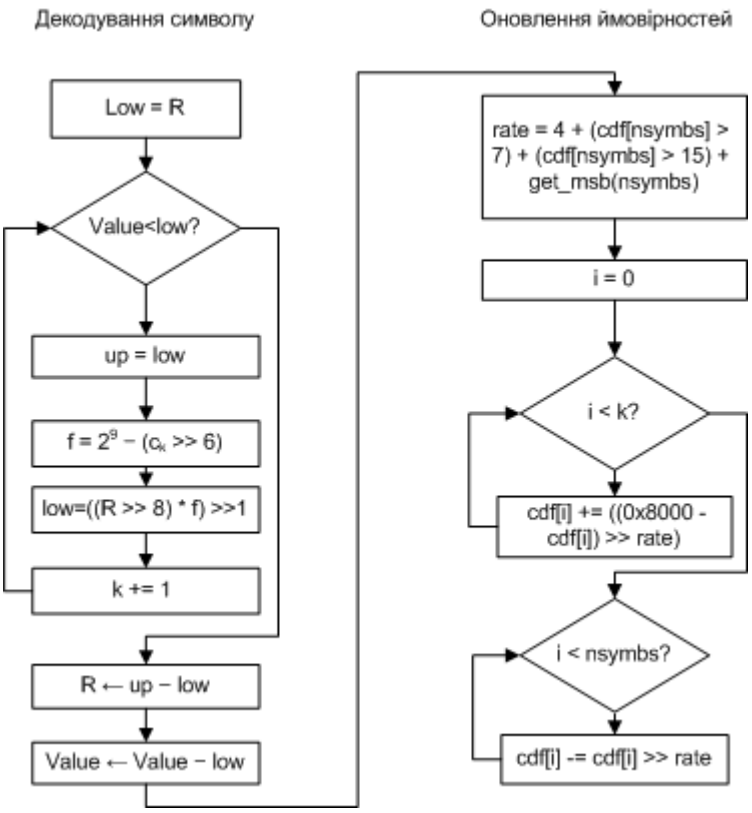

Рис. 2.8 Версія алгоритму мультисимвольного арифметичного ентропійного кодеру Daala [55], [56]

Відеокодек AV1 використовує адаптивне кодування – спочатку моделі ймовірностей (у вигляді масивів CDF-масивів) заповнюються значеннями, що встановлено за замовчуванням, а зі зміною кадру CDF-масиви оновлюються з використанням нового розподілу CDF, зазвичай деякого опорного кадру. Для цього енкодер AVI виконує R-D оптимізацію всього кадру, записуючи всі свої рішення. Потім він використовує статистику цих рішень, щоб обчислити оптимальні ймовірності для цього кадру, і приймає оптимальне рішення про те, кодувати чи ні явне перевизначення для кожного з них (модель Rate-Distortion[4]). Наразі Daala адаптує більшість ймовірностей, використовуючи прості підрахунки частоти, періодично змінюючи їх масштаб, оскільки загальна кількість перевищує 15 біт. Оскільки загальний підрахунок частоти, як правило, не є степенем двійки, ми

[4]Кодування з втратами характеризується двома аспектами а) швидкість передачі даних R: середня кількість бітів на вибірку (або на одиницю часу) та б) спотворенням D: вимірювання відхилення між вихідним сигналом s і відновленим сигналом s. Модель R-D полягає в пошуку максимально досяжної ефективності кодування (компроміс між бітовою швидкістю R і спотворенням D) [57]

адаптуємо метод Стюівера і Моффата для кодування цих значень без будь-яких множень або поділів (за винятком того, що ми переоцінюємо ймовірності для символів на початку алфавіту, замість кінець). Цей метод є лише приблизним, але має накладні витрати приблизно від 1% до 2% порівняно з теоретичною ентропією, яка подібна до CABAC. Коли загальний підрахунок частоти має ступінь двійки, ми використовуємо 15 x 16 —> 31 множення на кодові символи з незначними (< 0,01%) накладними витратами.

Основні обчислення у ентропійному кодування лягають на функцію aom_read_symbol. Згідно алгоритму ентропійного кодування daala, що використовується у репозиторії aom, спочатку знаходиться значення декодованого символу, далі ренормалізуються (приводяться до встановленого алгоритмом діапазону) поля dif, rng, cnt із відповідної структури struct od_ec_dec) ентропійного декодування, далі оновлюються ймовірності кумулятивного розподілу (представлені як масиви беззнакових слів). Таким 3 логічним крокам відповідають 3 окремі функції, які складають функцію aom_read_symbol:

- int od_ec_decode_cdf_q15(od_ec_dec *dec, const uint16_t *icdf, int nsyms);
- od_ec_dec_normalize(dec, dif, r, ret);
- update_cdf(cdf, ret, nsymbs);

Зазначені функції функції займають найбільше процесорного часу під час ентропійного декодування представлено автором на. Рис. 2.9.

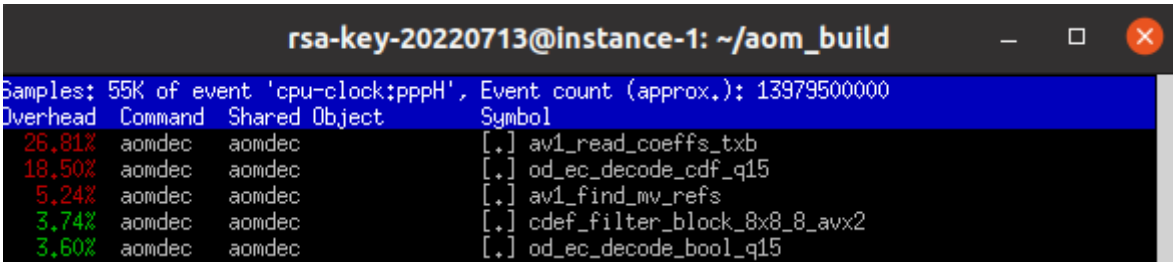

Рис. 2.9 Результати perf профілювання ентропійного декодування (файл виконання aomdec, знімок екрану)

Функція od_ec_decode_cdf_q15 є гарячою точкою (hotspot) алгоритму ентропійного декодування, і викликається із av1_read_coeffs_txb, яка на основі

розміру блоку та інших параметрів (простір кольоровості) визначає розмір та тип вхідного масиву кумулятивних ймовірностей (параметри cdf та nsymbs).

У функції od_ec_decode_cdf_q15 у файлі aom_dsp/entdec.c із кумулятивного розподілу обирається такий коефіцієнт, різниця між значенням якого і вірогідністю із масиву кумулятивного розподілу має мінімальне позитивне значення. Вхідними аргументами функції є масив cdf довжиною N+1 , що визначає кумулятивний розподіл для символу з N можливими значеннями. Результатом функції є змінний символ, який містить синтаксичний елемент, що було декодовано. Процес ентропійного декодування також змінює аргумент функції - вхідний масив cdf, для адаптації ймовірностей до вмісту бітового потоку - викликається функція update_cdf із функції read_symbol(cdf). Змінні cur, prev і symbol обчислюються по таким крокам [55, с. 359]:

```
cur == SymbolRange
symbol == -1
do {
symbol++
prev = cur
f = (1 << 15 ) - cdf[symbol]
cur = ((SymbolRange >> 8) * (f >> EC_PROB_SHIFT)) >> (7 - EC_PROB_SHIFT)
cur += EC_MIN_PROB * (N - symbol - 1)
} while (SymbolValue << cur )
```

Значення змінної SymbolRange встановлюється як різниця prev і cur, а SymbolValue різниця SymbolValue і cur.

У функції od_ec_dec_normalize у файлі aom_dsp/entdec.c. Діапазон і значення перенормуються по крокам: 1) Значення змінної bits задається як 15 – FloorLog2(SymbolRange).Змінна bits – це кількість нових бітів, що додаються до SymbolValue. 2. Змінна SymbolRange переобчислюється як SymbolRange << bits. 3. Змінна numBits обчислюється як Min(bits,Max(0, SymbolMaxBits)) — вказує кількість бітів, які необхідно читати з бітового потоку. 4. Змінна newData читається через процесу парсінгу f(numBits). 5. Змінна paddedData обчислюється як newData << (bits - numBits). 6. Змінна SymbolValue обчислюється як

paddedData^(((SymbolValue+1)<<bits)-1). 7. Змінна SymbolMaxBits обчислюється як SymbolMaxBits-bits [55, с. 360].

У функції update_cdf у файлі aom_dsp/entdec.c із кумулятивного розподілу обирається такий коефіцієнт, різниця між значенням якого і вірогідністю із масиву кумулятивного розподілу має мінімальне позитивне значення.

Функція нормалізації фактично оновлює три поля структури od_ec_dec: подвійне слово rng, слово rng, слово cnt, кожне за різними умовами, тому застосування розширення набору інструкцій щодо неї недоцільно. Доцільно розглянути і переписати із застосуванням розширення набору інструкцій SIMD функції od_ec_decode та update_cdf.

До того ж в залежності від довжини масиву кумулятивного розподілу можна зробити декілька спеціалізацій функції декодування сиволу та оновлення ймовірностей. Оскільки виклики функцій ентропійного кодування здійснюються із загальної функції декодування коефіцієнтів макроблоку (uint8_t av1_read_coeffs_txb) , яка на вхід приймає бітовий потік, та в залежності від порядкового номера останнього ненульового коефіцієнта блока викликає алгоритм ентропійного декодування із відповідним масивом кумулятивного розподілу відповідної довжини. Нагадаємо, що максимальне значення закодованого символу різне — від 2 до 14. На Рис. 2.10 автором показано, що для випадку 0 (case 0:), декодований символ знаходиться на відрізку [0..5], тоді як для випадку 3 (case 3:) — на відрізку [0..8], і відповідно вхідна і вихідна довжина масивів кумулятивних розподілів ec_ctx->eob_flag_cdf також різна.

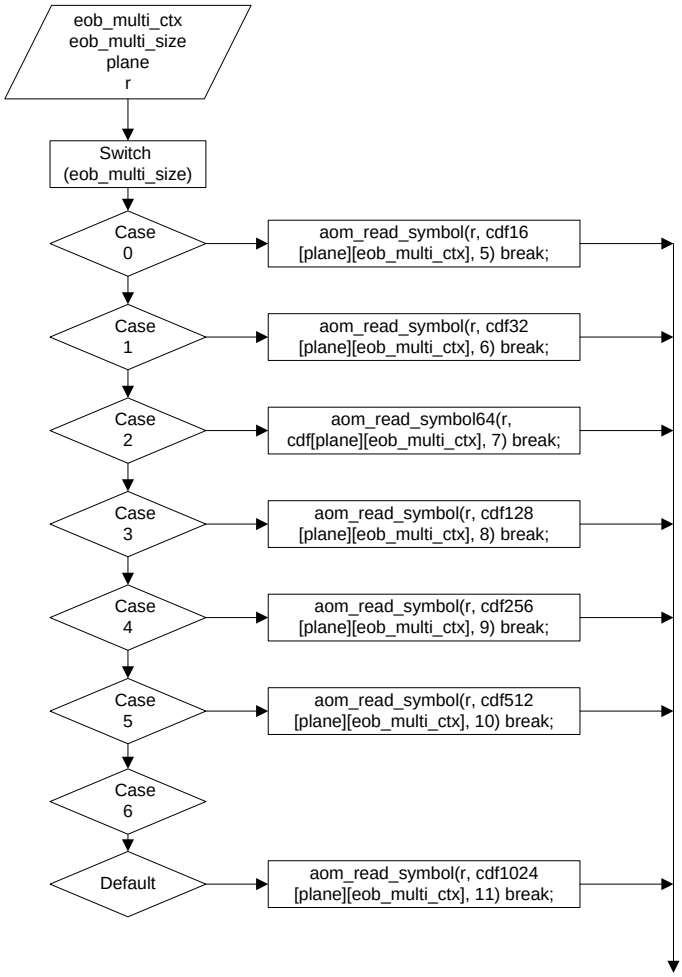

Рис. 2.10 Flow-chart діаграма декодування коефіцієнтів av1_decode_txb [55]

Висновки до розділу 2

Відеокодек – це програма, задача якої полягає у стисненні необроблених вхідних відеоданих у форматах RGB. YUV тощо, отриманих із відео-апаратури, відео-монтажу. Існує багато розробників відеокодеків, які пропонують різні схеми стиснення із використанням різних алгоритмів. Але схема роботи сучасних різних відеокодеків має багато спільного. Будь-який сучасний відео-кодек базується на між кадровому –передбаченні, що прибирає тимчасову однорідність, інтра-кодуванні – що прибирає просторову однорідність, дискретно-косинусному перетворенні та квантуванні, що прибирає математичну однорідність та ентропійному стисканні, що прибирає однорість даних, отриманих від усіх попередніх модулів.

Саме в ентропіному стисненні досягається найбільший відосток стиснення (пропорції вхідних / вихідних даних). Ентропійний кодек зазвичай є найбільш інтенсивним місцем у роботі програми-відеокодеку та аймає велику кількість циклів процесора. Етропійний кодек неможливо розпаралелити із-за ланцюжка даних бітового потоку, тому зазвичай модернізують існуючі алгоритми ентропійного стиснення – великі можливості у прискоренні надає застосування додатку SIMD-розширення , що є у сучасних центральних процесорах. В залежності від довжини векторного регістру досягається різний рівень прискорення. Поява сучасних центральних процесорів із додатком AVX-512, що має більшу довжину регістру та нові SIMD-команди надають можливість подальшого суттєвого пришвидшення роботи відеокодеків для рядового користувача, що ,зазвичай, проявляється у зменшенні навантаження на процесор під час перегляду відео, підвищення якості відео, зменшення обсягу передачі даних через інтернет.

РОЗДІЛ 3
ШЛЯХИ ПОКРАЩЕННЯ ЕНТРОПІЙНОГО КОДУВАННЯ НА БАЗІ РОЗШИРЕНОГО НАБОРУ ІНСТРУКЦІЙ SIMD

У п. 2.3 було розглянуто роботу та складено схему ентропійного декодування відео у AV1 за замовчування у репозиторії (скалярна версія на мові C). Нагадаємо, що ентропійне кодування складається з декількох частин: кодування відео, оновлення розподілу ймовірностей, ре нормалізація (див п. 2.3). В даному розділі розглянуто можливості оптимізації в частині декодування символу та частині оновлення розподілу ймовірностей із застосуванням розширеного набору інструкцій AVX-512.

Функції, в яких реалізували алгоритм ентропійного кодування відео є «гарячими точками» під час декодування відео (дивись Додадок Є. Результати профілювання виконавчого файлу aomdec). Так алгоритм декодування символу реалізовано у функції od_ec_decode_q15, алгоритм оновлення кумулятивних ймовірностей – у функції update_cdf, алгоритм ре нормалізації – od_ec_normalize.

На процесорі із підтримкою розширеного набору інструкцій SIMD AVX2 усі інші модулі відео кодування використовують цей набір (наприклад алгоритми косинусного перетворення, передбачення, компенсації руху). В той час як алгоритм ентропійного кодування не використовує наявний набір розширень інструкцій SIMD. Тому, зробимо припущення, що ентропійне кодування не використовує усі можливості процесорної системи. Спроби переписати дані функції із набором інструкцій SIMD SSE та SIMD AVX можна знайти у альтернативному репозиторії dav1d.

Використовується мова “Assembler” (представлення NASM, що підтримує інструкції AVX-512, інші представлення YASM не підтримує набір інструкцій SIMDAVX-512).

3.1. Функція декодування символу SIMD AVX-512

Другою із двох необхідних функцій для оптимізації, що викликається з головної функції декодування символу ентропіного кодування aom_read_symbol є є функція декодування символу. У п. 2.3 вже було показано flow-chart діаграму цієї частини. Порівняємо її із векторизованою версією на базі розширеного набору інструкцій SIMD AVX-512. Зазначимо, що ми використовуємо лише інструкції AVX-512 рівня споживання 1 (AVX2 = AVX-512 Light, див. п. 1.2), що виконуються процесором на тій же тактовій частоті, що і решта коду aomdec, що написана на AVX2.

Функції декодування символу використовує цикл із кількістю ітерацій рівний розміру вхідного масиву cdf M є [2..14]. У векторизованій версії функцій цикл замінюється інструкцію SIMD паралельних обчислень над усією довжиною вхідного масиву. Причому так як значення декодованого символу лежить до значення M, можна проводити паралельні обчислення над довжиною масиву 16, зайві значення доріжок векторного регістру, що мають номери більше M не зберігаються та не впливають на вибір декодованого символу.

Варто зазначити два моменти: по-перше для векторизації на процесорних системах із набором розширень інструкцій SIMD потрібно переписати даний відрізок коду наступним чином: рядок

$ v = ((uint16_t)(r>>8 * (icdf[++ret] >> EC_PROB_SHIFT) >> 1)

на

$ v = ((uint16_t)((r>>8)<<8) * ((uint16_t)(icdf[++ret] >> EC_PROB_SHIFT) << 7)) >> 16

Потім адаптуємо паралельні обчислення із вказаним рядком. Необхідність заміни покликана тим, що максимальний результат перемноження двох 16-бітних цілих чисел може — 32-бітне ціле число, яке у скалярній версії зберігається у регістрі загального призначення, а потім для використання старших бітів відбувається зсув на число 6. Оскільки інструкцій SSE/AVX/AVX-512 інструкції множення двох додатніх цілих 16-бітних чисел працюють без проміжного

розширення добутку до 32-біт, то у векторизованій версії використаємо одразу старші біти подвійного слова без зсуву. Для цього у наборі, але є інструкція vpmulhw, яка зберігає лише 16 старших біт добутку. Чим і викликана необхідність переписання даної ділянки коду. Це рядки 08-12 на рисунку 3.1.

По-друге, для ефективної векторизації потрібно додати масив (look-up), який буде замінювати операцію:

$ v += EC_MIN_PROB * (N – ret)*, де EC_MIN_PROB = 4*

Так як змінна ret – це лічильник у циклі і збільшується від 0 до M на кожній ітерації, то масив буде мати вигляд:

[(M – 0) *4, (M – 1) *4, … , (M – M)*4]

На Рисунку 3.1 автором показано діаграма реалізації алгоритму ентропійного кодування з одного боку - на базі SIMD AVX2, з іншого – на базі SIMD AVX-512. Причому використовується масив array [60, 56, 52 .. 0], з якого для функції декодування символу в залежності від вхідного N (у регістрі rdx) зчитуються номера починаючи із 16 — N (N → аргумент nsymbs, що передається у функцію od_ec_decode_q15 та update_cdf через регістр rdx).

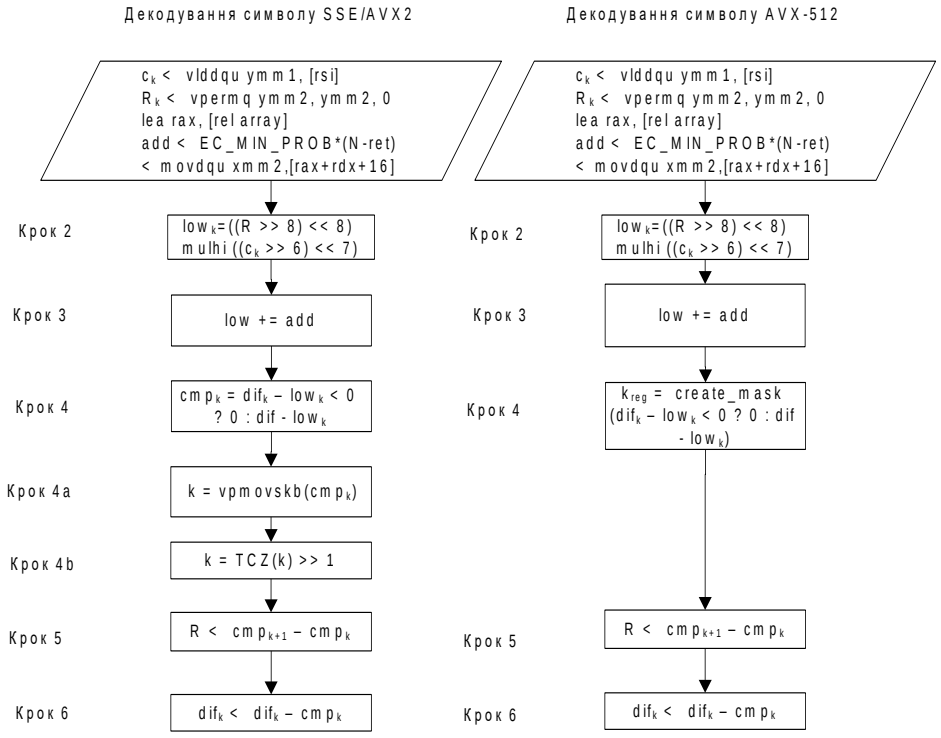

Рис. 3.1 Flow-chart діаграма реалізації алгоритму ентропійного кодування на базі SIMD AVX2 та SIMD AVX-512.

В основі алгоритму на базі SIMD лежить заміна циклу dowhile на паралельні обчислення над 16 елементами та використання спеціальних інструкцій із набору SIMD, що здійснюють замінюють дві-три логічні операції скалярних обчисленням.

Крок 1 – завантаження данних

Замість по-елементного присвоювання у версії на мові асемблер ми завантажуємо 64 біти структури із початку C-структури od_ec_dec (репозиторій libaom), яка вирівняна у пам'яті по межі 2 байт, тому ми вираховуємо відступ від початку структури до початку необхідних даних (необхідний відступ 32 байти), і завантажуємо одразу 8 байт у регістр: 1 подвійне слово od_ec_dec.diff, та 2 слова od_ec_dec.rng та od_ec_dec.cnt - (рядок 1 у Табл. Ґ Додатку Ґ). Потім за допомогою інструкцій vpermq (перемішування за маскою) та punpcklqdq (дублювання молодших бітів у старші), заповнюємо усі значення доріжок у векторному регістрі однаковими данними. В наборі інструкцій ми можемо скопіювати 1 елемент вектора лише у всі елементи його молодшої частини, для копіювання у всі елементи потрібно додаткова інструкція, яка копіює молодші біти xmm-регістру у старші. Але все одно розповсюдження скалярного значення по всьому вектору займе лише 2 такти, оскільки значення throughput 0.33 а кількість циклів на інструкцію 1, тому всі 3 пари інструкцій виконаються одночасно (див Розділ 1.2 Instruction Level Parallelism).

Заповнення значень доріжок векторного регістру однаковими даними виконується за допомогою інструкцій shuffle та permute (Рядки 14, 16 у Табл.Ґ Додатку Ґ), що приймають 3 аргументи: регістра-приймач, регістр-відправник та скалярне число, що виступає у якості маски, що вказує яку доріжку регістра-відправника розповсюдити по векторному регістру. Наприклад, для змінної od_ec_dec.rng створюємо вектор, де всі 8 елементів рівні rng - це shuffle 3-го слова у квад-слові із маскою 170 (тобто 2<<6 + 2<<4 + 2<<2 + 2) (Рядки 14 Додаток Ґ). Для dif це shuffle 1-го подвійного слова у квад-слові із маскою 0 (тобто 1«6 + 1«4 + 1«2 + 1) (Рядки 16 Додаток Ґ).

Lookup-массив rate, що було створено на стеку, завантажується 2 інструкціями: за допомогою інструкції lea – взяття (завантаження) його ефективної адреси на стеку у регістр загального призначення та завантаження із цієї адреси 16 елементів із місця, що зміщено на nsymbs (регістр rdx).

Крок 2

На другому кроці здійснюється знаходження добутку на базі вхідних змінних R та c_k. Версії для AVX2 та AVX-512 тут не відрізняються. Замість простого зсуву на 8 біт вліво діапазону R, та множення на кумулятивну ймовірність розподілу c_k (також із зсувом вправо на 6 розрядів), у векторизованій версії потрібно взяти верхні 16 біт добутку (виконується однією інструкцією SIMD mulhi – 5 тактів, пропускна здатність 0.5) між верхніми 8 біт R (зсув вправо на 8 та зсув вліво на 8) та кумулятивною ймовірністю, що зсунута на 7 вліво (рядки 19-23 Додаток Ґ). Це вимагає 4 прості інструкції (інструкції без машинного розчеплення на μ-ор (дивись Розділ 1.2). Враховуючи ILP такі 2 пари інструкцій виконаються паралельно за 2 такти оскільки є вільні порти, та немає залежності данних (2 пари двох послідовних інструкцій 1 такти, пропускна здатність 0.5).

Крок 4

На четвертому кроці здійснюється знаходження декодованого символу k всієї функції ентропійного декодування. У скалярній версії це вібувається у циклі dowhile простим перебором: ітеративно порівнюються два без знакові числа dif та low та зберігається номер останньої ітерації k, яка задовольняє умові циклу, також при виході із циклу в R зберігається останнє значення різниці up та low, де up – значення low із попередньої ітерації. У SIMD версіях це реалізовано без циклу по іншому: в AVX2 Інструкція SIMD насиченого віднімання vpsubuswa, b – замінює ітеративні порівняння «чи більше» та присвоєнні нуля різниці, якщо вона відємна (1,0.5 + 3,1). Визначення декодованого символу k є кількістю нульових елементів (тобто тих, для яких різниця dif_k-low<=0). Для AVX2 потрібні 2 додаткові кроки. На кроці 4a AVX2 після порівняння додатоково необхідна інструкція взяття старших бітів доріжок vpmovskb (2 такти, пропускна здатність 1) , а на кроці 4b – скалярна інструкція для підрахунку серії нульових бітів у кінці значення k

(trailingzerocount – 3 такти, пропускна здатність 1). SIMD інструкція vpmovskb створює із значення векторного регістру скалярне 32 бітне число у регістрі загального призначення, де кожен біт відповідає старшому біту кожній відповідній доріжці-байту векторного регістра. В AVX-512 цей крок можна зробити за одну інструкцію vpcmpuwa, b, imm8 (3 такти, пропускна здатність 1), що порівнює упаковані беззнакові слова в a і b на основі операнда imm8, і зберігає результат як слово у векторному регістрі-масці спеціального призначення k. Реалізації функцій декодування символу на базі SIMD AVX2 з одного боку та SIMD AVX-512 з іншого представлено автором у Таблиці 3.1.

Таблиця 3.1

Порівняння рівнозначних фрагментів коду для вирішення Проблеми 1 ентропійного кодування із застосування розширення набору інструкцій AVX2 та AVX-512

Line	AVX2	clocks	thoughput	AVX-512	clocks	thoughput
134	vpsubusw ymm1, ymm3	1	0.33	vpcmpu* k7, ymm1, ymm3, 2*	3	1
135	vpxor ymm2, ymm2	1	0.33	vpcmpeqw ymm2, ymm2, ymm2	1	0.5
136	vpcmpeqw ymm1, ymm2	1	0.5	vmovdqu16 ymm1 {k7}{z}, ymm2*	3	0.5
137	vpmovmskb rax, xmm1	2	1	kmovq rax, k7	1	1
138	vpcmpeqw ymm2, ymm2, ymm2	1	0.5	tzcnt rax, rax	3	1
139	tzcnt rax, rax	3	1			
140	shr rax, 1	1	0.5			

Джерело: дані автора (Повний лістинг коду у Додатку А)

** - інструкції з набору розширень AVX-512*

З данних у Таблиці 3.1, можна припустити, що крок 3 у версії на базі AVX-512 ефективніший: версія AVX2 займе 8 тактів, AVX-512 – 3 такти (враховуючи поправку на паралелізм на рівні інструкцій). Значення k-регістру можливе для повторного використання у функції оновлення ймовірностей (що також збереже такти процесорний час та зменшить кількість інструкцій ентропійного кодування).

Крок 5

В SIMD AVX-2 та SSE версіях неможливо одразу зберегти різницю між двома сусідніми доріжками векторного регістру, порядкові номери яких змінні під час виконання програми. Тому єдиним способом є: 1) збереження значення векторного

регістру на стеку, 2) по-елементне завантаження необхідних сусідніх елементів зі стеку 3) та їх скалярне віднімання (між регістрами загального призначення).

Розрахунок часу виконання функції декодування символу

Для прорахунку вимірюємо відносну кількість тактів функції декодування ентропійного кодеру на мові Assembler із використанням інструкцій набору SIMD та на мові С (доступного за замовчуванням, Таблиця 3.2. дизасемблер код на мові С). Лістинг коду на мові Assembler отримаємо через спеціальні директиви під час компіляції , який генерує C++ компілятор (Visual C або g++ у версії Release – оптимізований компілятором код, не Debug).

Для прорахунку враховуємо паралелізм на рівні інструкцій та суперскалярний суперконвеєр — для цього достатньо порахувати одночасно сумарну кількість циклів для інструкцій, врахувавши інструкції, які можуть виконатись одночасно (відносна пропускна здатність 0.2, 0.25, 0.33, 0.5). Нагадаємо, відносна пропускна здатність (reciprocal throughput) — це кількість певних однакових інструкцій (типу дії та регістрів), які можуть виконуватись одночасно на АЛП, наприклад операція перемноження двох xmm-регістрів має значення reciprocal throughput 0.5, та cycles 1 – що означає, що дві (1/0.5) такі операції можуть виконатись одночасно за 1 такт, проте команди не повинні залежати від результату одна одної за 1 такт (Див. Розділ 1.3).

Якщо подивитись на оптимізований компілятором лістинг асемблерну із об'єктного файл Release версії та порівняти його ефективність з нашою SIMD-версією власноруч написаного коду на мові Assembler то кількість тактів ентропійного декодування – менша. Кількість тактів та залишкова пропускну здатність для кожної інструкції береться із специфікації Agner Fog Instuction Tables для мікроархітектури Icelake/TigerLake [20, p.354] та із офіційного сайту компанії Intel [22]. Лістинг приведено у Таблиці Ґ Додатку Ґ.

Як бачимо із Таблиці 3.2, компілятор не тільки не здатен згенерувати код під набір розширень інструкцій SIMD, але й генерує не найкращий код на мові Ассемблер. Наприклад, використовує менш швидкі зсуви shr/shl замість більш швидких shrx/shlx для мікроархітектури Skylake-X. Компілятор генерує багато

непотрібного коду – що дуже важливо в таких ділянках коду, наприклад, у цілях безпеки комплітор зберігає багато змінних на стек при виклиці функції. Однак, при використанні набору розширень інструкцій SIMD нам непотрібно класти на стек, і потім знімати значення 3 регістрів, що також зберігає нам ~12 тактів, оскільки ми не чіпаємо волатильні регістри, а працюємо переважно із неволатильними xmm-регістрами.

В цілому версія ентропійного декодування на базі SIMD AVX-512 має менше інструкцій ніж скалярна функція (23 такти проти 45). Але враховуючи висоту конвеєра п’ять, 45 прогнозованих нами тактів процесора ~ рівне 10 реальним тактам процесора. Враховуючи те, що кількість тактів процесора при виконанні секції коду скалярної версії буде збільшуватись із кожною ітерацією циклу (база скалярної версії). Тому можна припустити, що SIMD версія буде швидше коли кількість ітерацій, необхідна для декодування символу у скарярній версії буде більшою за 23*5, тобто при значенні декодованого символу k є [4..12] кількість реальних тактів процесора 45*4 > 23*5. Для пітвердження чи спростуванні цього припущення у п.3.3 здійснюється тестування результатів ентропійного кодування.

3.2. Функція оновлення ймовірностей – SIMD AVX-512

Другою із двох необхідних функцій для оптимізації, що викликається з головної функції декодування символу ентропіного кодування aom_read_symbol є функція оновлення ймовірностей update_cdf. Поглянемо на код функції (Рис.3.2).

Крок 1

На кроці 1 завантажуються дані у потрібному нам форматі – параметр зсуву повинен знаходитись у регістрі векторного призначення – оскільки зсув на змінну доріжок векторного регістру можливий лише якщо вона перебуває у векторному регістрі (vpsllwymm, ymm, xmm). Тому на кроці 1 за допомогою інструкції movd у векторний регістр завантажується слово із значенням nsymbs із регістру загального призначення rdx (вхідний параметр функції згідно callingconvention).

Також це слово потрібно завантажити із регістру загального призначення rdxy регістр k_1спеціального призначення SIMD AVX-512. Обидва таких завантаження витрачають по 1 такту процессора та здайснюються паралельно (кількість тактів 1, пропускна здатність 0.5).

Автором вдосконалено функцію оновлення ймовірностей (файлі msac.c, віддаленого репозиторій videolan dav1d) на базі AVX2 та AVX-512, що представлено на рисунку 3.2.

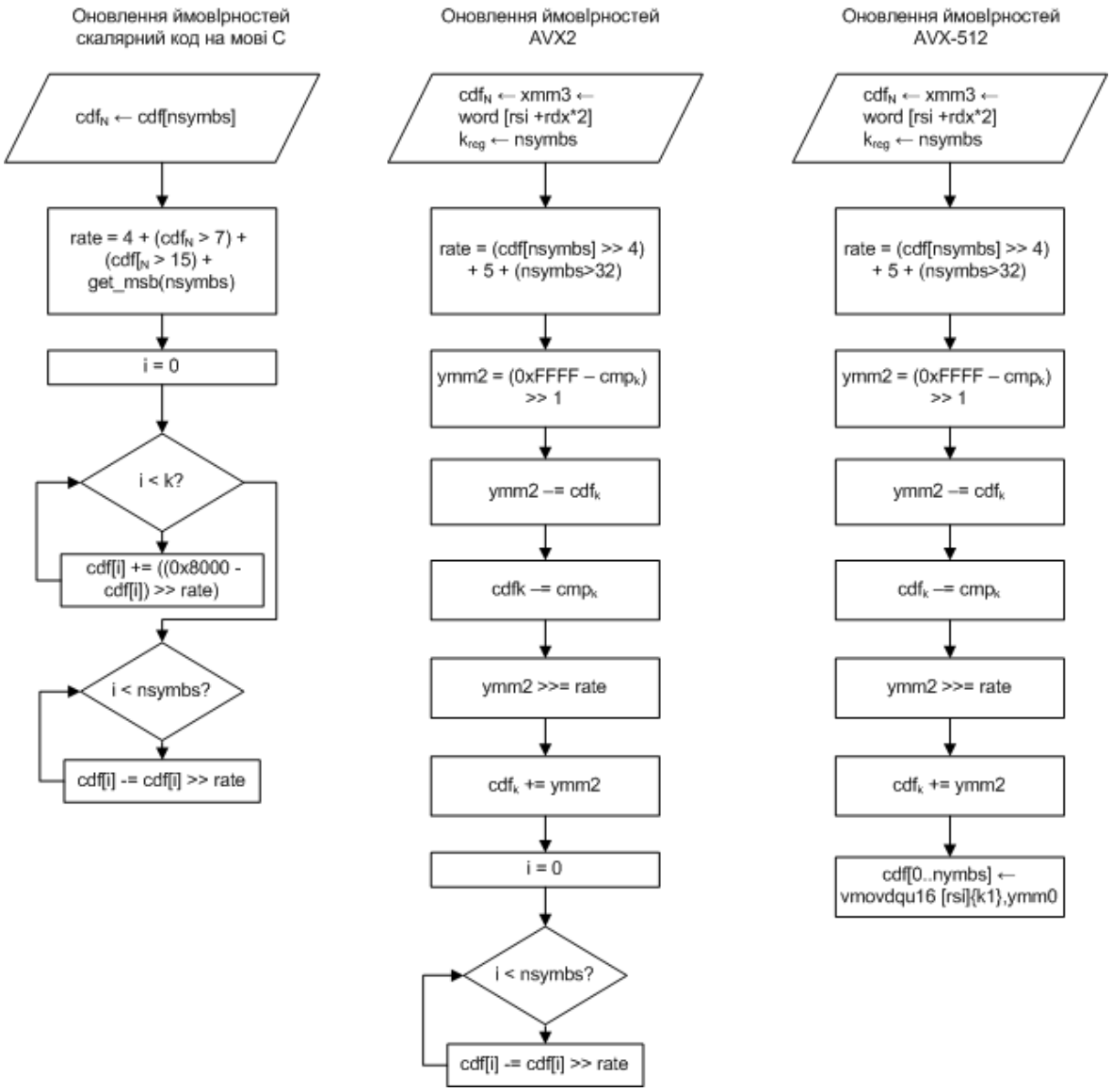

Рис. 3.2 Flow-chart діаграма реалізації алгоритму ентропійного кодування на базі SIMD AVX2 та SIMD AVX-512.

Крок 2

На кроці 2 вираховується значення змінної rate т.к. як і в скалярній версії коду алгоритму, але на мові Assembler (див. Додаток Д – рядок 48-52). Дві інструкції

зсувів праворуч shrx здійснюються паралельно, після цього виконується скалярна інструкція додавання add.

Крок 3

На кроці 3 створюєсо значення 2^{16}-1 (усі біти - одиниці) у всіх доріжках-словах векторного регістру ymm2. Після чого рахуємо середнє значення між створеним значенням векторним регістру значення порівнянь із векторного регістру, що збереглося із функції декодування символу (може бути або 0, або 2^{16}-1) за допомогою SIMD-інструкції усереднення vpavg. Можливими результатами цих двох операцій будуть числа (2^{16}-1 + 2^{16}-1)/2 або (2^{16}-1 + 0)/2, тобто 2^{16}-1 та 2^{15} (0x8000). SIMD-інструкція віднімання двох без знакових слів vpsubuw (1 такт із пропускною здатністю 0.5) та SIMD-інструкція знаходження середнього vpavgb (1 такт із пропускною здатністю 0.5) відбуваються одна за одною та займуть 2 такти процессорного часу. (Рядки 55-57 Додаток Д).

Крок 4

Значення регістру ymm2 зменшуємо на значення векторного регістру із значеннями ймовірностей кумулятивного розподілу із векторного регістру ymm0, що отримано на попередньому кроці за допомогою SIMD інструкції віднімання vpsubw. Результатом цієї операції буде значення 2^{16}-1 – cdf[i] та 0x8000 – cdf[i].

Крок 5

Значення регістру із ймовірностями ymm0 зменшуємо на значення векторного регістру із значенняи порівнянь із векторного регістру, що збереглося із функції декодування символу (може бути або 0, або 2^{16}-1) за допомогою SIMD інструкції віднімання зі знаком vpsubw. Можливими результатами цих такої операцій будуть числа cdf[i] - 2^{16}-1 або cdf[i].

Крок 6

Значення регістру ymm2 зсовуємо праворуч на на ratерозрядів, що отримано на попередньому кроціта знаходиться в регістрі xmm3, за допомогою SIMD інструкції віднімання vpsraw. Результатом цієї операції є значення (2^{16}-1 – cdf[i]) >>rate та (0x8000 – cdf[i]) >>rate.

Крок 7

На сьомому кроці до значення слів-доріжок векторного регістру ymm0 із кроку 5 додаємо до значення слів-доріжок векторного регістру ymm2 із кроку 6 за допомогою SIMD інструкції vpaddw (1 такт, пропускна здатність 0.5). Можливими значення результату є cdf[i] - 2^{16} + (2^{16}-1 – cdf[i]) >>rate або cdf[i] + (0x8000 – cdf[i]) >>rate. Очевидно, що результатом віднімання від слова зі знаком числа 2^{16}-1 тотожно додаванню плюс одиниці. У представленні 16-бітних чисел зі знаком число -1 є словом, де всі біти – одиниці, тобто 2^{16}-1. Тому результатом результів на кроці 7 є векторний регістр заповнений доріжками, які можуть приймати всього 2 значення – або cdf[i] - cdf[i] >>rate або cdf[i] + (0x8000 – cdf[i]) >>rate. Це коректний результат, що дорівнює таким же значенням із скалярного алгоритму (Див. Рис. 3.1).

Крок 8

Кроки 1-7 є однаковими як для SIMD AVX2, так і для SIMD AVX-512. Векторна версія коду функції оновлення ймовірностей працює із одразу 16 елементами, що може перевищувати необхідну кількість елементів масиву кумулятивного розподілу ймовірностей cdf. Під час завантаження ми можемо брати зайві елементи із памяті за межами масиву, оскільки декодовані значення символу все одно будуть знаходитись до значення nsymbs і це не вплине на коректність декодування символу чи процес оновлення коефіцієнтів. Проте під час збереження ми не можемо зберігати слова-доріжки усього вектору у пам'ять, оскільки це вийде за межі ділянки памяті масиву, що ми оновлюємо і перезапише наступний масив, який використовується у інших ситуаціях (у репозиторії libaom масиви зберігаються у пам'яті щільно — з вирівнюванням 2 байти, тому не можемо записати увесь вектор для меншого розміру вхідного масиву — оскільки масив запишеться на елемент наступного масиву, який використовується для декодування інших символів із іншим розміром і типом коефіцієнтів).

На кроці 8 відбувається запис змінної кількості слів-доріжок векторного регістру у пам'ять. У скалярній версії таке зберігання здійснюється по елементно ітеративно у циклі (за декілька скалярних інструкцій зберігання) — проте такий код буде містити багата операцій звернення до пам'яті. Для цього немає наявних

інструкцій розширеного набору SIMD AVX2, але такі інструкції є у наборі розширень інструкцій AVX-512, які до працюють із 256-бітними векторними регістрами ymm. Варіант коду із використанням розширеного набору інструкцій показано у Додатку Ґ.. В розширених наборах інструкцій SIMD AVX2 це також єдиний спосіб, оскільки подібних SIMD-інструкцій немає. Однак у розширеному наборі інструкцій SIMD AVX-512 такі інструкції присутні, й належать до класу енергоспоживання L1 (AVX2/AVX-512 Light), що не змінює тактову частоту процесорної системи.

Проблема, яка залишається для коду із застосуванням набору інструкцій SIMD AVX2 є зберігання оновлених коефіцієнтів — для кожного різного вхідного розміру масиву ймовірностей ми повинні зберігати рівнозначне число 16-бітних елементів векторного регістру. Якщо аргумент функції є числом кратним 8, та кратним 8 мінус 1 (для 7, 8, 15, 16), то такий код буде працювати — але для інших буде перезаписувати сусідні масиви і програма не буде працювати коректно. Застосовувати цикл із поелементим завантаженням у пам'ять неефективно. Тому це можна виправити застосувавши нове розширення набору інструкції AVX-512.

Проблема полягає у збереженні вибіркових елементів векторного регістру у память за мінімальну кількість тактів та інструкцій. У літературі така проблема має назву “проблеми лівостороннього пакування”. Маємо вхідний і вихідний масив, але потрібно записати лише ті елементи, які відповідають певні умові. Ілюстрація проблеми лівої упаковки з векторного регістру показана на Рис. 3.3:

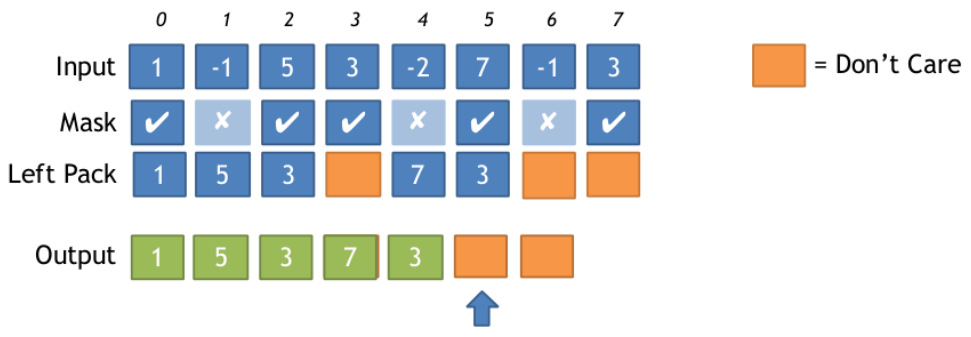

Рис. 3.3 Проблема 1 — проблема “лівостороннього пакування” [58]

Реалізація алогоритму через скалярні інструкцій вимагає 4 зайві звернення до памяті на запис/зчитування. Найбільш ефективна реалізація із використанням набору інструкцій SIMD AVX2 буде із використанням короткого циклу, до максимального значення (nsymbs поділити на 2), який буде переставляти подвійні слова-доріжки векторного регістру по та зберігати їх – тобто кількість звернень пропорційно кількості nsymbs. Оскільки в AVX2 інструкції перестановки доріжок у векторному регістрі ymm обмежена лише 32 бітними словами-доріжками. Тому в AVX2 буде також цикл із зайвими додатковим тактами на інструкції jump less (142 рядок), перестановка елементів (137-138 рядок).

Із застосуванням набору інструкцій AVX-512 вирішення цієї проблеми значно простіше, й такий набір інструкцій виконується за значно меншу кількість тактів. Для збереження змінного числа слів-доріжок векторного регістру слід створити маску-слово у k регістрі призначеного для масок в AVX-512, в якому порядковий номер біта, що рівний одиниці буде відповідати порядковому номеру доріжки-слова векторного регістру, що буде збережено у память. Значення в k регістрі створюємо за допомогою простого арифметичного виразу 1<<nsymbs-1, тобто робимо одиничне бітове поле на інтервалі [0..nsymbs-1] (рядки 61-65 Додаток Д).

Розрахунок продуктивності функції оновлення ймовірностей

Прорахуємо його продуктивність у порівнянні із дизасемблером оптимізованої скалярної версії коду за замовчуванням у репозиторії aom (Додаток ДТаблиця Д). У Таблиці 3.2 автором наведено фрагмент лістинг коду для вирішення проблеми лівостороннього пакування на базі SIMD.

Таблиця 3.2

Порівняння фрагментів коду для вирішення Проблеми 2 ентропійного кодування на базі розширених наборів інструкцій AVX2 та AVX-512

Line	AVX2	clocks	thoughput	AVX-512	clocks	thoughput
134	vpcmpeqw ymm2, ymm2; -1	3	1	mov r11, 1	1	0.25
135	vpxor ymm3, ymm3 ; 0	1	0.33	shlx r11, r11, r8	1	0.5
136	do_while_1:			sub r11, 1	1	0.25
137	vpermd ymm4, ymm0, ymm3	3	1	kmovd k1d, r11//*	2	1
138	vpsubw ymm3, ymm2; +1	1	0.33	vmovdqu16 [rdx] {k1}, ymm0//*	5	1

Таблиця 3.2 (продовження)

Порівняння фрагментів коду для вирішення Проблеми 2 ентропійного кодування на базі розширених наборів інструкцій AVX2 та AVX-512

Line	AVX2	clocks	thoughput	AVX-512	clocks	thoughput
139	pextrd [rdx+r13*2],ymm3,0	3	0.5	mov [rdx + r8*2], r11w	2	0.5
140	add r13, 2	1	0.25			
141	cmp r13, r8	1	0.25			
142	jl loop	2	0.5			
143	mov [rdx + r8*2], r11w	2	1			
	TOTAL AVX2 (incl. ILP & throuput)	18	-	TOTAL AVX512 (incl. ILP & throuput)	12	-

* - *інструкції з набору розширень AVX-512*

Із Таблиці 3.2 видно, що векторизувати і написати код універсальної функції на мові “Assembler” із використанням набору інструкцій SIMD AVX2 складно і неефективно у порівнянні із SIMD AVX-512, а час виконання коду на базі набору інструкцій AVX2 значно більше у порівнянні як із SIMD AVX-512, так із скалярною версію коду за замовчуванням із репозиторію aom. Мінімальна кількість виконання AVX2 версії — 18 тактів (хоча медіанне значення throughput інструкцій становить 0.25-0.5, проте всі інструкції взаємозалежні), тіло циклу займає 12 тактів, тобто в гіршому випадку (при розмірі вхідному масиві cdf – аргумента функції 16 і декодованому символу, що дорівнює довжині масива 16) час виконання становитиме 192 такти, проте таких випадків не багато. Порівняно до аналогічного фрагменту коду із AVX-512, час виконання якого складає 12 тактів незалежно від вхідних параметрів функції, це у 19 разів більше. У порівнянні із скалярною версією, де цикл із змінною кількістю ітерацій збільшить кількість енергії витраченої у пристрої передбачення переходів та штрафних циклів за неправильне передбачення (скидання усіх значень, що вже завантажені та попередньо обраховані у суперскалярному суперконвеєрі) (див. Розділ 1.2). Це пояснюється тим що вхідний закодований для декодера відео файл може містити довільні дані — тому відсутня закономірність для передбачення значень

коефіцієнтів косинусного пертворення макроболоків і відповідно значень закодованих символів ентропійним кодером (див. Розділ 2.2).

Фінальний вигляд функції ентропійного кодування наведено у Додатку Д. Код, що міститься у Додатку Д є продовженням функції декодування символу od_ec_decode_q15, яку переписано у п.3.1 власноруч на мові Assembler із використанням розширення набору інструкцій AVX2 (Додаток Ґ). Це покликано тим, що частина значень, які вже обчислені і містяться у векторних регістрах як результат роботи od_ec_decode_q15, недоцільно зберігати у пам’ять і потім знову зчитувати з пам’яті у новій функції — тому доцільно оновити вхідні масиви кумулятивного розподілу у продовження функції od_ec_decode_q15 версії із використанням розширення набору інструкцій SIMD, і потім ренормалізувати коефіцієнти та повернути значення декодованого символу (Додаток Ґ Таблиця Ґ).

Як бачимо кількість інструкцій у версії за замовчуванням більша у 2 рази проте більша довжина і ширина конвеєра дозволяє і такому коду працювати порівняно швидко.

3.3. Вимірювання продуктивності коду

Для вимірювання продуктивності коду використовується профайлер perf. Дослідження здійснено на процесорній системі Intel Skylake-X з підтримкою розширеного набору інструкцій SIMD AVX-512.

Вимірювання ефективності здійснюється за допомогою:

- заміру часу роботи файлу, що виконується aomdec (у циклах процесора) та показники роботи лічильників продуктивності (hardwareperformancecounters) кількість вкрадених циклів pipeline_stalls, stalled_cycles_frondend, stalled_cycles_backend.

- Юніт-тестування (тест на швидкість) продуктивності функції od_ec_decode_cdf_q15 на мові C, та із використання набору інструкцій SIMD – AVX-512.

За замовчуванням у Google Cloud Compute Engine не має доступу до таких лічильників, що можна перевірити за допомогою команди sudo dmesg | grep PMU. Результат – повідомлення «Performance Events: unsupported p6 CPU model 85 no PMU driver, software events only» (Software Events та PMCs показані на Рис. 3.4).

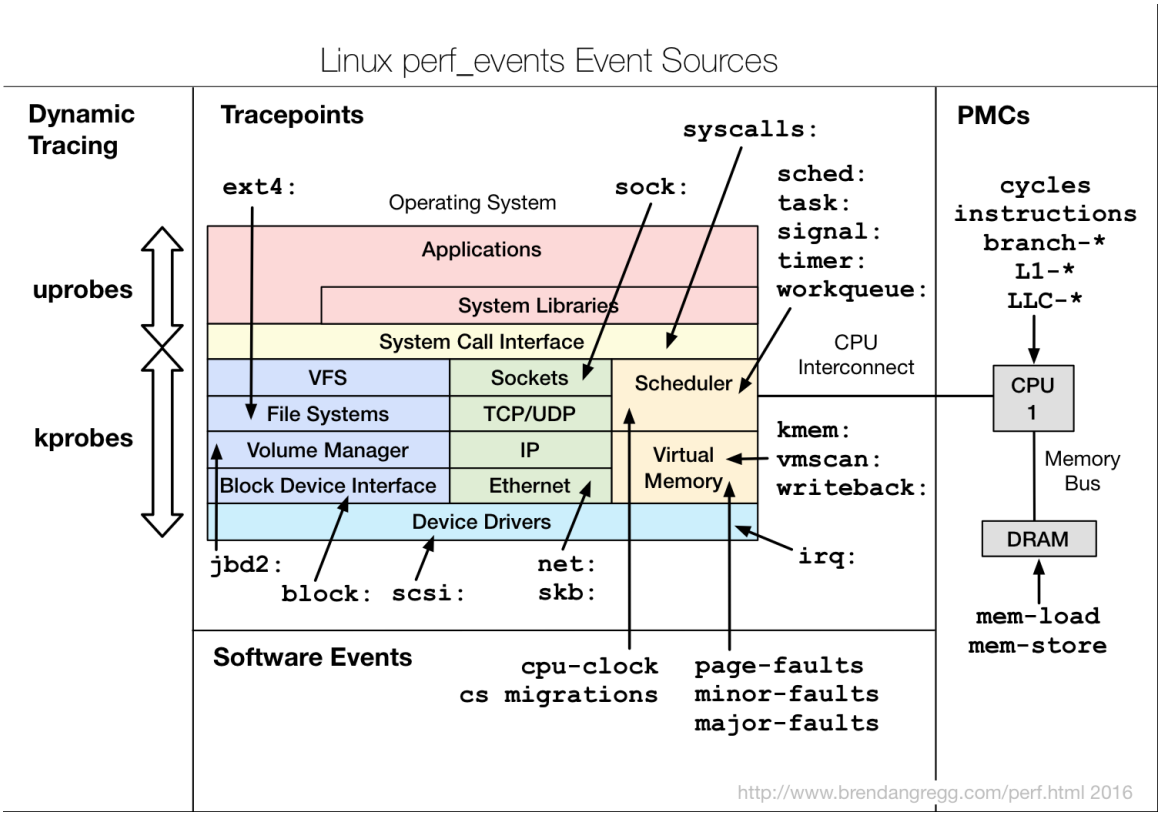

Рис. 3.4 Програмні та апаратні лічильники продуктивності (CPU/Linux)[59]

Тому єдиною метрикою вимірювання ефективності роботи у нашому випадку є час роботи програми із скалярними інструкціями та із інструкціями SIMD AVX-512. Обрано підбірку файлів та заміряно час виконання, а також середнє значення декодованого символу.

Виміримо продуктивність набору інструкцій за допомогою доступних лічильників продуктивності процесорної системи - програмних (доступні у профайлері perf). Ці лічильники зареєстровані в операційній системі під час інсталяції програмного забезпечення, що дозволяє будь-кому з відповідними дозволами переглядати їх. Для цього у ОС Windows доступна функція QueryPerformanceCounter (profileapi.h у Win32 API), яка зчитує значення апаратного лічильника продуктивності, що є часовою міткою високої точності

(роздільної здатності менше 1 мкс) і використаємо її для заміру часового інтервалу виконнання [60]. Для цього в репозиторії aom вже зроблена структура *aom_usec_timer* та відповідні функції. Для цього треба вставити початок і кінець до і після набору інструкцій який ми хочемо виміряти, та обчислити різницю (Рис 3.5).

```
#ifdefined(_WIN32)
structaom_usec_timer {
LARGE_INTEGER begin, end;
};
staticINLINEvoid aom_usec_timer_start(structaom_usec_timer *t) {
QueryPerformanceCounter(&t->begin);
}
staticINLINEvoid aom_usec_timer_mark(structaom_usec_timer *t) {
QueryPerformanceCounter(&t->end);
}
staticINLINEint64_t aom_usec_timer_elapsed(structaom_usec_timer *t) {
LARGE_INTEGER freq, diff;
diff.QuadPart = t->end.QuadPart – t→begin.QuadPart;
QueryPerformanceFrequency(&freq);
return diff.QuadPart * 1000000 / freq.QuadPart;}
#endif //_WIN32
```

Рис. 3.5 Вимірювання продуктивності коду за допомогою апаратних лічильників продуктивності (циклів чи мікросекунд)

Для виміру такого невеликого набору інструкцій запустимо код функції декодування символу та оновлення кумулятивних ймовірностей (поєднанні для ефективного використання регістрів, див. Розділ 2.3) 10 млн. разів попередньо створивши копію об'єкта декодеру od_ec_dec та масивів кумулятивного розподілу (оскільки функція змінює його). На вхід у функцію декодування блоку перед вибором параметрів для функції ентропійного декодування передаються 2 параметри: розмір кумулятивного масиву ймовірностей – він же дорівнює значенню розміру блоку, та сам масив. В залежності від цих двох параметрів і побудуємо таблицю результатів. Час виконання оптимізованого коду та скалярного коду – співвідношення продуктивності представлено автором в Табл. 3.3- 3.4.

Таблиця 3.3

Прискорення функції ентропійного декодування на AVX-512 відносно C в залежності від значень величини макроблоку та декодованого символу

	Значення декодованого символу											
Розмір блоку	0	1	2	3	4	5	6	7	8	9	10	11
5	1.92	1.50	1.46	1.37	1.27	-	-	-	-	-	-	-
6	1.80	1.07	1.21	1.04	1.22	1.01	-	-	-	-	-	-
7	1.66	1.33	1.29	1.19	1.11	1.05	0.96	-	-	-	-	-
8	1.57	1.32	1.20	1.14	1.08	0.99	0.93	0.91	-	-	-	-
9	1.97	1.27	1.15	1.04	1.00	0.94	0.88	0.87	1.08	-	-	-
10	1.95	1.30	1.08	1.00	0.97	0.85	0.84	0.96	1.02	0.97	-	-
11+	1.75	1.62	1.43	1.00	0.99	0.98	0.85	0.96	0.99	0.92	0.97	-

Таблиця 3.4

Прискорення функції ентропійного декодування AVX-512 відносно AVX2 в залежності від значень величини макроблоку та декодованого символу

	Значення декодованого символу											
Розмір блоку	0	1	2	3	4	5	6	7	8	9	10	11
5	1.04	1.12	1.13	1.05	1.04	-	-	-	-	-	-	-
6	1.03	1.03	1.21	1.04	1.05	1.04	-	-	-	-	-	-
7	1.03	1.03	1.02	1.03	1.03	1.04	1.05	-	-	-	-	-
8	1.01	1.03	1.02	1.02	1.02	1.02	1.02	1.03	-	-	-	-
9	1.07	1.04	1.05	1.04	1.04	1.04	1.05	1.05	1.07	-	-	-
10	1.01	1.02	1.01	1.01	1.02	1.02	1.01	1.02	1.01	1.04	-	-
11+	1.06	1.00	1.04	1.01	1.00	1.00	1.01	1.01	1.01	1.01	1.03	-

Результати тестування із Табл. 3.4 свідчать про значно менший час декодування комбінацій «Розмір блоку – значення декодованого символу», що часто зустрічаються, у версії ентропійного декодування символу у версії на базі SIMD AVX-512 у порівнянні із скалярною версією. Наприклад, при найменших розмірах блоків версія на базі SIMD AVX-512 – продуктивніша при всіх можливих значеннях декодованого символу. Однак чим більше значення декодованого символу – тим менше прискорення версії на базі SIMD AVX-512. Але враховуючи той факт, що середньозважене значення декодованого символу складає 5 (див.

Табл. 3.5), а медіанний розмір блоку, що зустрічається – 6, то навіть в нашому випадку не має сенсу спеціалізації функції ентропійного кодування в залежності від розміру блоку (значення декодованого символу прогнозувати неможливо), оскільки у більшості випадків функція ентропійного декодування на базі SIMD AVX-512 буде працювати швиде. Але в деяких випадках для ентропійного декодування спеціалізація можлива використання скалярну версію для декодування символу, де де значення eob_multi_size (=nsymbs) [0..3], а для декодування символу, де значення eob_multi_size (=nsymbs) [4..11] – версію коду із використання набору розширення інструкцій SIMD AVX-512.

Також результати тестування свідчать про значне прискорення векторизованої частини агрегованої функції, що відповідає за функції update_cdf, яка після кожного ентропійного декодування симоволу, згідно з алгоритмом, оновлює масив кумулятивного розподілу вірогідностей — це адаптивне кодування що описується у Розділі 2.3. Цим пожна пояснити відносне покращення продуктивності коду на малих значеннях декодованого символу, оскільки більше вірогідностей оновлюється поза основним циклом.

Результати повних замірів, що представлені автором у Таблиці 3.5, наочно підтверджують прискорення ентропійного декодування на базі SIMD AVX-512 відносно декодування на базі звичайних інструкцій, як і можливість спеціалізації функцій ентропійного декодування на базі SIMD та C для окремих випадків. Коливання результатів пояснюється наявністю процесів операційної системи та їх впливом на процес декодування, а також перегріванням процесору, що запускає механізм троттлінгу (скидання тактової частоти).

Із Таблиці 3.5 видно, що версія написана на мові Assembler із застосуванням розширеного набору інструкцій SIMD приблизно на 10% швидше за скалярну версію коду у кращому випадку, та на 7% гірше у гіршому випадку. Хоча як ми описували у Розділі 1.3 такі інструкції відносяться до групи ліцензій L1, що виконуються процесорній системі із зниженням частоти на 20-30%.

Таблиця 3.5

Час виконання ентропійного декодування на деяких відео файлах формату AV1

Назва файлу	Середнє значення символу	AVX-512 (спроба 1)	AVX-512 (спроба 2)	AVX-512 (спроба 3)	AVX2 (спроба 1)	AVX2 (спроба 2)	AVX2 (спроба 3)	C99 (спроба 1)	C99 (спроба 2)	C99 (спроба 3)	AVX-512 (середнє)	AVX2 (середнє)	C99 (середнє)
Big_Buck_Bunny_1080_10s_30MB.webm	5.9	20.60	20.72	20.74	18.80	18.61	18.63	18.44	18.74	18.45	20.69	18.68	18.54
CityHall_1920x1080.webm	4.76	57.03	56.92	57.48	56.81	57.17	57.39	56.65	55.96	56.18	57.14	57.12	56.26
CityHall_3840x2160.webm	4.75	19.08	19.02	18.95	18.70	19.10	19.06	19.06	19.10	19.06	19.02	18.95	19.07
Sparks (1000 кадрів)	3.89	94.99	93.41	93.88	91.62	90.19	92.23	91.27	92.79	93.78	94.09	91.35	92.61
Chimera-AV1-10bit-1920x1080-6191kbps.ivf (300 кадрів)	5.28	55.57	54.36	55.03	53.72	54.12	53.73	52.64	51.76	54.92	54.99	53.86	53.11

Джерело: відео: Amazon, Netflix, Elecard

Але врахуємо той факт, що застосування AVX2 у всіх інших частинах кодеку AV1 використовується за замовчуванням на усіх процесорних системах на базі Intel, і зниження частоти виправдано значним прискорення за рахунок ширини векторного регістру у паралельних обчислень SIMD.Тому навіть скалярний код ентропійного кодування буде працювати на тій же частоті, що й AVX-512 типу ліцензії L1, оскільки ядро заблоковано і його блокування постійно оновлюється при виконанні інструкцій SIMD із інших викликів функцій відеокодеку.

Для нових мультипроцесорних систем Intel Icelake та Intel TigerLake слід використовувати розширений набір інструкцій SIMD типу AVX-512, при цьому розширена довжина 512 біт векторного регістра не використовується, необхідним тільки є використання логіки інструкцій з цього набору інструкцій з AVX-512, що дозволить декодувати швидше за менше інструкцій.

3.4. Інтеграція AVX-512 коду у репозиторій libaom

Для роботи із інструкціями з набору SIMD AVX-512 використовується хмарні обчислення на базі одноядерна процесорна система серверного процесора мікроархітектури Skylake-X (Google Cloud Compute Engine), що працює під

управлінням операційної системи Linux (ОС Ubuntu 20.04). Пересвідчитись у підтримці бажаного типу інструкцій – за допомогою команди lscpu (Знімок екрану представлено автором на рис. 3.6)

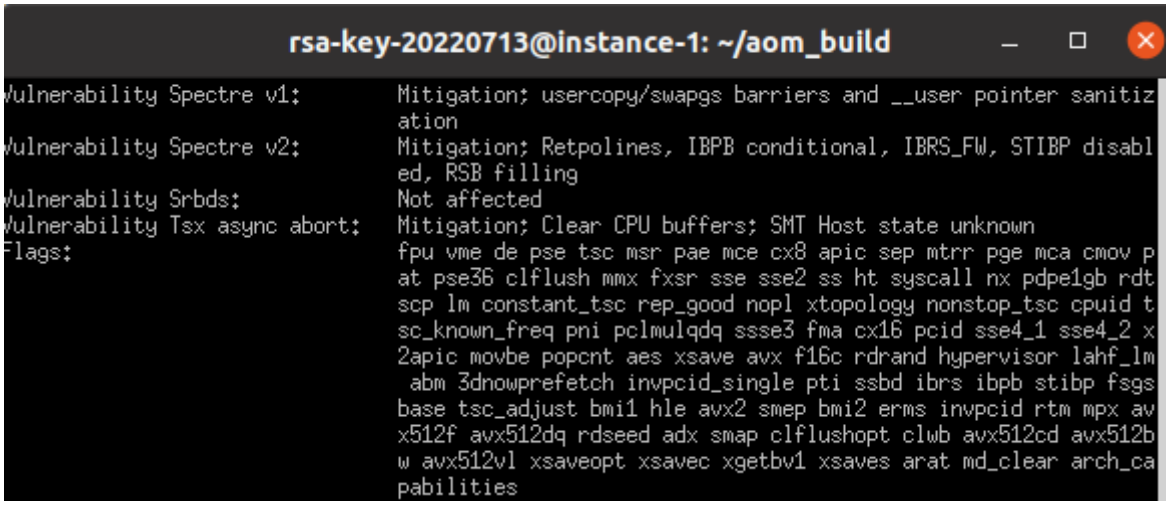

Рис. 3.6 Перевірка підтримки інструкцій SIMD AVX-512 на Google Cloud ОС Ubuntu 20.04

Інструкції щодо встановлення проекту під середовище Ubuntu 2017 описали на офіційній сторінці репозиторію Google AOM (AV1) [55]. Необхідною передумовою є наявність у системі підтримки NASM (Netwide-Assembler) та Cmake (завантажити із офіційних сайтів та встановити у Windows). За допомогою make-файлів можна зібрати проект відекодеку під різні архітектури, підтримки різних інструкцій, але за замовчуванням у проекті з репозиторію google libaom немає підтримки збірки під AVX-512 Для завантаження репозиторію потрібно ввести команди. Тому підтримка збірки make для під SIMD AVX-512 додано власноруч і потребувала редагування файлів з розширенням make з репозиторію.

Інтеграція коду на мові Ассемблер у репозиторій google aom

По-перше, необхідно активувати NASM (буде використовуватись замість YASM, який не підтримує код із AVX-512 набору розширень інструкцій). Для цього при побудові коду репозиторію необхідно передати аргумент командного рядка у cmake -DENABLE_NASM=1.

По-друге, для інтеграції файлу entdec_avx512.asm у код репозиторію, необхідно додати його у відповідний make-файл: av1.cmake у директорії aom/av1.

Для цього дадаємо цей файл до бібліотеки декодера, яка буде використовуватись для збірки файла виконання aomdec (Представлено автором на Рис. 3.7).

По-третє, додати оголошення відповідної функцію у файл av1_rtcd_defs.pl: *add_proto qw/int/, "asm_decode_symbol_adapt16", "od_ec_dec *ec, aom_cdf_prob *cdf, int nsymbs"; specialize qw/asm_decode_symbol_adapt16_avx512/;*

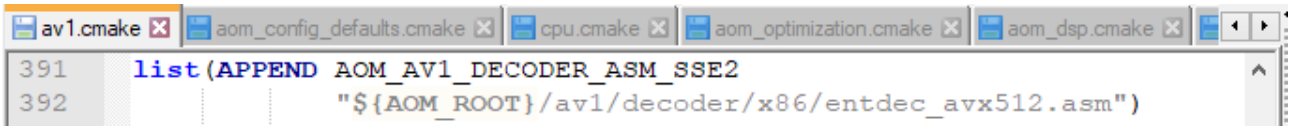

Рис. 3.7 Налаштування відеокодеку AV1- додавання файлу із кодом на мові

Вказане на Рис. 3.7 оголошення необхідно для коректного лінкування.

Якщо все пройшло коректно, то при заміні функції на нову версію на мові Assembler при збірці у командному рядку можна буде побачити, що об’єктний файл із нового коду на мові Assembler створився і нова функція знайдена (немає помилок при створенні aomdec.exe) (представлено автором на рис. 3.8):

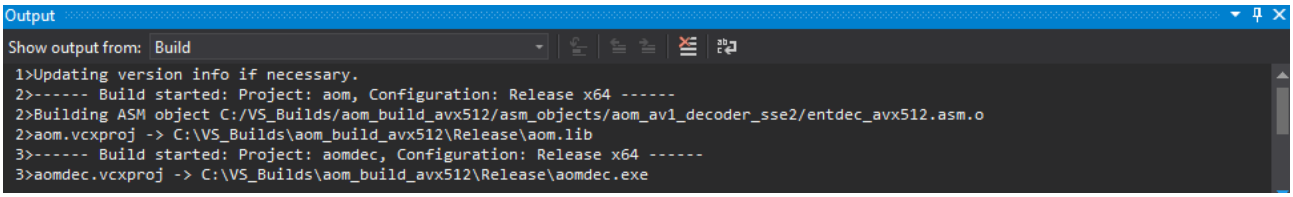

Рис. 3.8 Налаштування відеокодеку AV1- додавання файлу із кодом на мові Ассемблер у збірку

Для завантаження проекту слід використати команди

$ git clone https://aomedia.googlesource.com/aom

Збірка проекту Інструкції у середовище Ubuntu 2017 описана на офіційній сторінці репозиторію Google AOM (AV1) [55].

За основу візьмемо збірку для Linux для архітектури X64 із підтримкою декодера AV1, інструкцій AVX2 (самостійно змінено make-файл для додавання інструкцій AVX-512) та не будемо збирати тестові проекти примірники. Для цього переходимо у папку aom_build_avx512 і виконуємо дві наступні команди.

$ make ../aom -DENABLE_TESTS=0 -DCONFIG_AV1_ENCODER=0 -DENABLE_AVX2=1

$make -- build .

Якщо все пройшло успішно, то побачимо створений проект у директорії aom_build_avx512.

Для роботи декодеру AV1 до файлу aomdec, що з'явиться і буде виконуватись після збірки цього проекту потрібно передати наступні вхідні аргументи: імя файлу для декодування (завантажити із інтернет .webm файл, що закодували кодеком AV1 попередньо, та вказати вихідний файл — звичайний raw файл у форматі RGB. Для цього достатньо ввести у Visual Studio у налаштуваннях проекту у полі Application Command Arguments наступну команду:

$../Big_Buck_Bunny_1080_10s_30MB.webm -o OUTPUT.FILE --progress

Опція “-progress” показує чи працює кодек та оновлює вікно виконання (cmd) та вказує кількість кадрів, що успішно декодувались.

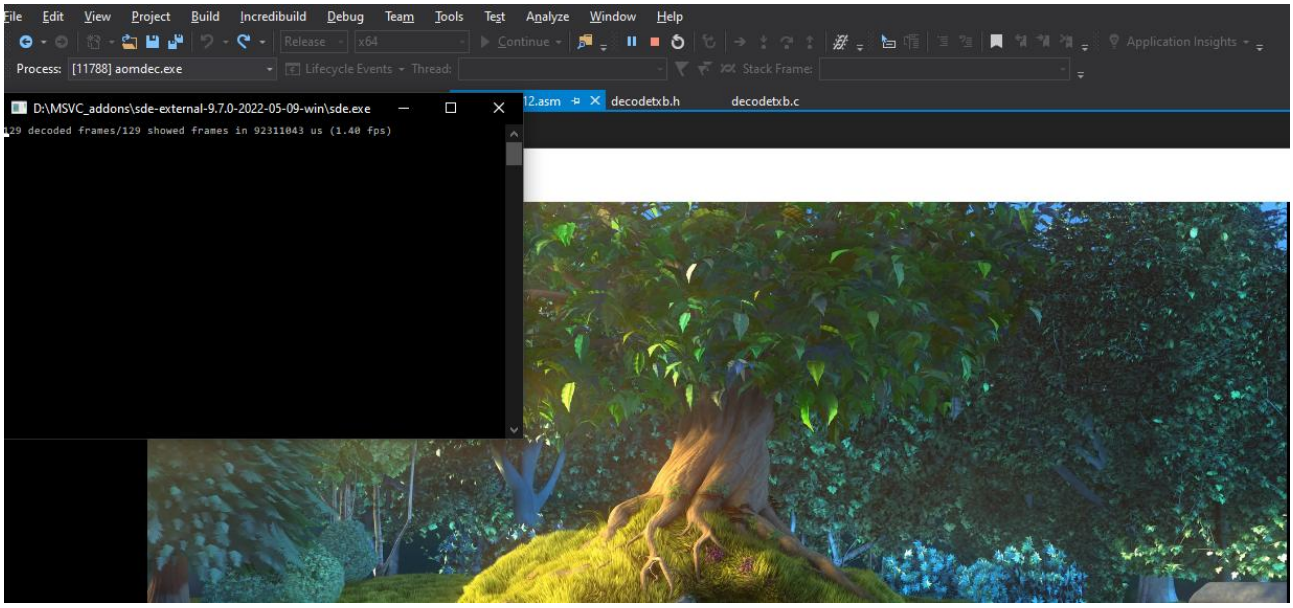

Рис. 3.9 Налаштування відеокодеку AV1: перевірка роботи програми з Intel SDE

Після успішного налаштування побачимо вікно виконання програми bash cmd, а після буде створено вихідний файл формату rgb OUTPUT.FILE який можемо відкрити за допомогою VLC-player, наприклад і побачити фрагмент мультфільму якщо все декодувалось коректно (представлено на рис.3.9).

Висновки до розділу 3

Досліджено оптимізований та згенерований за замовчуванням компілятором лістинг коду на мові Assembler ентропійного кодування.

Написано можливі реалізації коду із застосуванням розширеного набору інструкцій SIMD та проведено порівняння загальної кількості тактів виконання із врахуванням особливостей архітектури мультипроцесорної системи.

Виміряно продуктивність коду в залежності від характеристик ентропії закодованого відео файлу – середнього значення декодованого символу та розміру блоку. Встановлено залежності підвищення продуктивності коду на базі SIMD AXV-512 відносно коду на мові С та версії на базі SIMD AVX2 в залежності від характеристик відео-файлу.

Встановлено, що функція ентропійного декодування відео AV1 на мові Assembler на базі SIMD AVX-512 швидша за існуючу скалярну версію у випадку, якщо аргумент 1 функції вказує відносно великий розмір масиву кумулятивного розподілу ймовірностей (eob_sz> 5), та відносно більше значення, що повертається функцією (eob_pt> 5) - в такому випадку - на 10% швидше за скалярну версію коду у кращому випадку, та на 7% повільніше у гіршому випадку. (прискорення на базі SIMD залежить від кількості елементів масиву що оброблюються, та позиції символу у цьому масиві: більше – краще). Перевага над існуючою реалізацією на базі SIMD AVX2 – в середньому на 3-5% у всіх випадках, і досягається за рахунок скорочення кількості інструкцій функції ентропійного кодування за рахунок використання переваг набору SIMD AVX-512 над SIMD AVX2 для вдосконалення функції.

На основі аналізу і порівняння лістингу скалярної на мові С та векторизованої версії на мові Assembler змінено вигляд основної функції декодування decode_txb.

Також, у Розділі 3 показано інтеграція коду на мові Assembler представлення NASM (NetWide Assembler) у публічний репозиторій Google AOM шляхом по-перше написання коду на мові Assembler представлення NASM (Netwide-

Assembler) для всіх можливих типів розширеного набору інструкції SIMD (SSE, AVX/AVX2, AVX-512), по-друге редагування і доповнення файлів налаштувань збірки (make-файлів).

РОЗДІЛ 4
РОЗРОБКА СТАРТАП-ПРОЄКТУ

4.1. Інформаційна картка стартап-проєкту

Таблиця 4.1

Інформаційна карта проєкту

1. Назва проєкту	Оптимізація відео-кодування під різні мікроархітектури на базі SIMD
2. Автори проекту	Тимур Бойко
3. Коротка анотація	Приватний віддалений репозиторій із вдосконаленим кодом відео кодування AV1, що адаптований до максимальної утилізації можливостей мікроархітектури на базі SIMD. Клієнт може завантажити та інтегрувати код у свою систему, яка буде працювати швидше (більше кадрів/секунду під час кодування і декодування відео) – що дозволить досягти рекомендованих 24, 30 або 60 кадрів/с. У подальшому планується розширення як процесорних мікроархітектур, так і додавання коду для графічної процесорної субсистеми.
4. Термін реалізації проєкту	18 місяців
5. Необхідні ресурси	Фінансові ресурси
	1. Зарплати співробітників на місяць Інженер 1 (DSP/мат профіль) – 50 тис. грн Інженер 2 (GPU профіль) – 50 тис. грн Інженер 3 (C/C++ профіль) – 40 тис. грн Спеціаліст по маркетингу – 20 тис. грн. Бухгалтер – 30 тис. грн. 2. Оренда офісу – 360 тис. грн. 3. Оформлення авторського плану – 10 тис. грн. Всього 3.89 млн. грн. (3.52 + 0.36 + 0.01)
	Матеріальні ресурси
	3 ПК ASUS на базі Intel 7 – 15x3 тис. грн. 2 Ноутбуки ASUS на базі Intel 7 – 25x2 тис. грн

Таблиця 4.1 (продовження)

Інформаційна карта проєкту

5. Необхідні ресурси	Матеріальні ресурси
	5 Моніторів SAMSUNG 32” 10 bit – 11x5 тис. грн. 1 GPU NVIDIA RTX 3060 – 20 тис. грн. Всього 170 тис. грн.
	Інтелектуальні ресурси
	Програмне забезпечення Intel vTune, SDK, відео-редактори
6. Опис проблеми, яку вирішує проект	Відео-послідовність із більшим числом кадрів на секунду (fps), менше завантаження процесорної системи клієнта
7. Головні цілі та завдання	- переписування коду для пришвидшення відео-кодування для цільової мікроархітектури процесорний системи на базі SIMD (для ARM NEON, AVX-512) - переписування коду для пришвидшення відео-кодування для цільової мікроархітектури процесорний системи на базі GPU (для ARM MALI, AMD RADEON) - переписування коду для пришвидшення відео-кодування для цільової мікроархітектури процесорний системи на базі логічної зміни частин відео-кодування - переписування алгоритмів різних відео-кодеків (AV1, VP9, H.264), видалення непотрібних клієнту частин коду (агрегація, спеціалізація, тощо)
8. Очікувані результати	Проект з пришвидшення роботи відео-кодеків надасть клієнтам-замовникам можливість мати на своїх пристроях стабільну плавну картинку і в досягти цільового показника кадрів/с (24, 30, 60, 120)

4.2. Формування команди стартап-проєкту

Таблиця 4.2

Ролі співробітників у проєкті

Спеціальність	Роль
Маркетолог	Дипломат

Таблиця 4.2(продовження)

Ролі співробітників у проєкті

Спеціальність	Роль
Бухгалтер	Юридичний консультант, слідчий за не порушенням інтелектуальних прав
Інженер 1	Генератор ідеї, виконавець
Інженер 2	Генератор ідеї, виконавець
Інженер 3	Спеціаліст із інтеграції, майстер з мереж

Таблиця 4.3

Поставлені завдання та час на їх виконання

	Завдання	Час
1	Аналіз можливостей пришвидшення різних відео-кодеків	5
2	Розробка Технічного завдання	1,25
3	Розподілення ролей проекту	0.25
4	Оптимізація коду	10
5	Тестування коду	1
6	Пошук інвесторів	2
7	Маркетингова кампанія	2
8	Дослідження ринку та конкурентів	1

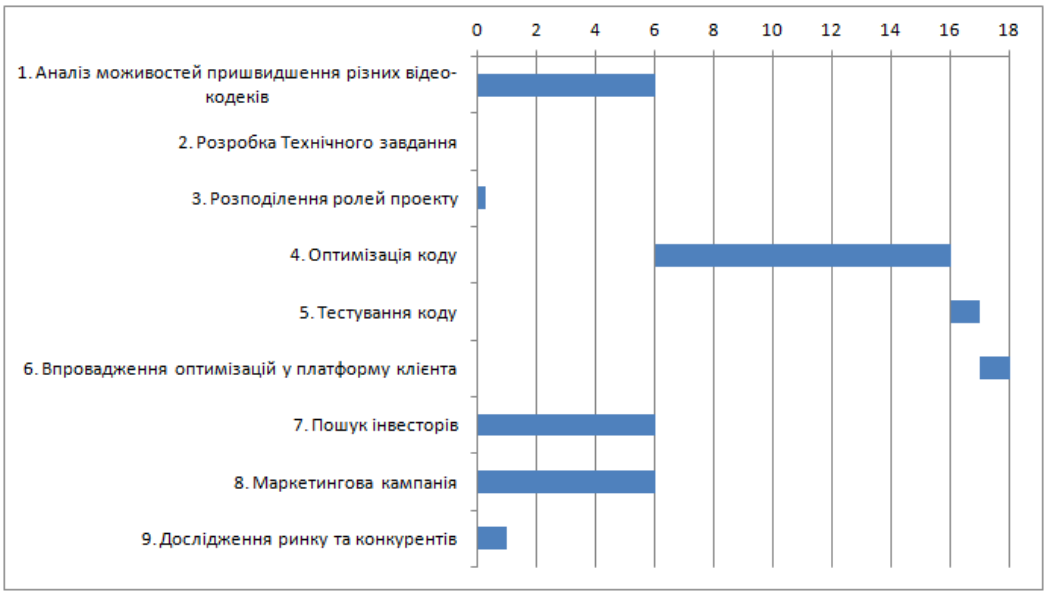

Рис. 4.1 Графік розподілення часу (діаграма Ганта)

Таблиця 4.4

Визначення важливості факторів щодо їх вкладу у створення та реалізацію інноваційного проекту

Фактор	Вага (важливість)
Ідея	10
Підготовка бізнес плану	5
Компетентність	10
Співпраця та ризики	8
Обов'язки	9

Таблиця 4.5

Оцінювання особистого внеску кожного партнера у створення та реалізацію стартапу

Фактор	Інженер 1	Інженер 2	Інженер 3	Маркетолог	Бухгалтер
Ідея	45	45	10	0	0
Підготовка бізнес плану	20	20	10	30	30
Компетентність	20	20	20	20	20
Співпраця та ризики	30	30	10	30	0
Обов'язки	20	20	20	20	20
Разом	135	135	70	100	70
Особистий внесок у проєкт	26%	26%	14%	20%	14%

4.3. Морфологічна карта стартап-проєкту

Морфологічні карти використані для генерування ідеї стартап-проєкту. Система на базі відео-кодування повинна бути швидшою за існуючі програмні рішення, мати спеціалізації під усі можливості мікроархітектур та операційних систем, вихідний результат роботи системи (кодований відео файл) не повинен відрізнятись від існуючих програмних рішень.

Визначено основні функції для користування системою:

- процесорна система із розширеним набором інструкцій SIMD AVX-512;
- процесорна система із розширеним набором інструкцій ARM NEON;
- потужна графічна процесорна (32 обчислювальні модулі і більше);
- кодеки, що використовує замовник – AV1;
- будь-яка ОС.

Побудуємо морфологічну карту, на якій зазначимо потенційні варіанти рішень – засобів реалізації кожної функції

Таблиця 4.6

Морфологічна карта стартап-проєкту

Основні параметри	1	2	3	4	5	6
SIMD ISA	SSE/NEON	AVX/AVX-2	AVX-512	SSE/NEON	AVX/AVX-2	AVX-512
GPU CUs	<32	<32	<32	32+	32+	32+
Алгоритм кодування відео	AV1	AV1	AV1	AV1	AV1	AV1

Ідею стартап-проєкту можна сформулювати так: мультимедійна система кодування відео пришвидшена за допомогою зміни коду під особливості конкретної мікро-архітектури.

Задум товару:

1. Товар за задумом. Приватний репозиторій із кодом відео AV1, адаптований до максимальної утилізації можливостей мікроархітектури процессорів із підтримкою набору розширень SIMD AVX-512 та потужним графічним прискорювачем.

2. Товар у реальному виконанні. Приватний репозиторій із кодом ентропійного кодування відео AV1, адаптований до максимальної утилізації можливостей мікроархітектури процессорів із підтримкою набору розширень SIMD AVX-512

3. Товар з підкріпленням. Приватний репозиторій із вдосконаленням коду відео AV1 для підтримки пришвидшення на базі потужних графічних прискорювачів (GPU із 32+ CUs), а також для максимальної утилізації можливості інших мікро-архітектур: ARM-NEON. А також розширеним набором алгоритмів кодування відео (AV1, VP9, H.264). Переформатування логічних частин кодування відео. Технічна підтримка та покращення роботи.

Таблиця 4.7

Опрацювання питань для удосконалення продукту

№	Запитання	Відповідь
1	Частиною яких систем є продукт?	Мультимедійних систем на платформі клієнта, які використовують алгоритм кодування відео AV1
2	Які функції надсистеми може виконувати продукт? Як їх з ним пов’язати?	Для використання бізнесом (власниками інтернет-тв, відео конференц-звязку) для пришвидшення роботи своїх сервісів.
3	Чи можна розділити продукт на частини?	Так, на логічні частини алгоритму кодування відео
4	Чи можна об’єднати (агрегувати) кілька елементів продукту в один?	Так
4	Чи можна нерухомі частини продукту зробити рухомими і навпаки?	Ні
6	Яким має бути ідеальний продукт?	Пришвидшене кодування усіх логічних частин відео кодування на усіх можливих мікроархітектурах процесорних та графічних процесорних системах
7	Що відбудеться, якщо вилучити цей продукт? Чим його можна замінити?	Відео-кодування бути працювати повільніше, а завантаження процесорної системи може зрости. Можна замінити погіршенням характеристик відео-кодування із використанням загальнодоступного репозиторію із кодом за замовчуванням

Таблиця 4.7 (продовження)

Опрацювання питань для удосконалення продукту

№	Запитання	Відповідь
8	Яким цей продукт був у минулому?	Раніше не був оптимізований під найбільш сучасні процесорні системи
9	На розвиток яких функцій було спрямоване удосконалення продукту?	Пришвидшення кодування відео
10	Які функції залишилися «недорозвиненими»?	GPU-оптимізація. Інші функції потребують подальшого дослідження та виявлення після отримання зворотнього зв’язку від користувачів

Проведемо відбір найцікавішихідей та сформуємо їх список.

Ідея 1. Надання послуг зі зміни коду відео кодування під конкрентну мікроархітектуру під замовлення клієнта.

Ідея 2. Приватний репозиторій із пришвидшеним зміненим кодом спеціалізованим під певні мікро-архітектури (платний доступ), внесені авторами зміни захищені авторськими правами.

Ідея 3. Додаток відео-конференц зв’язку для всіх ОС та мікроархітектур, у який інтегрувати змінений код.

Ідея 4. Створення різноманітних фільтрів та спецефектів для мультимедійних систем, що використовують кодування відео.

Агрегування 1. При комбінуванні ідей 1, 2 буде створений особистий репозиторій із підтримкою для певного клієнта, інтегрованим у мультимедійну систему для певного клієнта із його подальшою підтримкою.

Агрегування 2. При комбінуванні ідей 1 та 3 буде створений додаток для клієнта, що працює швидше за аналогічні

Агрегування 3. При комбінуванні ідей 1 та 4 будуть створені фільтри для мульмедійних систем клієнта

Отже, після агрегування маємо 3 додаткові ідеї. При їх комбінування перелік унікальних ідей збільшується до 10.

Оберемо найбільш привабливі ідеї та перевіримо ці ідеї на своєчасність.

Ідея 1. Надання послуг зі зміни коду відео кодування під конкрентну мікроархітектуру під замовлення клієнта:

- разове замовленя не гарантує подальші (не має стабільного грошового потоку);
- витрати часу тільки на роботи, що будуть оплачені;
- враховує усі особливості клієнта;
- можливість власної інтеграції у систему клієнта.

Ідея 2. Приватний репозиторій із пришвидшеним зміненим кодом спеціалізованим під певні мікро-архітектури (платний доступ), внесені авторами зміни захищені авторськими правами:

- немає необхідності шукати індивідуальних замовників;
- продаються вже зроблені роботи у вигляді підписки (що гарантує грошовий потік);
- роботи із оптимізації здійснюються за планом власної команди;
- великий обсяг робіт;
- роботи, що вже зроблені можна продавати в подальшому іншим.

Ідея 3. Додаток відео-конференц зв’язку для всіх ОС та мікроархітектур, у який інтегрувати змінений код:

- продаються вже зроблені роботи у вигляді додатку (що гарантує грошовий потік);
- велика потенційна аудиторія користувачів;
- можливий окремий продаж по підписці коду відео-кодування, що використовує додаток для непублічного використання;
- надвеликий обсяг робіт, особливо непрофільних (front-end, мережі, захист інформації);
- роботи, що вже зроблені можна продавати в подальшому іншим.

Ідея 4. Створення різноманітних фільтрів та спец ефектів для мультимедійних систем, що використовують кодування відео:

- разове замовленя не гарантує подальші (не має стабільного грошового потоку);

- велика потенційна аудиторія користувачів

- менший обсяг робіт;

- велика конкуренція (у зв’язку із меншою необхідною експертизою)

Агрегування 1. При комбінуванні ідей 1, 2 буде створений особистий репозиторій із підтримкою для певного клієнта, інтегрованим у мультимедійну систему для певного клієнта із його подальшою підтримкою:

- немає необхідності шукати індивідуальних замовників;

- продаються вже зроблені роботи у вигляді підписки (що гарантує грошовий потік);

- можливість продаж додаткових послуг клієнту (тих же фільтрів або інтеграції).

Агрегування 2. При комбінуванні ідей 1та 3буде створений додаток для клієнта, що працює швидше за аналогічні:

- разове замовленя не гарантує подальші (не має стабільного грошового потоку);

- більша вартість робіт;

- неможливість переключатись на інші замовлення.

Агрегування 3. При комбінуванні ідей 1 та 4 будуть створені фільтри для мульмедійних систем клієнта:

- разове замовленя не гарантує подальші (не має стабільного грошового потоку);

- менша вартість робіт;

- неможливість переключатись на інші замовлення.

Якщо поглянути на ідеї, що згенерували, та проаналізувати привабливі харакеристики, то стає очевидним, що Ідея 2 - Приватний репозиторійіз пришвидшеним коду для усіх мікроархітектур - є найбільш оптимальної для стартап-проєкту, як з точки зору організаційної структури стартапу, так з точки зору генерування майбутнього грошового потоку стартап-проєктом. Якщо список

підтримки мікроархітектур та логічних частин відео-кодування буде зростати, буде зростати і список потенційних замовників та максимальна вартість підписки.

4.4. Розроблення ринкової стратегії проєкту

Таблиця 4.8

Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Компанії відео-конференц звязку	100%	85%	Низький, присутні 3 гравці	Легка
2	Компанії з надання послуг інтернет-телебачення	50%	15%	Низький, присутні 3 гравці	Середня

За результатами аналізу цільових груп клієнтів у Таблиці 4.8 обираємо компанії відео-конференц-звязку як цільові, для яких буде запропоновано продукт компанії. Однак, сервісам інтернет-телебачення продукт також буде запропоновано. Зусилля будуть здійснюватись у співвідношенні 85:15 відповідно до орієнтовного попиту у межах цільової групи. Дане співвідношення із перевагою для компаній відео-конференц зв’язку може виглядати як стратегія концентрованого маркетингу (переважно не диференційованого).

Так як стартап-проєкт – це нова компанія на ринку, то їй необхідно зайняти долю ринку. Тому в Таблиці 4.9 як основний ринок пропонуються ринок послуг для оптимізації коду відео-кодування під процесорні архітектури, як альтернатива – під утилізацію можливостей графічних прискорювачей.

Таблиця 4.9

Визначення базової стратегії розвитку

Обрана альтернатива розвитку проєкту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
Створення приватного репозиторію із пришвидшеним кодом відео кодування під різні мікроархітектури на базі SIMD та GPU	Стратегія концентрованого маркетингу	- Більша швидкість роботи відео-додатків клієнта із-за використання послуг відео кодування - Супутні послуги (напр.- інтеграція у сервіси клієнта)	Стратегія спеціалізації

Тому компанія спочатку буде спеціалізуватись на SIMD оптимізація, а потім – на GPU без прагнення зайняти увесь ринок. Мала ринкова доля не може істотно вплинути на спроможність конкурентність компанії – оскільки це зростаючий ринок і попит на таку спеціалізація зростає.

Щодо стратегії конкурентної поведінки – тут обираємо стратегію зайняття ніші (Табл. 4.10)

Таблиця 4.10

Визначення стратегії конкурентної поведінки

Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основніриси товару конкурента?	Стратегія конкурентної поведінки
Ні, але із цільовою мікроархітектурою стартап-проєкту - так	Так, а також створювати конкуренцію іншим	Ні	Стратегія заняття конкурентної ніші - SIMD

Ніша – процесорні системи з підтримкою набору розширень інструкцій SIMD виду AVX-512. Така ніша задовольняє вимогам нішової стратегії розвитку:
- прибутковість (новий тип архітектури та відносна дороговизна гарантують наявність грошей у клієнта);
- підтримка набору розширень інструкцій SIMD виду AVX-512 гарантовано як мінімум 10 років;
- захищена, оскільки наявність експертиза у відео та ще й оптимізаціях під таку архітектуру – це доволі рідкісний тип робочої сили;
- не є привабливою для конкурентів – оскільки більшість процесорних систем клієнта не мають підтримки таких можливостей, а затрати на навчання співробітників – доволі довгий (роки) та неокупний процес;
- така ніша повністю відповідає можливостям співробітників даного стартап-проєкту.

Компанія буде підтримувати свою нішову унікальність через постійне підвищення їх експертизи (обмін знань, форуми, конференції), формування лояльності нішових клієнтів, підтримувати цей вхідний барєр.

На основі вимог потенційних споживачів до продукту (Табл. 4.16) та використання стратегій у Табл. 4.9 та Табл. 4.10 визначаємо стратегію позиціонування стартап-проєкту (Табл. 4.11)

Таблиця 4.11

Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції стартапу	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (3 ключових)
1	Використання відео кодування VP9, AV1 або H.264	Стратегія спеціалізації	Репозиторій доступний 24/7 для всіх підписників	швидкість роботи відео-кодування; індивідульні послуги: налаштування та інтеграція клієнту.

Таблиця 4.11 (продовження)

Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключовіконку-рентоспроможні позиції стартапу	Вибір асоціацій, які мають сфо-рмувати комплексну позицію власного проекту (3 ключових)
2	Використання можливостей SIMD та графічних прискорювачів	Стратегія спеціалізації	Репозиторій доступний 24/7длявсіх підписників	прискорення та розвантаження процесорної системи шляхом графічної процесорної системи; стабільність й безпомилковість.

Підсумовуючи конкурентний аналіз у Табл. 4.6 – Табл.11 сформуємо маркетингову концепцію продукту.

Таблиця 4.12

Визначення ключових переваг концепції потенційного товару

	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Швидкість роботи відео-кодування	Нові методи пришвидшення кодування відео	Відсутність аналогів із подібною експертизою
2	Адаптивність	Покриттямайже усіх мікроархітектур	Найбільше покриття кодеків та мікроархітектур
3	Індивідуальність	Можливість індивідуального налаштування	Еластичність та максимальна утилізація можливостей мультимедійної системи клієнта

Таблиця 4.13

Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Пришвидшення відео-кодування AV1 на базі на базі SIMDAVX-512 – у представленні приватного репозиторію, що постійно оновлюється		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1. Швидкість роботи	М	Тх/Ор
	2. Адаптивність (пітримка різних SIMD архітектур)	М	Тх/Ор
	3. Індивідуальність (інтеграція й підтримка клієнта)	М	Тх

Таблиця 4.13 (продовження)

Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
II. Товар у реальному виконанні	Якість:відповідає вимогам представлення коду Clang format та Google Code, пройдений аналіз файлу виконання на помилки та безпеку в системі Jenkins.
	Пакування:віддалений електронний репозиторій коду із персональним ключемдоступом. Електронна документація – в репозиторії і містить: - загальна назва продукту, власна назва; - авторські права та ліцензія; - технічні вимоги; - інструкції з налаштування та інтеграції; - контакти технічної підтримки.
III.Товаріз Підкріпленням	До продажу: індивідуальне виконання підтримки певних мікроархітектур
	Після продажу: додавання підтримки різних мікроархітектур у репозиторій, технічнапідтримка
За рахунок чого потенційнийтовар буде захищено від копіювання:ліцензії, авторського права, патенту	

Після маркетинговоїпрограмиформуємо концепцію маркетингових комунікацій (Табл. 4.14), яка базується на стратегії позиціонування (Табл. 4.11) із описом специфіки поведінки споживачів.

Таблиця 4.14

Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Каналикомуні-кацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
Клієнтидізнаютьсяпро товар з професій-них форумів, власним пошуком, перфоманс репортів	Інтернет	Вдосконале-ний продукт	Проінформувати про існування продукту	Швидша робота відео-сервісів, менше наванта-ження процесору

Отже, сформована у п.4.4 маркетингова програма для стартап-проєкту включає концепції потенційного товару, просування, попередній аналіз маркетингових комунікацій та інші.

4.5. Аналіз ринкових можливостей

Таблиця 4.15

Попередня характеристика потенційного ринку стартап-проєкту

№п/п	Показники стану ринку (найменування)	Характеристика
1	Кількістьголовних гравців, од	3
2	Загальний обсяг витрат на маркетинг (конференції, форуми)	100 000 грн./рік
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Відсутні
5	Специфічні вимоги до стандартизації та сертифікації	- однакова контрольна сума закодованого та декодованого файлів як в репо Google libaom; - відсутність попереджень та помилок, витоків памяті, підтверджена за допомогою спеціальних програм типу Jenkins; - Clang format;
6	Середня норма рентабельності в галузі (або по ринку), %	немає даних, оціночно ~30%

Таблиця 4.16

Характеристика потенційних клієнтів стартап-проєкту

Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
Вдосконалений метод кодування відео на базі можливостей архітектури SIMD	Бізнес: компанії розробники конференц-звязку та з надання послуг інтернет телебачення	фактори, що формують поведінку клієнта: - підтримка послугами цільової мікро-архітектури клієнта; - відсутність будь-яких помилок; - гарантія виконання робіт із заявленим пришвидшенням. Для деяких клієнтів ціна не є ірішальним.	До продукту: - пришвидшення кодування відео; - моливості надання супутніх послуг: інтеграція коду у медіа-систему клієнта - відсутність помилок та технічна підтримка. До компанії: - наявність експертизи у компанії або її співробітників.

Таблиця 4.17

Фактори загроз

№ п/	Фактор	Зміст загрози	Можлива реакція компанії
1.	Поява у споживача власних підрозділів вдосконалення коду відеокодування	Поява конкурентів із великих брендів	- поглибити спеціалізацію розробки на деякі інші не покриті мікроархітектури
2.	Репутація на ринку	Відсутність «історії успіху» бренду у порівнянні із конкурентами.	Вдоскналити маркетингову кампанію: - провести дослідження учасників форумів та конференцій для участі - дослідити цільову аудиторію видань для публікації звітів

Таблиця 4.17 (продовження)

Фактори загроз

№ п/	Фактор	Зміст загрози	Можлива реакція компанії
3.	Ціна	Умисне зниження цін (не значне) конкурентами для переманювання потенційних споживачів	У ході контакту із потенційним замовникам продемонструвати вищий рівень експертизи, пояснити переваги вищої ціни: - вищий рівень прискорення - кращий рівень тестування; - навички і знання розробників, залучення нових спеціалістів в штат для виправлення ситуації

Таблиця 4.18

Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1.	Нові типи мікроархітектур процесорних систем	Поява нових можливостей для переписування коду для нових архітектур	- додаткові витрати часу на вдосконалення рішення (дослідницьки роботи) - однак спочатку такий код буде оплачуватись вище.
2.	Відсутність бар'єрів виходу на глобальний ринок: стрімке поширення якісного відео звязку у бізнес-середовищі	Зростання обсягів ринку, поява нових платформ. Відео звязок як віддалене робоче місце потребує вищої якості та захисту інформації. Можливість - зайняти долю ринку у країнах із більш платоспроможним попитом, розвивати бренд глобально	Витрати на маркетинг в країнах, ринки яких є цікавими компанії

Таблиця 4.18 (продовження)

Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
3.	Потреба клієнта в індивідуальній модифікації послуг	Можливість додаткової оплати супутніх послуг	- Розширення штату під додаткові проекту; - Обмін знань із лідирую-чими компаніями світу; - Формування лояльності клієнта у наступних його проектах

Таблиця 4.19

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Чиста	Покупці і продавці не впливають на ціну - встановлюється на рівні середньої ціни ринкових угод за певний період	Необхідність оптимізації витрат для формування прибуткової діяльності за ринкової рівноважної ціни
Глобальний Рівень	Конкуренти за межами Країни, національний ринок має обмежену кількість покупців	Необхідність пошуку клієнтів на глобальному ринку, однак плюс: більше потенційних клієнтів і вища вартість послуг = зайняття більшої долі ринку
Внутрішньогалузева	Зростання конкуренціїсеред виробників продукту: компаній конференц відео зв’язку, сервісів інтернет-телебачення	Формування попиту та диктування покупцями умов на послуги для покращення відео для компаній конференц відео зв’язку
Товарно-видова	Конкуренція між послугами такого виду	Компанія має зосередись на задоволенні потреб цього виду

Таблиця 4.19 (продовження)

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
- Цінова (основна) (можлива нецінова)	Конкуренція стається із-за вдосконалення якості продукту та зменшення цін, однак існують випадки відданя переваги унікальності продукту	Зниження ціни або безкоштовне включення додаткових послуг. (встановлення вищої ціни на певну унікальну оптимізацію певної мікроархітектури)
Немарочна	Роль торгової марки незначна	Заохочення клієнтів якістю послуг, а не репутацією бренду

Таблиця 4.20

Аналіз конкуренції в галузіза М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачаль-ники	Клієнти	Товари-Замінники
	Ittiam DXC Squad	Барєри: Наявність досвіду та експертизи у кодуванні відео	Виробник є постачальником	Фактори сили споживачів: - Цінова чутливість споживачів - Обмежена кількість споживачів - Відмітні переваги продуктів компанії (унікальність)	Апаратна реалізація відео кодування: однак не покриє непопулярниі платформи

Таблиця 4.20 (продовження)

Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Висновки	Галузева конкуренція (між компаніями, що виробляють той же продукт або групу продуктів); Інтенсивність – чиста конкуренція	Можливості виходу на ринок є. Існує мала кількість конкурентів, що нівелюється кількістю споживачів. Строки виходу на ринок не залежать	Постачальник не має сили диктувати умови роботи на ринку	Клієнти диктують умови роботи на ринку: - цільова мікроархітектура, під яку переписується код відео кодування; - ціна (верхня межа оплати послуг стартапу)	Обмежень немає, оскільки товарів-замінників (апаратної реалізації) не має

Таблиця 4.21

Порівняльний аналіз сильних та слабких сторін

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-замінників у порівнянні зі стартап						
			-3	-2	-1	0	+1	+2	+3
1	Унікальність (швидша робота кодування відео)	20	x						
2	Можливість інтеграції та персоналізації	20				x			
3	Ціна	10			x				
4	Репутація компанії		x						
5	Відсутність барєрів виходу на ринок								x

Таблиця 4.22

SWOT-аналізстартап-проєкту

(S) Сильні сторони: - Унікальність: швидша робота кодування відео, набору мікроархітектур, що підтримуються; - можливість інтеграції та персоналізації; - відсутність барєрів на ринку	(W) Слабкі сторони: - ціна (можливо виявлення необхідного рівня в процесі діяльності); - репутація бренду стартап-проєкту.
(O) Можливості: - зайняття більшої долі ринку (виходу на найбільш дорожчі іноземні ринки); - формування власної ринкової унікальності (через співпрацю з клієнтів-володарями з рідкісними нових мікроархітектур); - неможливість впровадження аналогічного коду у комерційну діяльність конкурентів (патент)	(T) - цінова конкуренція; - конкуренція брендів; - зниження ринкового попиту (створення у цільових клієнтів власних підрозділів з метою оптимізації коду);

Таблиця 4.23

Альтернативи ринкового впровадження стартап-проєкту

Альтернатива (плановий набір заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки впровадження заходів
Контекстна реклама на відповідних сайтах з інформацією про форуми та конференції з відео-кодування	20%	3-5 місяців

Альтернативним впровадженням є контекстна реклама по пошуковим запросам стосовно пришвидшення відео-кодування, інтернет реклама на альтернативних репозиторіях. Однак, по-перше, вона з високою ймовірністю буде заблокована програмами фільтрації реклами. По-друге, розробники як правило цікавлять такими послугами самі, тому варто розміщувати її при відповідних

запитах «відео-кодування» чи «відео SIMD» у системах пошуку. Можливість друку більшого числа статей на тематику відео-кодування не є актуальною, бо потребує значної експертизи та людських ресурсів для їх написання. Це альтернативний сценарій, який може бути задіяний лише при провалі основного сценарію (дивись Таблицю 4.5).

4.6. Виробничий план

Таблиця 4.24

Календарний план-графік реалізації стартап-проєкту

№		Період впровадження стартап-проєкту						
		Рік 0				Рік 1	Рік 2	Рік 3
		I кв.	II кв.	III кв.	IV кв.			
1	Науково-дослідницькі роботи	+	+	+	+	+	+	+
2	Розробка технічного завдання та техніко-економічного обґрунтування	+						
3	Проектування та тестування	+	+	+				
4	Юридична реєстрація підприємства	+						
5	Придбання прав на нематеріальні активи	+						
6	Оренда офісного приміщення	+	+	+	+	+	+	+
7	Покупка комп’ютерного обладнання	+						
8	Пре-виробничі маркетинг-дослідження	+						
9	Впровадження ідеї стартап-проєкту	+						
10	Покупка інших матеріальних ресурсів	+						
11	Маркетингова кампанія				+			
12	Продаж продукту					+	+	+

Таблиця 4.25

Планова потреба у виробничих площах

№	Тип приміщення (будівлі)	Кількість одиниць	Площа, кв. м	Вимоги до приміщення (будівлі)	Умови користування приміщенням	Вартість на місяць, тис. грн.
1	Офісне	1	100	Норми робочого місця згідно ДСТУ	Орендна	30
Всього на рік:		-	-	-	-	360

Таблиця 4.26

Планова потреба у виробничому обладнанні та устаткуванні

№	Вид обладнання (устаткування, пристрою)	Тип (модель)	Виробник обладнання (устаткування, пристрою)	Терміни постачання	Вартість, тис. грн.
1	ПК	ROG (Intel 11400)	ASUS	1 тиждень	15
2	ПК	ROG (Intel 11400)	ASUS	1 тиждень	15
3	ПК	ROG (Intel 11400)	ASUS	1 тиждень	15
4	Відеокарта	RTX 3060 Ti	ASUS	1 тиждень	18
5	Монітор	IPS 32” 10 bit	Samsung	1 тиждень	11
6	Монітор	IPS 32” 10 bit	Samsung	1 тиждень	11
7	Монітор	IPS 32” 10 bit	Samsung	1 тиждень	11
8	Монітор	IPS 32” 10 bit	Samsung	1 тиждень	11
9	Монітор	IPS 32” 10 bit	Samsung	1 тиждень	11
10	Ноутбук	Vivobook OLED 14” Ryzen 5600H	ASUS	1 тиждень	25
11	Ноутбук	Vivobook OLED 14” Ryzen 5600H	ASUS	1 тиждень	25
12	Принтер-сканер	LaserJet	HP	1 тиждень	20
13	ПЗ	Windows Corporate	Microsoft	1 тиждень	20
14	Intel vTune	Intel MultiUser x3	Intel	1 тиждень	40
Всього на рік:		-	-	-	-

Таблиця 4.27

Планова вартість нематеріальних активів

№	Вид активів	Активи,щоможуть бути віднесенідо даного виду	Вартість, тис. грн.
1.	Права користування майном	Право на оренду приміщень	30
2.	Авторське право та суміжні з ним права	Право на комп'ютерніпрограми для розробки продукту	60

Таблиця 4.28

Плановий обсяг виробництва продукції стартап-проєкту

№	Вид продукції	Одиниця виміру	Обсяг виробництва за період			
			I рік	II рік	III рік	IV рік
1.	Програмне забезпечення	Підписка	2	5	10	15

Вартість підписки вираховується із відношення річних затрат стартап-проєкту до кількості прогнозованих клієнтів.

Компанія хоче бути резидентом «ДІЯ-СІТІ» та оподатковувати працюючих осіб по цивільним договорам по ставці 15%відзарплати

Таблиця 4.29

Планова потреба та витрати на персонал

№	Категорія персоналу	Чисельність	Заробітна плата,тис. грн.на місяць	Відрахування насоціальні заходи,тис грн. на місяць	Витратинаоплатупраці за період, тис. грн.		
					I рік	II рік	III рік
1	Інженер DSP	1	50	7,5	600	1200	1800
2	Інженер GPU	1	50	7,5	600	1200	1800
3	Інженер C++	1	40	6	480	960	1440
4	Маркетолог	1	20	3	240	480	720
5	Бухгалтер	1	30	4,5	360	720	1080
	-	5	190	28,5	2280	4660	6840

Таблиця 4.30

Загальні початкові витрати проекту

№	Стаття витрат	Витрати у нульовий рік,тис. грн.
1	Науково-дослідницькі роботи	600
2	Розробка технічного завдання та техніко-економічного обґрунтування	100
3	Проектування та тестування	100
4	Оренду офісного приміщення	360
5	Придбання робочого устаткування	188
6	Впровадження системи	0
7	Покупка нематеріальних активів	60
8	Інтернет-звязок	6
11	Пре-виробничі маркетингові дослідження	30
12	Витрати, пов’язані з діяльністю персоналу	-
	Разом	1644

Таблиця 4.31

Планові загальногосподарські витрати

№	Стаття витрат	Витрати за період, тис		
		I рік	II рік	III рік
1	Витрати на оренду земельних ділянок, будівель, приміщень, споруд	360	720	1080
2	Витрати на обладнання	188	0	0
3	Витрати на придбаннянематеріальних активів	60	0	0
4	Витрати на персонал (на відрядження, соціальні заходи тощо)	2280	4660	6840
5	Витрати на зв'язок	16	32	48
6	Витрати на паливо та електроенергію	30	60	90
7	Витрати на водопостачання	12	24	36
8	Витрати на утримання обладнання та приміщень	10	15	20
9	Витрати на збут	25	50	75
10	Витрати на просування та рекламу	30	60	90
11	Оплата юридичних послуг	10	20	30
12	Податкові платежі (земельний, комунальний податки, інші)	10	20	30
	Разом	3031	5661	8339

Висновки до розділу 4

У розділі 4 магістерської дисертації описано стартап-проєкт на основі бізнес-ідеї, зроблено бізнес-аналіз програмного забезпечення, що описується у розділі 3 – пришвидшений код відео кодування AV1 на базі SIMD. У даному розділі представлення цієї бізнес-ідеї – у вигляді репозиторію із платною підпискою.

В результаті маркетингового дослідження встановлено, що є можливість ринкової капіталізації проекту – оскільки на ринку є попит на такі послуги, ринок зростає.

Перспективи впровадження ідеї стартап-проєкту також існують – цільова група клієнтів (за видом мікроархітектури клієнта) буде зростати, а барєр входу у ціновому виразі відсутній, не ключову роль грає і репутація бренду, а головний барєр – досвід команди співробітників.

Для ринкового впровадження краще представити ідею у вигляді приватного репозиторію (що постійно оновлюється) із платною підпискою. Причому комерційне використання ідей буде захищено ліцензією, а виріб – патентом. Подальше вдосконалення проекту – доцільне для задоволення зростаючих потреб споживачів.

Бізнес-аналітика та маркетингове дослідження зроблено і представлено у вигляді таблиць: було узагальнено інформацію про організаційну структуру стартап-проєкту, детальний опис команди співробітників. Для швидкого і критичного представлення складені інформаційні та морфологічні картки стартап-проєкту. Створено маркетингова стратегія стартап-проєкту та створений виробничий план. Здійснено аналіз бізнес-можливостей та конкурентного середовища.

В ході дослідження середовища стартап-проєкту, було встановлено, що потенційний ринок та дохід стартап-проєкту знаходиться поза межами України. Це пов’язано, по-перше із малим обсягом вітчизняного ринку відео-конференц зв’язку (компанії з розробки таких сервісів) та компаній інтернет-телебачення. В

цілому на ринку присутні декілька конкурентів щодо пришвидшення роботи відео-кодування на базі SIMD, але вони переважно займаються або з іноземними клієнтами, або для власних потреб в своїх IoT-девайсах.

Бізнес-ідея стартапу є першим варіантом, і планується розширення її покриття на пришвидшення відео-кодування на усіх мікроархітектурах процесорних систем та різних відео-кодеках.

ВИСНОВКИ

У дисертації досліджено особливості сучасних мультипроцесорних систем. Розглянуто відмінності між поняттями «архітектура», «мікро-архітектура», «апаратна реалізація», вдосконалено розуміння цих понять.

У дисертації досліджено особливості та місце розширеного набору інструкцій SIMD у сучасні мультипроцесорній системі, визначено її місце та характер взаємодії в архітектурі.

Окрему увагу приділено визначеню місце сучасних процесорних систем у класичних системах за Флінном, місця доповненя і розширень у апаратній реалізації класичних архітектур, їх гібридний характер.

У дисертації наголошується на відмінностях і необхідності розділення понять «VLIW» та «набор розширень інструкцій SIMD».

Досліджено можливості та особливості роботи нового більшого розширення набору інструкцій нових процесорів Intel Skylake/Icelake/TigerLake – типу розширення AVX-512.

Визначено місце ентропійного кодування у кодуванні відео, показано принцип роботи кодування відео та його логічну схему.

Досліджено основні теоретичні методи ентропійного кодування та визначено належність ентропійного кодування daala згідно класифікації, показано принцип його роботи.

У дисертації вдосконалено код функції ентропійного на мові Assembler представлення NASM (Netwide-Assembler) для всіх можливих архітектур із підтримкою набору розширень SIMD типу SSE та AVX2.

У дисертації вперше додано код функції ентропіного кодування на мові Assembler представлення NASM (Netwide-Assembler) для архітектур із підтримкою набору розширень SIMD типу AVX-512.

У дисертації показано методи оцінки продуктивності коду як за допомогою синтетичних тестів так і за допомогою змін, що відключають усі модулі кодування відео окрім ентропійного кодування у публічному репозиторії AOM.

Запропонована автором реалізація ентропійного декодування відео AV1 на мові Assembler із застосуванням розширеного набору інструкцій SIMD AVX-512 в середньому на 10% швидше за скалярну версію коду у кращому випадку, та на 7% повільніше у гіршому випадку (в залежності від параметрів контексту відео файлу, що закодували), ау порівнянні із існуючою реалізацією на базі SIMD AVX2 – в середньому на 3-5% швидше у всіх випадках. Перевага над скалярною версією досягається за рахунок паралелізму на рівні даних кумулятивних розподілів ймовірностей та позиції декодованого символу у цьому масиві, а перевага над існуючими реалізаціями на базі SIMD AVX2 – за рахунок скорочення кількості інструкцій програми.

Цілі, поставлені у дисертації виконано.

Мета, поставлена у дисертації досягнута.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Харріс Д. М., Харріс С.Л. Цифрова схемотехніка та архітектура комп'ютера, друге видання, English Edition. [Текст] / Харріс Д., Харріс С. : К.: ДМК Прес, 2013. — 376 с.
2. Fisher A., Faraboschi P., Young C., Kaufmann M. A VLIW Approach to Architecture, Compilers and Tools.[Text] /Fisher A., Faraboschi P. —San Francisco.: Joseph. —2005. — 712 p.
3. Афонін І., Кабанчик Д. Сучасні процесорні архітектури.[Текст] // Журнал «СТА».— М.: «СТА-Пресс»,2020. —№1. — с.100-108.
4. An Introduction To Very-Long Instruction Word (VLIW) Computer Architecture [Електронний ресурс] / Philips Semiconductors. — Режим доступу : http://twins.ee.nctu.edu.tw/courses/ca_08/literature/11_vliw.pdf — Дата доступу : Серпень 2022.
5. Flynn M. Some Computer Organizations and Their Effectiveness.[Text] / Flynn M//Transactions on Computers C21/9.—Baltimore: IEEE,1972.— p. 948-960.
6. Корочкін О.В., Русанова О.В. Н72 Паралельні та розподілені обчислення. Навч. посібник. [Електронний ресурс] / О.В.Корочкін, Русанова О.В. – Електронні текстові дані (2 файла: 43,8 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2020. – 123 с.
7. K. Hwang. Advanced Computer Architecture: Parallelism, Scalability, Programmability. [Text] / K. Hwang.—N.Y.: McGraw-Hill, 1993.— 770 p.
8. Spector A, Gifford, D. The space shuttle primary computer system [Text] /Spector A, Gifford, D. // Communications of the ACM. 27 (9). N.Y.: ACM Press, 1984. — №9. — p.872–900.
9. Максимов Н. В. Партика Т. Л., Попов І. І. М17 Архітектура ЕОМ та обчислювальних систем: Підручник [Текст] / Максимов Н. В.— М.: Форум: Інфра-М, 2005. — 512 с.
10. “Skylake (client) - Microarchitectures - Intel”. [Електронний ресурс] / WikiChip. — Режим доступу:

https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(client) — Дата доступу : Серпень 2022.

11. Malishevsky A., Beck D., Schmid A., Landry E. Dynamic Branch Prediction. [Text] / Malishevsky A., Beck D., Schmid A., Landry E. — Oregon, USA: Oregon State University, 2001. — 16 p.
12. Eyerman S., Smith J., Eeckhout L. Characterizing the branch misprediction penalty. [Text] // IEEE International Symposium on Performance Analysis of Systems and Software. —N.Y.: IEEE, 2006. — p. 48-58.
13. D. Harris, S. Harris. Digital Design and Computer Architecture, Engineering professional collection [Text] / D. Harris, L. Harris. — Amsterdam, Netherlands: Elsevier, 2012. . — 690 c.
14. Shen J., Lipasti M. Modern Processor Design: Fundamentals of Superscalar Processors 2nd Edition. [Text] / Shen J., Lipasti M – Long Grove, Illinois: Waveland Press, 2013. — 658 p.
15. Crawford J., Gelsinger P. Programming the 80386. [Text] / Crawford J., Gelsinger P. — N.Y.: SYBEX, 1987. — 774 p.
16. Patt, Y., Hwu W., Shebanow M. HPS, a new microarchitecture: Rationale and introduction. [Text] / Patt, Y., Hwu W., Shebanow M. — Chicago:ACM SIGMICRO Newsletter. — 1985. — №16(4). — pp.103-108.
17. “Register Renaming”. [Електронний ресурс] / University of Minnesota. — Режим доступу : https://www.d.umn.edu/~gshute/arch/register-renaming.xhtml — Дата доступу : Серпень 2022.
18. Perais A. Increasing the performance of superscalar processors through value prediction. Hardware Architecture. [Text] / Perais A. — Renne, France.: Université Rennes, 2015. — №1. — p. 40-49.
19. Харріс Д. М., Харріс С.Л. Цифрова схемотехніка та архітектура комп'ютера, третє видання [Текст] / Харріс Д., Харріс С.: переклад з англ. Imagination Technologies.— К.: ДМК Прес, 2018.— 792 с.
20. Fog A. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs. [Електронний ресурс] /

Fog A. // Technical University Denmark.—http://www.agner.org/ /optimize/instruction_tables.pdf— Дата доступу : Червень 2022.

21. Granlund T. Instruction latencies and throughput for AMD and Intel x86 Processors. [Електронний ресурс]. —https://gmplib.org/~tege/x86-timing.pdf— Дата доступу : Червень 2022.
22. Intel Architecture Code Analyzer. [Електронний ресурс] // Intel Corporation. —https://software.intel.com/en-us/articles/intel-architecture-code-analyzer— Дата доступу: Червень 2022.
23. Intel 64 and IA-32 Architectures Optimization Reference Manual 2012. [Електронний ресурс] // IntelCorporation. — https://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-optimization-manual.pdf— Дата доступу: Червень 2022.
24. Intel 64 and IA-32 Architectures Optimization Reference Manual 2017. [Електронний ресурс] // Intel Corporation. — https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.pdf.— Дата доступу: Червень 2022.
25. Reddit. [Електронний ресурс] // Reddit. —https://reddit.com.— Дата доступу: Червень 2022.
26. Abel A., Reineke J.Uops.info: Characterizing Latency, Throughput, and Port Usage of Instructions on Intel Microarchitectures. [Text] / Abel A., Reineke J. //ASPLOS'19: Proceedings of 24th International Conference on Architectural Support for Programming Languages and OS. —2019.—p.673–686
27. x86/x64 SIMD Instruction List (SSE to AVX512). [Електронний ресурс]. — https://www.officedaytime.com/simd512e. — Дата доступу: Червень 2022.
28. Richardson I. H.264 and MPEG-4 Video Compression. [Text] / Richardson I. — Hoboken, N.J.: Wiley. — 2003. — 306 p.
29. Encoding Parameters of Digital Televisions for Studios.[Електронний ресурс] // ITU-R BT.601-4. — https://www.itu.int/rec/R-REC-BT.601-4-199407-S/en 1990.— Дата доступу: Червень 2022.

30. Tekalp A. Digital Video Processing. [Text] / Tekalp A.— Englewood Cliffs, NJ: Prentice-Hall.— 1995.— 624 p.
31. Poynton C. A Technical Introduction to Digital Video. [Text] / Poynton C.— New York, N.Y.: Wiley.— 1996.—736 p.
32. Wallace G. The JPEG still picture compression standard. [Text] / Wallace G. —. Chicago, USA.:Communications of the ACM.— № vol. 34, no. 4.— p. 30–44.
33. Ahmed N., Natarajan T., Rao K. On image processing and a discrete cosine transform. [Text] / Ahmed N // Transactions on Computers.— N.Y.: IEEE. .— № 23/1.— 1974.— p. 90-93
34. Ely S. MPEG video coding A simple introduction. [Text] / Ely S. // EBU Technical Review.— Geneva: EBU. — №12. — 1995.— p. 12-23
35. Shannon C. A mathematical theory of communication. [Text] / Shannon C // Bell System Technical Journal. —N.Y.:Bell. —№7 vol. 27. — 1948. — p. 379–423.
36. D. Huffman. A method for the construction of minimum redundancy codes. [Text] / D. Huffman // Proceedings of the IRE. — N.Y.: IEEE. — №9 vol. 40.— 1952. — p. 1098–1101.
37. Golomb S. Run-length encoding. [Text] Golomb S.// IEEE Transactions on Information Theory. — N.Y.:IEEE. —№ 3 (vol. IT-12). — 1966. — p. 399–401.
38. O'HanenB., Wisan M. JPEG Compression.[Електронний ресурс]. — http://mse.redwoods.edu/darnold/math45/laproj/fall2005/benmatt/paper.pdf. — Дата доступу: Серпень 2022.
39. Sharma M. Compression Using Huffman Coding. [Text] / Sharma M. // International Journal of Computer Science and Network Security.— № 5 (vol 10).— 2010.— p.133-141.
40. M. Mitzenmacher. On the Hardness of Finding Optimal Multiple Preset Dictionaries.[Text] / M. Mitzenmacher // IEEE Transaction on Information Theory.— N.Y.: IEEE.— №7 vol. 50.— 2004.—p. 1536-1539.
41. K. Gregory. The JPEG Still Picture Compression Standard. [Text] / K. Gregory// Massachusetts:Wallace Multimedia Engineering Digital Equipment Corporation.—N.Y.:IEEE.— 1991.

42. Witten I, Neal R, Cleary J. Arithmetic coding for data compression. [Text] / Witten I, Neal R // Communications of the ACM.—N.Y.:ACM.— № 6 vol. 30.— 1987.— p. 857–865.
43. RissanenJ., LangdonG. Arithmetic Coding. [Text] / RissanenJ., LangdonG // IBM Journal of Research and Development.— N.Y.:IBM.— № 23(2).— 1979.— p. 146–162.
44. Redmill D., Bull D. Error Resilient Arithmetic Coding of Still Images. [Text] // Image Communications Group, Centre for Communications Research, University of Bristol.— N.Y.:IEEE.— № vol (2).— 1996.— p. 109-112.
45. Kavitha V., Easwarakumar K. Enhancing Privacy in Arithmetic Coding. [Text] / Kavitha V. // ICGST-AIML.— Delaware, USA: ICGST-AIML Journal.— Volume 8, Issue I.— 2008.
46. Li Z., Drew M. Fundamental of Multimedia. [Text] / Li Z., Drew M. // School of Computing Science Fraser University.—N.J.: Prentice Hall.— 2004.—576 p.
47. Chanda S., Rana U. Evaluation of Huffman-Code and B-Code Algorithms for Image Compression Standards. [Text] / Chanda S., Rana U. // Proceedings of Fifth International Conference on Soft Computing for Problem Solving.—Berlin: Springer.— Vol 1.— 2016.—p. 1051-1060.
48. Terriberry T. Coding tools for a next generation video codec.[Електронний ресурс]. — https://tools.ietf.org/html/draft-terriberry-codingtools-02, 2015. —Дата доступу: Серпень 2022.
49. Stuiver L., Moffat A. Piecewise integer mapping for arithmetic coding. [Text] // Stuiver L., Moffat A // Data Compression Conference, Proceedings. —N.Y.: IEEE. — 1998. — p. 3-12.
50. Marpe D., Schwarz H., Wiegand T. Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. [Text] // IEEE Transactions on Circuits and Systems for Video Technology. — 2003. —№7 vol. 13. — p. 620-636.
51. Valin J., Terriberry T., Egge N., Daede T., Cho Y., Montgomery C., Bebenita. M. Daala: Building a next-generation video codec from unconventional technology.

[Text] // IEEE 18th International Workshop on Multimedia Signal Processing (MMSP). — N.Y.:IEEE. — 2016. —p. 1-6.

52. Fenwick P. A new data structure for cumulative frequency tables. [Text] / Fenwick P. // SoftwarePractice and Experience. — Hoboken, N.J.: John Wiley & Sons, Inc. — №3 vol 24. —1994. —p. 327-336.
53. Moffat A. Critique of “Novel design of arithmetic coding for data compression”. [Text] / Moffat A. // IEE Proc. Comput. Digit. Tech. — N.Y.:IEEE. — 1997. — p. 257-292.
54. Moffat A., Harman N., Witten I., Bell T. An empirical evaluation of coding methods for multi-symbol alphabets. [Text] // Inf. Process. Manage. — N.Y.:IEEE. —№6(30) . — 1994. — p. 791-804
55. “Alliance for Open Media”. [Електронний ресурс]. — http://aomedia.org. — Дата доступу: Липень 2022.
56. Chen Y. An Overview of Core Coding Tools in the AV1 Video Codec. [Text] / Chen Y.// Picture Coding Symposium (PCS. — N.Y.:IEEE. — 2018. —p. 41-45.
57. Polyanskiy Y., Wu Y. [Електронний ресурс] // University of Illinois Urbana-Champaign. — https://ocw.mit.edu/courses/6-441-information-theory-spring-2016/resources/mit6_441s16_course_notes. — Дата доступу: Липень 2022.
58. Stakoverflow - AVX2 what is the most efficient way to pack left based on a mask?[Електронний ресурс]// Stakoverflow. — https://stackoverflow.com/questions/36932240/avx2-what-is-the-most-efficient-way-to-pack-left-based-on-a-mask. — Дата доступу: Вересень 2022.
59. Brendan G. Linux perf Examples // Brendangregg.com. — https://www.brendangregg.com/perf.html. — Дата доступу: Вересень 2022.
60. KleenA.PMU-Tools. [Електронний ресурс] / KleenA// Github. — https://github.com/andikleen/pmu-tools/blob/master/hsw_client_ratios.py. — Дата доступу: Вересень 2022.
61. PerformanceCounters. [Електронний ресурс] // Microsoft. — https://docs.microsoft.com/en-us/windows/win32/perfctrs/performance-counters-portal. — Дата доступу: Вересень 2022.

ДОДАТКИ

Додаток А

Код на мові Assembler у представленні NASM для мультипроцесорних систем із підтримкою AVX-512

```
global asm_decode_symbol_adapt16_avx512
asm_decode_symbol_adapt16_avx512:
push          rbp     ; Save the old base pointer value.
mov           rbp, rsp ; Set the new base pointer value.
sub           rsp, 120   ; Make room for one 120-byte local variable.
push          r8      ; Save the values of registers that the function
push          r9      ; will modify. This function uses EDI and ESI.
push          r10
push          r11
;mov          rax, 0xFF00FF00FF00FF00 ; zero low 8 bits
movq          xmm3, [rdi + 32]
vlddqu        ymm1, [rsi]
neg           rdx
pshuflw       xmm2, xmm3, 170
movd          [rsp+20], xmm2
vpermq        ymm2, ymm2, 0
vmovdqu       ymm0, ymm1
vpsrlw        ymm1, 6
vpsllw        ymm1, 7
vpsrlw        ymm2, 8
vpsllw        ymm2, 8
vpmulhuw      ymm1, ymm2
lea           rax, [rel array] ;array(%rip), %rax
movdqu        xmm2, [rax + rdx + 16]
neg           rdx
```

```
vpmovsxbw      ymm2, xmm2
vpaddw         ymm1, ymm2
pshuflw        xmm3, xmm3, 85
vpermq         ymm3, ymm3, 0
vmovdqu        [rsp + 24], ymm1
vpcmpuw        k7, ymm1, ymm3, 2
vpcmpeqw       ymm2, ymm2, ymm2
vmovdqu16      ymm1 {k7}{z}, ymm2
kmovq          rax, k7
update_cdf_avx512:
mov            r10, 0x0
mov            r10w, word [rsi + rdx * 2]
vpcmpeqw       ymm2, ymm2
mov            r11d, r10d
shr            r10d, 4
add            r10d, 5
cmp            r11d, 32
adc            r11d, 0
movd           xmm3, r10d
vpavgw         ymm2, ymm1
vpsubw         ymm2, ymm0
vpsubw         ymm0, ymm1
vpsraw         ymm2, xmm3
vpaddw         ymm0, ymm2
mov            [rsi + rdx * 2], r11w
mov            r11, 1
shlx           r11, r11, rdx
sub            r11, 1
kmovq          k1, r11
vmovdqu16      [rsi] {k1}, ymm0
```

```
;.renorm:
tzcnt        rax, rax
;shr         rax, 1
mov          r8w, [rsp + 2*rax + 24]
mov          r11w, [rsp + 2*rax + 22]
;shr         rax, 1
;shl         r8, 16
;add         r8, r11
;shl         rax, 32
;add         rax, r8
;ret
;.renorm2:
sub          r11w, r8w ;rng = u-v (uint16_t)
shl          r8, 16  ; v<<16              (uint32_t)
mov          r9d, [rdi + 32]
sub          r9d, r8d ; (dif_f_input)             (r9d - r8 = r->dif - v<<16)
;.renorm4:
bsr          r8d, r11d
sub          r8w, 15
add          word [rdi + 38], r8w
neg          r8d
add          r9d, 1
shlx         r9d, r9d, r8d
sub          r9d, 1
shlx         r11d, r11d, r8d  ; dec->rng = rng << d;
mov          [rdi+36], r11w  ;w  2
mov          [rdi+32], r9d  ;dw 4
; Subroutine Epilogue
pop r11     ; Recover register values
pop r10
```

```
pop r9
pop r8
mov rsp, rbp ; Deallocate local variables
pop rbp ; Restore the caller's base pointer value
ret
```

Додаток Б

Код на мові Assembler у представленні NASM для мультипроцесорних систем із підтримкою AVX2

```
global asm_decode_symbol_adapt16_avx2
asm_decode_symbol_adapt16_avx2:
push        rbp     ; Save the old base pointer value.
mov         rbp, rsp ; Set the new base pointer value.
sub         rsp, 120   ; Make room for one 120-byte local variable.
push        r8      ; Save the values of registers that the function
push        r9      ; will modify. This function uses EDI and ESI.
push        r10
push        r11
;mov        rax, 0xFF00FF00FF00FF00 ; zero low 8 bits
movq        xmm3, [rdi + 32]
vlddqu      ymm1, [rsi]
neg         rdx
pshuflw     xmm2, xmm3, 170
movd        [rsp+20], xmm2
vpermq      ymm2, ymm2, 0
vmovdqu     ymm0, ymm1
vpsrlw      ymm1, 6
vpsllw      ymm1, 7
vpsrlw      ymm2, 8
vpsllw      ymm2, 8
vpmulhuw    ymm1, ymm2
lea         rax, [rel array] ;array(%rip), %rax
movdqu      xmm2, [rax + rdx + 16]
neg         rdx
vpmovsxbw   ymm2, xmm2
```

```
vpaddw         ymm1, ymm2
pshuflw        xmm3, xmm3, 85
vpermq         ymm3, ymm3, 0
vmovdqu        [rsp + 24], ymm1
vpsubusw       ymm1, ymm3
vpxor          ymm2, ymm2
vpcmpeqw       ymm1, ymm2
vpmovmskb      rax, ymm1
update_cdf_avx2:
mov            r10, 0x0
mov            r10w, word [rsi + rdx * 2]
vpcmpeqw       ymm2, ymm2
mov            r11d, r10d
shr            r10d, 4
add            r10d, 5
cmp            r11d, 32
adc            r11d, 0
movd           xmm3, r10d
vpavgw         ymm2, ymm1
vpsubw         ymm2, ymm0
vpsubw         ymm0, ymm1
vpsraw         ymm2, xmm3
vpaddw         ymm0, ymm2
mov            [rsi + rdx * 2], r11w
vmovdqu        [rsp + 58], ymm0
mov rcx, rdx
mov r8, 0
loop_save_updated_cdfs2:
movzx          r11, word [rsp + 58 + r8]
mov            [rsi + r8], r11w
```

```
add         r8, 2
dec         cl
jnz loop_save_updated_cdfs2
;.renorm:
tzcnt       rax, rax
shr         rax, 1
mov          r8w, [rsp + 2*rax + 24]
mov          r11w, [rsp + 2*rax + 22]
;shr         rax, 1
;shl         r8, 16
;add          r8, r11
;shl         rax, 32
;add          r8, r11
;shl         rax, 32
;add          rax, r8
;ret
;.renorm2:
sub         r11w, r8w ;rng = u-v (uint16_t)
shl         r8, 16  ; v<<16             (uint32_t)
mov          r9d, [rdi + 32]
sub         r9d, r8d ; (dif_f_input)            (r9d - r8 = r->dif - v<<16)
;.renorm4:
bsr         r8d, r11d
sub         r8w, 15
add          word [rdi + 38], r8w
neg         r8d
add         r9d, 1
shlx        r9d, r9d, r8d
sub         r9d, 1
shlx        r11d, r11d, r8d  ; dec->rng = rng << d;
```

```
mov         [rdi+36], r11w  ;w  2
mov         [rdi+32], r9d  ;dw 4
; Subroutine Epilogue
pop r11     ; Recover register values
pop r10
pop r9
pop r8
mov rsp, rbp ; Deallocate local variables
pop rbp ; Restore the caller's base pointer value
ret
```

Додаток В

Схема роботу відео кодування (AV1)

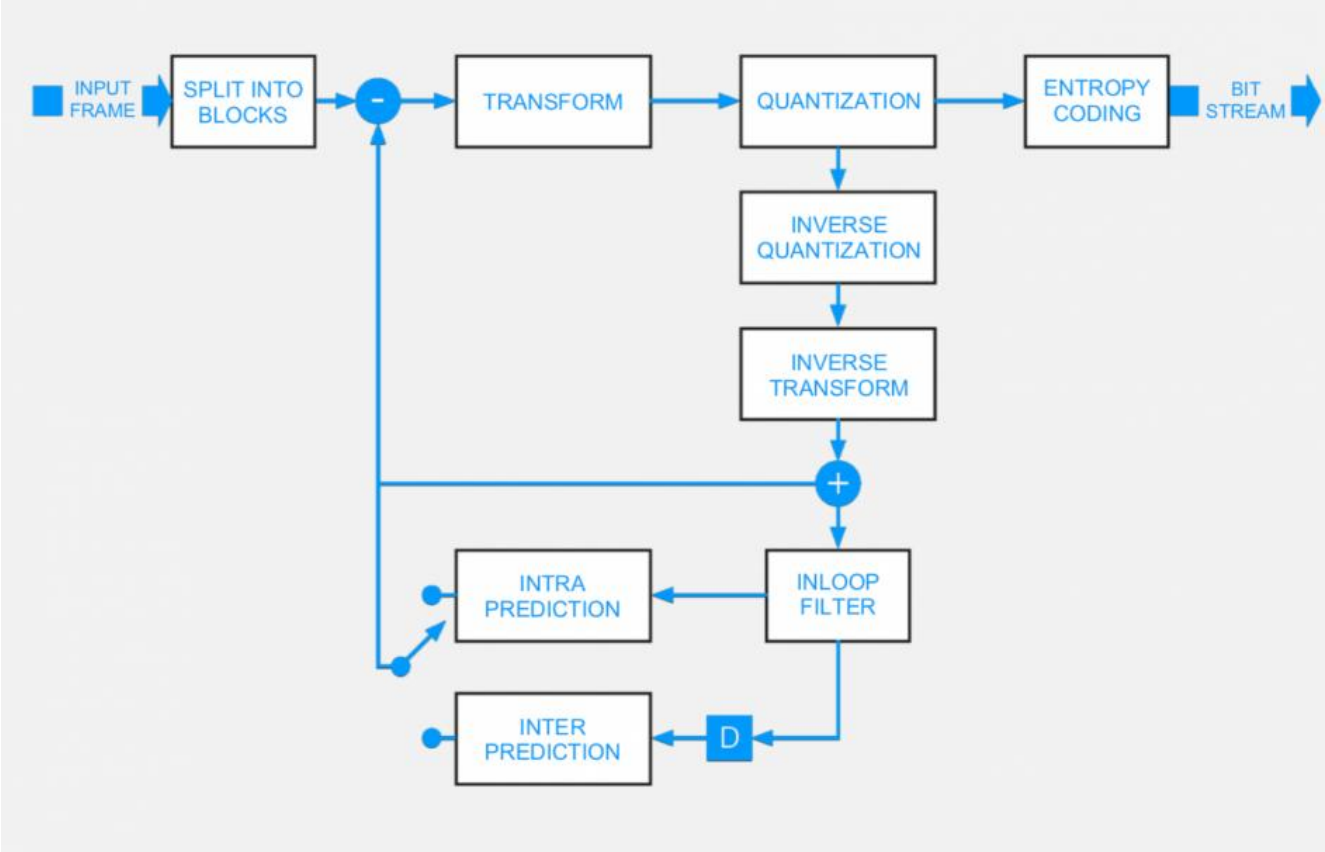

Рис. В Схема роботу відео кодування (AV1)

Джерело:[*https://www.sisvel.com/images/blog/2021/beamr_image.png*](https://www.sisvel.com/images/blog/2021/beamr_image.png)

Додаток Г

ТаблицяГ Представлення коду на мові Assembler ентропійного декодування із використанням набору розширень AVX2 аналогічного до скалярної версії функції за замовчуванням aom_read_symbol

```
array db 60, 56, 52, 48, 44, 40, 36, 32, 28, 24, 20, 16, 12, 8, 4, 0, 0, 0, 0, 0
;RCX/XMM0, RDX/XMM1, R8/XMM2, R9/XMM3
global asm_decode_symbol_adapt8_sse
...
```

; код на мові ассемблер — обчислюються одночасно 8 елементів		; код на мові С — обчислюються одночасно 1 елемент перебором у циклі do-while
01	movq xmm3, [rcx + 32]	*Завантажуємо член dec.dif та dec.rng в xmm3*
02	lddqu xmm1, [rdx]	*Завантажуємо* масив ймовірностей icdf в xmm1
03	neg r8	nsymbs = -nsymbs
04	pshuflw xmm2, xmm3, 170	Всі доріжки xmm2 = dec.rng;
05	movd [rsp+20], xmm2	*Завантажуємо* подвійне слово v = dec.rng
06	punpcklqdq xmm2, xmm2	
07	movdqu xmm0, xmm1	Аналогічно скалярній операції u = v (копіюємо xmm1 в xmm0)
08	vpsrlw ymm1, 6	Аналогічно скалярній операції a = icdf[0..16] >>6
09	vpsllw ymm1, 7	Аналогічно скалярній операції r = a << 7
10	vpsrlw ymm2, 8	Аналогічно скалярній операції r >> = 8
11	vpsllw ymm2, 8	Аналогічно скалярній операції b = (r>>8)<<8
12	vpmulhuw xmm1, xmm2	Аналогічно скалярній операції v = a*b >> 16
13	lea rax, [rel array]	Копіємо по ефективній адресі (RIP) лукап масив
14	lddqu xmm2, [rax + r8 + 16]	Завантажуємо 16 байт з лукап масиву
15	neg r8	nsymbs = -nsymbs
16	vpmovsxbw ymm2, xmm2	Sign extend packed 8-bit integers in a to packed 16-bit integers, and store the results in dst.
17	vpshuflw xmm3, xmm3, 85	c = r→dif>>16
18	punpcklqdq xmm3, xmm3	c = (unsigned)(dif >> (OD_EC_WINDOW_SIZE - 16));
19	paddw xmm1, xmm2	v += EC_MIN_PROB * (N - ret);
20	movdqu [rsp + 24], xmm1	u = v
21	psubusw xmm1, xmm3	насичене віднімання c-v, якщо c-v.lane[i]<0, то xmm3.lane[i] = 0
22	pxor xmm2, xmm2	створюємо нульовий вектор
23	pcmpeqw xmm1, xmm2	Порівнюємо c-v з нулем та записуємо результат порівняння
24	pmovmskb eax, xmm1	Створюєсо маску із значень старших бітів доріжок вектора
25	ret	Повертаємо керування

Джерело: складено автором (колонка 1-2), репозиторій google aom, master branch, файл entdec.c (колонка 3)

Додаток Ґ

Таблиця Ґ Порівняння версій коду функції od_ec_decode_q15 на мові Asembler — застосуванням розширення набору інструкцій SIMD та коду, який згенерував компілятор

; код на мові ассемблер — обчислюються одночасно 8 елементів	C	Rec Thr	T ILP	; дизасемблер код на мові С — одночасно 1 елемент цикл	C	Rec Thr	T ILP
00 push rbp	3	0.5	3	$LN16:			
01 mov rbp, rsp	1	0.25	1	mov QWORD PTR [rsp+8], rbx	3	0.5	
02 sub rsp, 120	1	0.25	1	mov QWORD PTR [rsp+16], rbp	3	0.5	3
03 movq xmm3, [rcx + 32]	1	0.33	0	mov QWORD PTR [rsp+24], rsi	3	0.5	
04 lddqu xmm1, [rdx]	3	0.5	3	push rdi	3	0.5	6
05 neg r8	1	0.25	3	sub rsp, 32	1	0.25	1
06 pshuflw xmm2, xmm3, 170	1	0.5	4	movzx r9d, WORD PTR [rcx+36]	3	0.5	9
07 movd [rsp+20], xmm2	3	0.5	7	lea esi, DWORD PTR [r8-1]	1	0.5	7
08 punpcklqdq xmm2, xmm2	1	0.5	8	mov ebp, DWORD PTR [rcx+32]	3	0.5	12
09 movdqu xmm0, xmm1	0	0.2	8	mov r10d, r9d	1	0.25	10
10 vpsrlw ymm1, 6	1	0.5	9	mov r8d, ebp	1	0.25	13
11 vpsllw ymm1, 7	1	0.5	10	shr r10d, 8	3	1	13
12 vpsrlw ymm2, 8	1	0.5	9	shr r8d, 16	3	1	16
13 vpsllw ymm2, 8	1	0.5	10	mov rdi, rcx	1	0.25	7
14 pmulhuw xmm1, xmm2	5	0.5	15	add rdx, -2	1	0.25	1
15 lea rax, [rel array]	1	1	11	mov ebx, -1	1	0.25	4
16 movq xmm2, [rax+r8+16]	2	0.5	17	npad 6			
17 neg r8	1	0.25	11	$LL4@od_ec_deco:			
18 pmovsxbw xmm2, xmm2	1	0.5	18	movzx ecx, WORD PTR [rdx+2]	3	0.5	10
19 pshuflw xmm3, xmm3, 85	1	0.5	19	lea rdx, QWORD PTR [rdx+2]	1	0.5	2
20 punpcklqdq xmm3, xmm3	1	0.5	20	shr ecx, 6	3	1	13
21 paddw xmm1, xmm2	1	0.33	19	inc ebx	1	0.25	5
22 movdqu [rsp + 24], xmm1	2	0.5	21	imul ecx, r10d	3	1	16
23 psubusw xmm1, xmm3	1	0.33	22	mov eax, esi	1	0.25	8
24 pxor xmm2, xmm2	1	0.33	20	sub eax, ebx	1	0.25	9
25 pcmpeqw xmm1, xmm2	1	0.5	21	mov r11d, r9d	1	0.25	10
26 pmovmskb eax, xmm1	2	1	23	shr ecx, 1	3	1	19
27 ret				lea r9d, DWORD PTR [rcx+rax*4]	1	0.5	11
28				cmp r8d, r9d	1	0.25	12
29				jb SHORT $LL4@od_ec_deco	2	1	21
30				sub r11d, r9d	1	0.25	22
31				mov ecx, 15	1	0.25	22
32				bsr eax, r11d	3	1	25
33				shl r9d, 16	3	1	25
34				sub ebp, r9d	1	0.25	26
35				sub ecx, eax	1	0.25	26
36				sub WORD PTR [rdi+38], cx	7	1	33
37				inc ebp	1	0.25	27
38				shl ebp, cl	1	1	28
39				dec ebp	1	0.25	29
40				shl r11w, cl	1	1	27
41				cmp WORD PTR [rdi+38], 0	1	0.25	34
42				mov DWORD PTR [rdi+32], ebp	2	1	36
43				mov WORD PTR [rdi+36], r11w	2	1	38
44				jge SHORT $LN14@od_ec_deco	2	1	40
45				mov rcx, rdi	1	0.25	41
46				call od_ec_dec_refill			
47				$LN14@od_ec_deco:			
48				mov rbp, QWORD PTR [rsp+56]	3	0.5	44
49				mov eax, ebx	1	0.25	42
50				mov rbx, QWORD PTR [rsp+48]	3	0.5	45
51				mov rsi, QWORD PTR [rsp+64]	3	0.5	44
52				add rsp, 32	1	0.25	42
53				pop rdi	3	0.5	45

```
54                                  ret         0
```

Джерело: Код складено автором, кількість тактів та пропускна здатність інструкції із Agner Fog instruction tables [Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD, and VIA CPUs - Agner Fog. Technical University of Denmark, 2022. Last updated 2022-06-11]

Додаток Д

ТаблицяД Представлення коду на мові Assembler ентропійного декодування із використанням набору розширень AVX2 аналогічного до скалярної версії функції за замовчуванням update_cdf

; код на мові ассемблер — обчислюються одночасно 8 елементів	; код на мові С — обчислюються одночасно 1 елемент перебором у циклі do-while
00 mov r10w, word [rdx + r8 * 2]	cdf[nsymbs]
01 vpcmpeqw ymm2, ymm2	Load icdf[++ret] into buf
02 mov r11d, r10d	nsymbs = -nsymbs
03 shr r10d, 4	r = dec->rng;
04 add r10d, 5	v = r
05 cmp r11d, 32	-
06 adc r11d, 0	u = v
07 movd ymm3, r10d	Rate
08 vpavgw ymm2, ymm1	1<<15 або 0
09 vpsubw ymm2, ymm0	1<<15 - cdf[i] чи max - cdf[i]
10 vpsubw ymm0, ymm1	cdf[i] – 0 чи cdf[i] - max
11 vpsraw ymm2, xmm3	1<<15 – cdf[i]>>rate чи (max-cdf[i])>>rate
12 vpaddw ymm0, ymm2	cdf[i] + 1<<15 – cdf[i]>>rate чи cdf[i] – (max – (max–cdf[i])>>rate) => cdf[i]=-cdf[i]<<rate
13 mov [rdx + r8*2], r11w	store cdf[nsymbs]
14 movdqu [rdx], xmm0	store cdf[0..nsymbs-1]
15 ret	

Джерело: складено автором

Додаток Е

Таблиця Е Порівняння версій коду функції update_cdf на мові Asembler — власноруч із застосуванням розширення набору інструкцій SIMD та коду, який згенерував компілятор у версії Release:

; код на мові ассемблер — обчислюються одночасно 8 елементів	C	Rec Thr	T ILP
00 mov r10w, word [rdx + r8 * 2]	2	0.5	
01 pcmpeqw xmm2, xmm2	1	0.5	
02 mov r11d, r10d	1	0.25	
03 shr r10d, 4	1	0.5	
04 add r10d, 5	1	0.25	
05 cmp r11d, 32	1	0.25	
06 adc r11d, 0	1	1	
07 movd xmm3, r10d	2	1	
08 vpavgw ymm2, ymm1	1	0.5	
09 vpsubw ymm2, ymm0	1	0.33	
10 vpsubw ymm0, ymm1	1	0.33	
11 vpsraw ymm2, xmm3	1	1	
12 vpaddw ymm0, ymm2	1	0.33	
13 mov [rdx + r8*2], r11w	2	1	
14 movdqu [rdx], xmm0	2	0.5	
15 ret			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			
26			
27			
28			
29			
30			
31			
32			
33			
34			
35			
36			
37			
38			
39			
40			
41			
42			
43			
44			
45			
46			
47			
48			
49			
50			
51			

; дизасемблер код на мові С — одночасно 1 елемент цикл		C	Rec Thr	T ILP
mov	rax, rsp			
mov	QWORD PTR [rax+8], rbx	3	0.5	
mov	QWORD PTR [rax+16], rbp	3	0.5	
mov	QWORD PTR [rax+24], rsi	3	0.5	
mov	QWORD PTR [rax+32], rdi	3	0.5	
xor	ebp, ebp	1	0.25	
movsxd	rsi, r8d	3	0.5	
mov	eax, ebp	1	0.5	
mov	r11d, ebp	3	0.5	
mov	r10, rcx	1	0.25	
mov	r9d, ebp	1	0.25	
cmp	WORD PTR [rcx+rsi*2], 31	3	1	
seta	r11b	3	1	
cmp	WORD PTR [rcx+rsi*2], 15	1	0.25	
seta	al	1	0.25	
add	r11d, eax			
mov	eax, ebp			
cmp	esi, 1			
setg	al			
add	eax, 3			
add	r11d, eax			
mov	eax, ebp			
cmp	esi, 3			
setg	al			
add	r11d, eax			
movsx	eax, dl			
test	dl, dl			
jle	SHORT $LN3@update_cdf			
mov	r8, rcx			
mov	r9d, eax			
mov	edi, eax			
$LL13@update_cdf:				
movzx	edx, WORD PTR [r8]			
mov	eax, 32768			
sub	eax, edx			
mov	ecx, r11d			
sar	eax, cl			
add	ax, dx			
mov	WORD PTR [r8], ax			
lea	r8, QWORD PTR [r8+2]			
sub	rdi, 1			
jne	SHORT $LL13@update_cdf			
lea	ecx, DWORD PTR [rsi-1]			
cmp	r9d, ecx			
jge	SHORT $LN6@update_cdf			
movsxd	rax, r9d			
sub	ecx, r9d			
mov	r9d, ecx			
lea	r8, QWORD PTR [r10+rax*2]			
$LL7@update_cdf:				
movzx	edx, WORD PTR [r8]			
movzx	ecx, r11w			

```
52                                  movzx   eax, dx
53                                  shr     ax, cl
54                                  sub     dx, ax
55                                  mov     WORD PTR [r8], dx
56                                  lea     r8, QWORD PTR [r8+2]
57                                  sub     r9, 1
58                                  movzx   eax, WORD PTR [r10+rsi*2]
59                                  mov     rbx, QWORD PTR [rsp+8]
60                                  cmp     ax, 32
61                                  mov     rdi, QWORD PTR [rsp+32]
62                                  adc     ax, bp
63                                  mov     rbp, QWORD PTR [rsp+16]
64                                  mov     WORD PTR [r10+rsi*2], ax
65                                  mov     rsi, QWORD PTR [rsp+24]
66                                  ret
```

Джерело: код складено автором, кількість тактів та пропускна здатність інструкції із Agner Fog instruction tables [Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD, and VIA CPUs - Agner Fog. Technical University of Denmark, 2022. Last updated 2022-06-11]

Додаток Є

Профілювання коду у linux до і після оптимізації

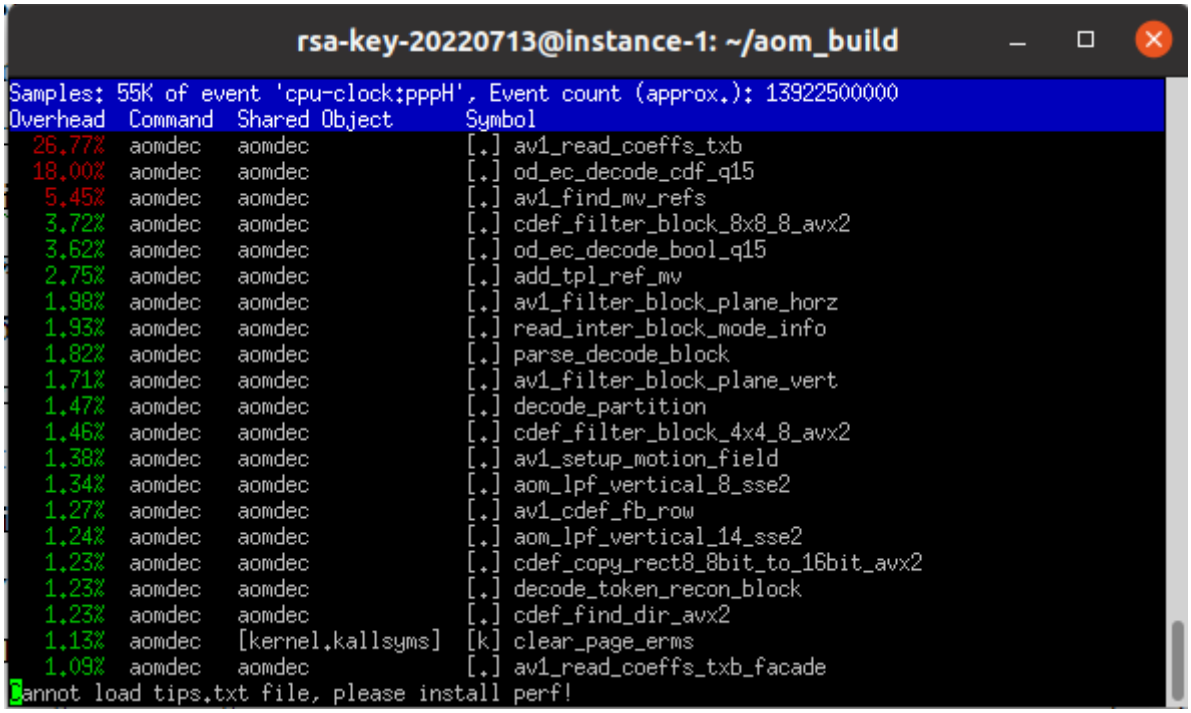

Рис. Є.1 Оптимізована SIMD версія:кількість тактів на функцію ентропійного декодування.

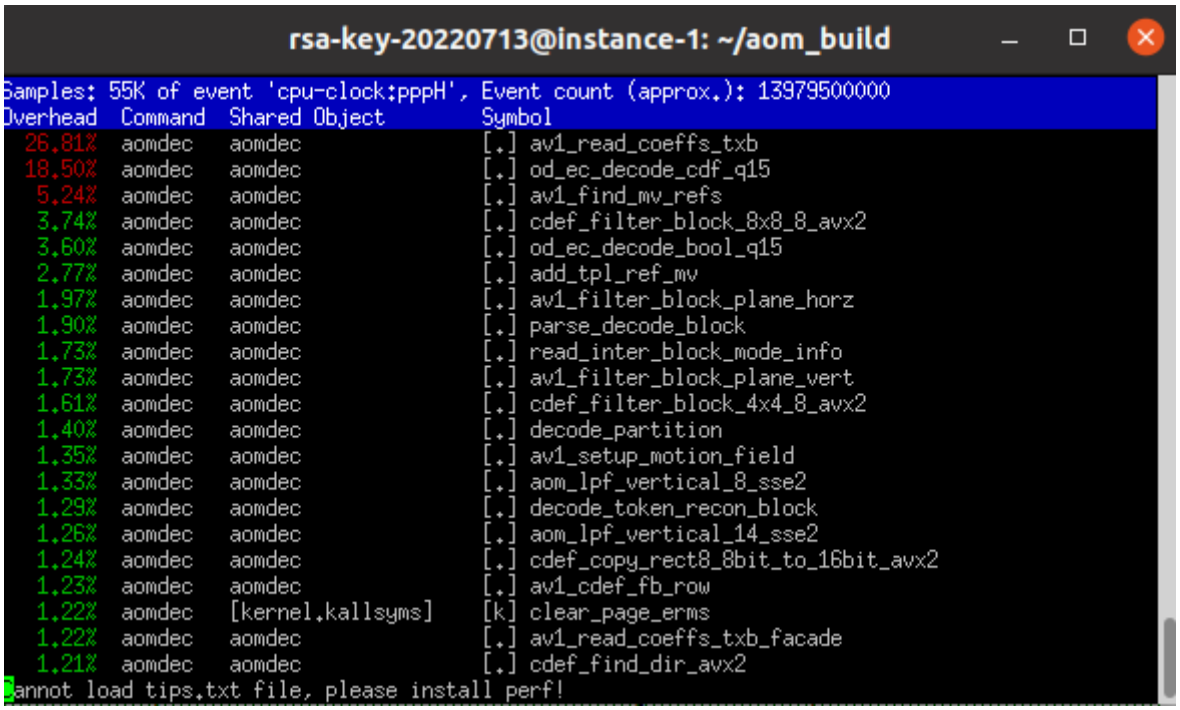

Рис. Є.2 Версія SIMD за замовчуванням: кількість тактів на функцію ентропійного декодування.