Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА

на тему:

Комп'ютерна аналого-цифрова система опрацювання низькочастотних сигналів з використанням CUDA-технологій

ПОЯСНЮВАЛЬНА ЗАПИСКА

Виконав: студент 4 курсу, групи
1КІ-18б
напряму підготовки (спеціальності)
123 – «Комп'ютерна інженерія»
(шифр і назва напряму підготовки, спеціальності)
Савчук Д.А.
(прізвище та ініціали)
Керівник Крупельницький Л.В.
(прізвище та ініціали)
Рецензент Лукічов В.В.
(прізвище та ініціали)

Допущено до захисту
д.т.н., проф. Азаров О.Д.

" 21 " червня 2022 р.

ВНТУ 2022

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 123 — «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач кафедри

обчислювальної техніки

проф. Азарову О.Д.

«08» 02 2022 р.

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Савчуку Дмитру Анатолійовичу

1 Тема роботи: «Комп'ютерна аналого-цифрова система опрацювання низькочастотних сигналів з використанням CUDA-технологій»

Керівник роботи к.т.н., доц. каф.ОТ Крупельницький Леонід Віталійович

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "24" 03 2022 року № 66

2 Строк подання студентом проекту (роботи) 21.06.2022р.

3 Вихідні дані до проекту (роботи) — чотири масива даних конвертовані із чотирьох канального звукового файлу з частотою дискритизації 48 кГц. розрядністю 16-24 двійкових розряди.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, огляд інсуючих методів і засобів акустичної локації та ідентифікації об'єктів на місцевості, аналіз технології обробки низькочастоних звукових сигналів при багатоканальному скануванні в системі мікрофонної решітки, розробка та тестування програмного забезпечення для обчислення низькочастотних сигналів за допомогою CUDA-ядер, та використання бібліотек для роботи із ядрами графічного процесора, висновки, перелік джерел посилання, додатки.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): конструкція мікрофонних решіток стаціонартного та портативного типу, структрурна схема АЦ-системи аудіолокації та ідентифікації об'єктів на місцевості

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1 — Консультанти розділів таблиці

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1–3	к. т. н., доцент каф. ОТ Крупельницький Л.В.		

7 Дата видачі завдання 05 вересня 2021 року.

8 Календарний план приведені в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів проекту (роботи)	Примітка
1	Постановка задачі роботи	09.09.2021	виконано
2	Огляд існуючих методів і засобів акустичної локації та індефікації об'єктів на місцевості	12.09.2021— 08.10.2021	виконано
3	Аналіз технології обробки низькочастоних звукових сигналів при багатоканальному скануванні в системі мікрофонної решітки	09.10.2021— 06.11.2021	виконано
4	Огляд та аналіз технології CUDA для багатопоточних обчислень	07.11.2021— 25.11.2021	виконано
5	Пошук та аналіз технологій для роботи з графічним процесором, та CUDA-ядрами за допомогою мови програмування C#	26.11.2021— 22.12.2021	виконано
6	Розробка програмного забезпечення для обчеслення низькочастотного сигналу за допомогою CUDA-ядер.	22.12.2021— 04.02.2022	виконано
7	Оформлення пояснювальної записки та ілюстративного матеріалу	04.02.2022— 03.03.2022	виконано
8	Аналіз виконання роботи, висновки, додатки	04.03.2022— 27.03.2022	виконано
9	Перевірка якості виконання бакалаврської роботи та усунення недоліків	28.03.2022	виконано

Студент _____ Савчук Д.А
(підпис) (прізвище та ініціали)

Керівник роботи _____ Крупельницький Л.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Савчук Д.А. Комп’ютерна аналого-цифрова система опрацювання низькочастотних сигналів з використанням CUDA-технологій. Бакалаврська кваліфікаційна робота зі спеціальності 123 — Комп’ютерна Інженерія, Вінниця: ВНТУ, 2022. Пояснювальна записка містить 67 сторінки, 16 рисунків та 19 посилань.

У бакалаврській дипломній роботі було розглянуто існуючі методи та підходи обробки низькочастотних сигналів. Аналізуються методи і засоби акутичної локації та індефікації об’єктів. Досліджено системи багакоканального сканування за допомогою мікрофонних решіток. Було розлянуто технологію CUDA, та бібліотеки для ви-користання CUDA-ядер в мові програмування C#. Було розроблено програмне забезпечення для опрацювання низькочастотних сигналів з використанням технологій CUDA

Ключові слова: CUDA-технології, CUDA-ядра, низькочастотний сигнал, мікрофонна решітка.

ABSTRACT

Savchuk D.A. Computer analog-digital system for processing low-frequency signals using CUDA-technologies. Bachelor's degree in specialty 123 — Computer Engineering, Vinnytsia: VNTU, 2022. The explanatory note contains 67 pages, 16 figures and 19 references.

In the bachelor's thesis the existing methods and approaches of low-frequency signal processing were considered. Methods and means of acute location and indexing of objects are analyzed. Multichannel scanning systems using microphone arrays have been studied. CUDA technology and libraries were used to use CUDA cores in the C # programming language. Software developed for processing low-frequency signals using CUDA technologies

Keywords: CUDA-technologies, CUDA-cores, low-frequency signal, microphone array.

ЗМІСТ

ВСТУП

Актуальність теми полягає у тому, що на сьогоднішній день у промислових, силових, охоронних організаціях використовуються інформаційно-розпізнавальні системи. Все це на пряму пов’язано з потоковою обробкою інформації, що впирається в швидкістю її обробки.

Вітчизняні аналоги даної розробки відсутні, а інформація про схожі зарубіжні системи потокової обробки є конфіденційною, або ж занадто загальною, не конкретизованою за технічними параметрами й методами їх отримання.

Запропонований метод потокової обробки відцифрованого низькочастотного сигналу є новим. Сигнал обробляється за допомогою технології CUDA, що являє собою технологію паралельних обчислень, що в свою чергу дозволяє значно збільшити швидкість обчислювальних процесів, завдяки використання графічного процесора від компанії Nvidia.

Також дана технологія надає можливість виклик в тексті програми на мові C# певних підпрограм, що виконуються в графічних процесорах Nvidia.

Виходячи із розглянутого, розробка даної системи є актуальною задачею. Сфера використання систем для розпізнавання з потоковою обробкою інформації вказує на актуальність даної проблеми та вимагає подальших досліджень.

Метою дослідження є розробка системи та програмного забезпечення до неї для потокової обробки інформації з метою використання у різних галузях, зокрема в статичних і портативних комп’ютерних системах, що забезпечать можливість обробки потокового низькочастотного сигналу за допомогою технології CUDA-ядер.

Для цього необхідно виконати такі **задачі**:

— здійснити аналіз галузі застосування системи;

— класифікувати методи розпізнавання звукових сигналів;

— розробити систему розпізнавання низькочастотних сигналів;

— розробити програмне забезпечення для обробки потокового низькочастотного сигналу;

— провести діагностику роботи системи та програмного забезпечення;

— провести тестування розробленої системи та програмного забезпечення;

Об’єктом дослідження є процеси, обробки низькочастотних сигналів в потоці часу

Предмет дослідження є методи та засоби опрацювання низькочастотних сигналів з використанням технології CUDA

Методи дослідження дипломної роботи: використовувався та досліджувався метод обробки низькочастотних сигналів і наведенні висновки про їх потокову обробку за допомогою технології CUDA;

Публікація за темою роботи — Комп'ютерна аналого-цифрова система опрацювання низькочастотних сигналів з використанням CUDA-технологій [Текст] / Д. А. Савчук // LI Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2022): Тез. доп. — Вінниця, 2022. — 1. Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/14885 (дата звернення 09.06.22).[1]

1 ОГЛЯД ІСНУЮЧИХ МЕТОДІВ І ЗАСОБІВ АКУСТИЧНОЇ ЛОКАЦІЇ ТА ІНДЕНТИФІКАЦІЇ ОБ'ЄКТОВ НА МІСЦЕВОСТІ.

1.1 Аналітичний огляд структурних і обчислювальних методів, апаратних і програмних засобів акустичної локації та ідентифікації об'єктів

У сучасному світі масово використовуються технології акустичної локації та ідентифікації об'єктів, які частково запозичені з радіолокаційних, гідролокаційних та сейсмічних – електронне сканування в каналах мікрофонної решітки, кореляційна обробка сигналів. Ефективність аудіолокації в основному визначається не тільки точністю та ідентичністю характеристик передачі вимірювальних каналів систем, а і швидкістю опрацювання сигналу. [2]

На входах вимірювальних каналів ефективно використовувати струмове посилення аналогових сигналів, що забезпечує низьку чутливість до зовнішніх перешкод і шумів.

Для досягнення точності та ідентичності вимірювальних каналів, крім самокалібрування аналого-цифрового підсилювача, буде використовуватися визначення та корекція динамічних характеристик електричних вимірювальних каналів та акустичних характеристик мікрофонів [3]. Метод самокалібрування передбачає, що лише окремі некориговані параметри (шум, швидкість) оптимально оптимізовані в каналах вимірювання. Деякі інші параметри (зміщення, коефіцієнт посилення, форма амплітудно-фазо-частотної характеристики, відмінності між каналами) можна визначити та відрегулювати чисельно. Поширення принципів калібрування та самонастроювання на вимірювальні канали систем змінного струму дає змогу покращити їх характеристики порівняно з відомими технічними рішеннями.[4]

Продуктивність класичних багатоканальних електронних алгоритмів сканування обмежена кількістю послідовних зсувів, обчисленнями сум кореляції сигналів і часто не забезпечує режим реального часу, а методи розпаралелювання не враховують специфіку сигналів.

Багатоканальна цифрова обробка сигналів у системі змінного струму для кожного з дискретно заданих напрямків і координат об'єкта для завдань багатоканальної

обробки може ефективно використовувати паралельні паралелі на основі нейроноподібних ієрархічних структур та з використанням графічних процесорів GPU із використання технології багатоканальної обробки CUDA.

Через апаратні помилки розташування, вплив зовнішніх факторів і переміщення об'єктів координати джерел сигналу можуть бути розмитими. Зворотний логічний висновок може бути використаний для уточнення дискретних координат об'єктів поля, що зводиться до розв'язування системи нечітких логічних рівнянь. Однак переважна більшість методів вирішення таких систем є аналітичними. Однак визначення множини мінімальних рішень залишається невирішеною проблемою.

Ідея використання методів нечіткої логіки полягає в тому, щоб підвищити роздільну здатність реальних координат об'єктів шляхом вирішення оберненої задачі ідентифікації. Можливе застосування методу розкладання багатовимірних рівнянь за принципом приросту.

Під час розпізнавання звукові сигнали представлені такими параметрами в частотній області, як коефіцієнти лінійного прогнозування, захищені коефіцієнти, піки спектральних частот. Для прийняття рішень в основному використовуються змішані, гауссові та приховані моделі Маркова. Також спостерігається значне зниження надійності розпізнавання сигналу при погіршенні відношення сигнал/шум [5].

При цьому розпізнавання фрагментів звукового сигналу, отриманих шляхом сканування кожного вузла сітки матриці, можна ефективно реалізувати на основі оригінального методу швидкого пошуку дерева в k-вимірному графі, представленими наборами параметрів опорного звукового фрагмента. Використання спектральних частотних піків як параметрів представлення аудіосигналу збільшить ймовірність правильного розпізнавання типу джерела звуку з низьким відношенням сигнал/шум.

Для підсистеми аутентифікації «один для одного» актуальним є використання прихованих (зашумлених) коротких пакетів, в яких ефективне криптокодування. Існує алгоритм поєднання потокового шифрування та шумостійкого кодування відеосигналів БПЛА за допомогою найпростішого методу криптозахисту – гейміфікації. Ці

та інші підходи вимагають значного додаткового апаратного та програмного шифрування. У той же час у вітчизняних та зарубіжних відкритих виданнях практично немає конкретної інформації про те, як побудувати системи криптографічного захисту інформації в спеціалізованих системах ідентифікації об’єктів у цій галузі.

Після розпізнавання звукових об’єктів їх можна ідентифікувати підсистемою. Враховуючи призначення системи, радіосигнал запиту на розпізнавання повинен мати низьку амплітуду та мінімальну тривалість. При цьому він повинен мати достатній рівень захисту.

1.2 Загальна структурна схема, склад та принцип роботи АЦ-системи аудіолокації й ідентифікації об’єктів на місцевості

Спеціа лізована система змінного струму для аудіолокації та ідентифікації об'єктів розроблена на основі багатоканальної обробки сигналів акустичних мікрофонних масивів. Акустичне розташування об’єктів та дистанційний збір інформації є дуже важливими в задачах автоматизованого контролю доступу та охорони територій, у комп’ютерних системах ідентифікації джерел звуку, у мультимедійних системах стадіонів, концертних залів, виставкових залів, тощо.

Розробку систем дистанційного збору та обробки акустичних даних можна розділити на три етапи:

— «механічний» етап настав на початку 20 століття, коли для виявлення літаків і розташування артилерійських батарей використовувалися спрямовані звукові трубки та відбивачі;

— етап обробки аналогового сигналу збігся з розвитком електронної технології та першим застосуванням мікрофонних решіток для збору акустичної інформації;

— етап потокової цифрової обробки сигналу. З 1990-х років інструменти цифрової обробки широко використовуються для вирішення широкого кола завдань, локалізації джерела звуку, його аналізу, в слухових апаратах та для дистанційного виявлення динаміка; виділення заданого звуку з шуму тощо [6].

Мікрофонна решітка – це один із типів спрямованих мікрофонів, реалізований у вигляді набору звукоприймачів, які працюють узгоджено (по фазі чи з фазовими затримками) і розташовані всередині чи зовні.

Принцип дії багатоканальних мікрофонних масивів заснований на цифровому підсумовуванні сигналів у каналах з введенням часових зрушень, необхідних для формування діаграми. Акустичний сигнал, показаний на рисунку 1.1, потрапляє на мікрофонну решітку, посилюється за допомогою попереднього підсилення низької частоти, перетворюється в цифрову форму за допомогою багатоканальних аналого-цифрових перетворювачів і передається на пристрій. В свою чергу персональний комп'ютер виконує необхідну обробку цифрового аудіосигналу з подальшим перетворенням в бітовий масив для обробки.

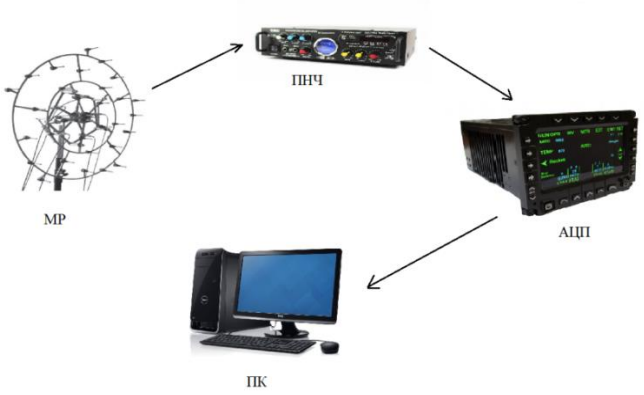


Рисунок 1.1 — Проходження акустичних сигналів у комп'ютерній системі з «цифровою» мікрофонною решіткою

Затримки виникають, коли акустичний сигнал досягає мікрофонної решітки через напрям джерела сигналу та швидкість звукової хвилі. Вибір сигналу з потрібного напрямку можливий після введення в кожен вимірювальний канал необхідної затримки, як показано на рисунку 1.2.

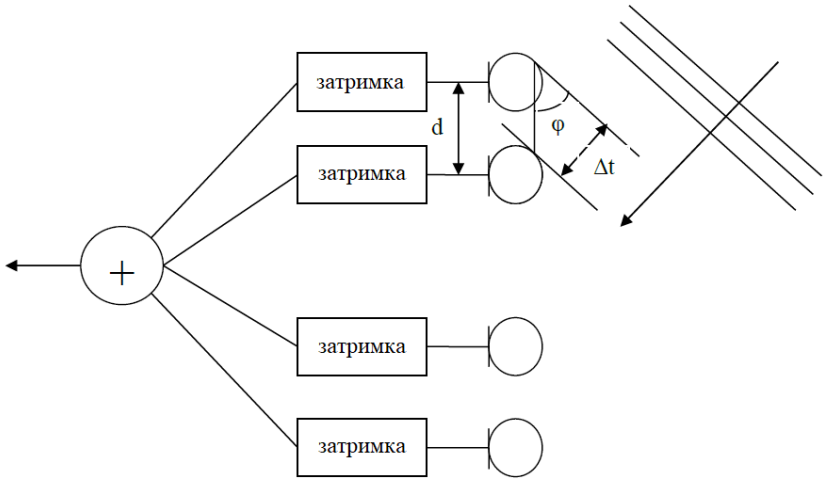


Рисунок 1.2 – Затримки сигналів для виділення напрямку на джерело звуку

Нехай мікрофонна решітка складається з N приймачів, d — відстань між сусідніми приймачами. Потім, щоб задати напрямок основної пелюстки візерунка під кутом φ, зсув часу між сусідніми мікрофонами має дорівнювати:

$$\Delta t = d * \sin(\varphi) / v, \quad (1.1)$$

де v – швидкість розповсюдження звуку в повітрі.

Під час обробки сигналів, прийнятих мікрофонною мережею, для виділення різних кутів напрямків φ, необхідно для кожного кута підсумувати вибірки сигналів, взяті з відповідними часовими затримками. Найпростіший спосіб ввести часові затримки між сигналами - зсунути їх на певну кількість відліків m, кількість яких пов'язана з частотою дискретизації:

$$\Delta t = m / Fd. \quad (1.2)$$

Кут φ діаграми направленості мікрофонної решітки при однаковій відстані між мікрофонами d залежить від кількості зсувів між дискретними відліками сусідніх сигналів перед підсумовуванням:

$$\varphi = \arcsin(m * v / (d * Fd)). \quad (1.3)$$

Крок сканування по дискретних відліках кута φ зменшується якщо частота дискретизації збільшується .

Послідовні значення кута φ при m = … , -3, -2, -1, 0, +1, +2, +3, … є рівновіддаленими тільки при малих значеннях m, коли наближено виконується рівність $\varphi = arcsin(\varphi)$. Однак, ця особливість не є принциповою при електронному скануванні місцевості: роздільна здатність зменшуватиметься на більших кутах, проте їх точні значення відповідатимуть співвідношенню.

Розглянемо деякі конструктивні параметри мікрофонних решіток. Зауважимо також, що відстань d між мікрофонами не може бути довільно великою, оскільки потрібно враховувати довжину періоду звукової хвилі сигналу частотою, яка визначається швидкістю звуку в повітрі:

$$Ds = v / Fs. \qquad (1.4)$$

Так, при v = 330 м/с і частоті звукового сигналу Fs = 3300 Гц довжина періоду сигналу складе 0.1 м (100 мм), що означатиме періодичне повторення сигналу, який приходитиме з кутів $\varphi = 180^o$ та $\varphi = -180^o$, якщо відстань між мікрофонами складатиме 100 мм. Це означатиме, що на частоті Fs = 3300 Гц утвориться дві бокових пелюстки діаграми направленості, рівні центральній на куті $\varphi = 0^o$.

З огляду на вищевказану інформацію та з урахуванням мети і основних бакалаврської роботи, було вирішено використати спосіб роботи системи, який ілюструє структурна система, що наведена на рисунку 1.3.

Аудіосигнали приймає одна мікрофонна решітка. Попередньо посилені аналогові сигнали в кожному з вимірювальних каналів перетворюються в цифрову форму за допомогою високоточного АЦП [7].

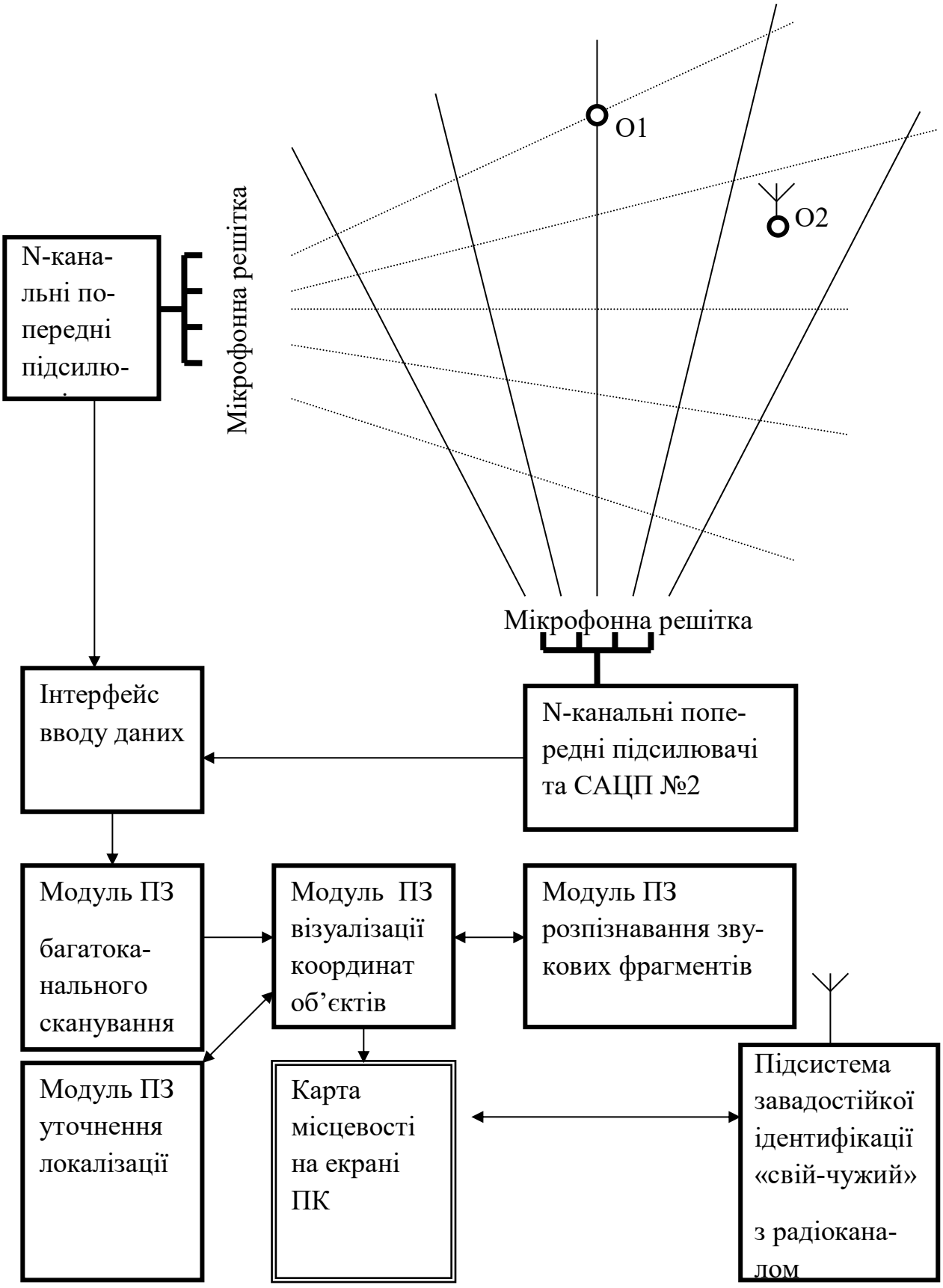


Рисунок 1.3 – Структрурна схема АЦ-системи аудіолокації та ідентифікації об'єктів на місцевості

1.3 Аналіз методів побудови і принципів функціонування мікрофонних решіток

В наші дні мікрофонні решітки використовуються в таких галузях, як дистанційний прийом аудіосигналів або збір голосових сигналів з виділенням потрібного голосу із багатьох, голосове управління в автомобілях і в інших системах голосового управління пристроями, зменшення шуму при розпізнаванні мови (наприклад, в слухових апаратах), а також в різних військових цілях .

Мікрофонна решітка є сукупністю кількох мікрофонів, які працюють узгоджено. Зміна фазових затримок сигналу, які приходять на різні мікрофони, дозволяє отримати просторову вибірковість прийому сигналу. Виходом мікрофонної решітки є суматор. Амплітуди і фази сигналів, що надходять з приймачів звуку на суматор, регулюються за допомогою вагових функцій. Таке регулювання реалізується програмно, що дозволяє без зміни конструкції або без пересування решітки отримати будь-яку діаграму направленості.

Електронні сигнали кожного з мікрофонів містять інформацію про звуки, що приходять з усіх напрямків. Спільна обробка цих сигналів дозволяє виділити звук, який надходить з заданого напрямку. Таким чином, мікрофонна решітка виділяє звук з заданого напрямку не самі по собі, а шляхом обробки багатоканального сигналу.

Базовими структурами МР є так звані Broadside array і Endfire array. У цьому сенсі це аналоги мікрофонів з параболічними відбивачами і трубчастих спрямованих мікрофонів.

При вирішенні завдань дистанційного збору аудіо інформації найбільшого поширення набули мікрофонні решітки типу Broadside, тому далі розглянемо саме цю структуру.

Проаналізуємо базовий алгоритм формування діаграми спрямованості. Зазвичай, сигнал цільового джерела інформації передбачається когерентним сигналом точкового джерела, шум в залежності від акустичної обстановки є акустичне поле з різними просторовими характеристиками.

Таблиця 1.1 – Порівняння видів мікрофонних решіток

Вид решітки	Переваги	Недоліки
Broadside	Плоска геометрія. Проста реалізація обробки. Можливість управління напрямком променю.	Менше придавлення поза віссю МР. Мінімальна відстань між мікрофонами та їх велика кількість, що необхідно для запобігання просторовому витоку.
Endfire	Краще придавлення поза віссю. Менший загальний розмір.	Неплоска (об'ємна) геометрія. Більш складна обробка. Придавлення корисного сигналу в діапазоні низьких частот. Для двовимірних решіток формування променю можливо тільки в горизонтальному напрямку (площині решітки).

Для виділення прямого звуку об’єкта основним (базовим) алгоритмом обробки сигналів МР є алгоритм формування діаграми ДС, варіантами якого є алгоритми затримки і підсумовування (delay-and-sum) та фільтрації і підсумовування (filter-and-sum).

Сенс цих алгоритмів полягає в тому, що сигнали мікрофонів складаються з різними затримками (різним зсувом фаз), вирівнюючи для кожної частоти фази сигналів, що приходять з обраного напряму (цільового об’єкту). У цьому випадку алгоритм формування ДС дозволяє підсилити сигнали, що формуються звуком, який приходить з обраного напрямку, тобто здійснює своєрідне «фокусування» звуків. Зсуви фаз можуть задатися, виходячи з наближення «далекого поля», для якого звукові хвилі вважаються плоскими, або «ближнього поля», для якого звукові хвилі вважаються сферичними. Дистанції, для яких справедливо наближення «далекого поля», визначаються за виразом[8]:

$$L > 2D^2/\lambda,$$

де L — відстань від джерела до МР;

D – лінійний розмір (апертура) МР;

λ – довжина звукової хвилі.

Основною характеристикою МР є діаграма спрямованості, що описує вибірковість МР. Діаграма спрямованості відображає просторову залежність передавальної функції МР, орієнтованої в заданому напрямку від кута приходу сигналу і, таким чином, характеризує ефективність виділення сигналу від цільового джерела і придавлення когерентних перешкод.

Діаграма спрямованості залежить від геометрії МР (числа і розташування мікрофонів) і алгоритмів обробки сигналів. На рисунку 1.5 наведено діаграми спрямованості еквідистантної восьмиелементної мікрофонної лінійки з апертурою 35 см, d = 5 см для (1, 2, 4) кГц. Горизонтальна вісь – кут приходу акустичного сигналу, вертикальна – амплітуда вихідного сигналу МР в лінійному масштабі.

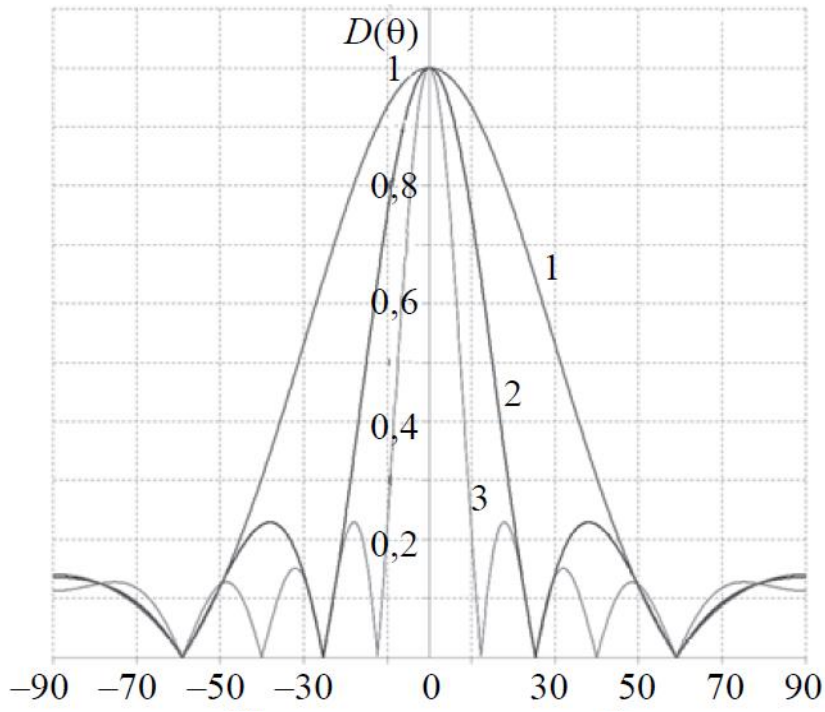


Рисунок 1.5 — Залежність діаграми спрямованості еквідистантної восьмиелементної мікрофонної решітки: 1 – 1 кГц; 2 – 2 кГц; 3 – 4 кГц

Найбільш важливими параметрами ДС є ширина основної пелюстки (за рівнем 3 дБ), рівень бічних пелюсток (або відношення рівня основної пелюстки до рівня бічних) і так званий «індекс спрямованості», що характеризує здатність МР придавлювати ізотропний шум, що приходить з усіх напрямків. Всі характеристики МР залежать від частоти сигналу. У таблиці 1.2 наведена залежність ширини основної пелюстки від частоти сигналу для восьмиелементної мікрофонної лінійки (8 см × 5 см).

Таблиця 1.2 — Ширина основної пелюстки діаграми направленості залежно від частоти

Частота, кГц	1 кГц	2 кГц	3 кГц	4 кГц	5 кГц	6 кГц	8 кГц
Ширина променю	±30	° ±15°	±10°	±7,5°	±6	° ±5°	±3,75°

Кутова роздільна здатність МР залежить від апертури (лінійного розміру МР), довжини хвилі (частоти сигналу), напрямку променя і методу обробки сигналу [27]:

$$\Delta\alpha = 1{,}22/\cos(\alpha)\times\lambda/D,$$

де $\Delta\alpha$ – кутова роздільна здатність (радіан); D – апертура; λ – довжина хвилі;

α – кут напрямку променю.

Просторова роздільна здатність в площині, що знаходиться на дистанції L, залежить від кута напряму променю наступним чином [29]:

$$X(\Delta\alpha) = 1{,}22/\cos^3(\alpha)\times\lambda L/D.$$

У таблиці 1.3 наведено залежність просторової роздільної здатності від кута напрямку променю і частоти сигналу для 8-елементної МР (8 × 5) см в площині, розташованій на відстані 10 м від лінійки.

Отже, аналіз методів побудови і основних параметрів МР показав, що вони знаходять все більше застосування в різних системах обробки аудіо інформації, зокрема, дистанційного збору аудіо інформації. Функціональні можливості МР визначаються

їх геометрією і алгоритмами цифрової обробки сигналів. Базові принципи роботи МР і алгоритми обробки їх сигналів викладені в ряді монографій, проте даний напрямок продовжує активно розвиватися.

Таблиця 1.3 — Просторова роздільна здатність (м) в площині, розташованій навідстані 10 м від мікрофонної решітки

Частота, кГц	Кут направлення променю						
	0°	10°	20°	30°	40°	50°	60°
1	11,86	12,10	14,23	18,26	26,33	44,59	94,88
2	5,93	6,05	7,11	9,13	13,16	22,39	47,44
4	2,97	3,02	3,56	4,56	6,58	11,15	23,72
6	1,98	2,02	2,37	3,04	4,39	7,43	15,81

Оптимізація геометрії МР виконується на основі статистичного моделювання, аналітичний розв'язок задачі відсутній. Оптимізація геометрії МР за різними критеріями приводить до різних структур з нееквідистантним розташуванням мікрофонів. Ряд зарубіжних компаній пропонують системи на основі МР для вирішення широкого кола завдань, пов'язаних з обробкою мовленевих і аудіосигналів. Ефективність практичного застосування МР в значній мірі визначається наявністю інформації, що узагальнює досвід їх практичного застосування.

2 АНАЛІЗ ТЕХНОЛОГІЇ ОБРОБКИ НИЗЬКОЧАСТОТНИХ ЗВУКОВИХ СИГНАЛІВ ПРИ БАГАТОКАНАЛЬНОМУ СКАНУВАННІ В СИСТЕМІ МІКРОФОННОЇ РЕШІТКИ

2.1 Конструктивно-схемотехнічні рішення мікрофонних решіток

Мікрофонні решітки можуть будуватись за рівномірними та нерівномірними схемами. Перевагою нерівномірної решітки з центральними проміжками і відстанями між мікрофонами, кратними числам 1-2-1, що є спрощення алгоритму отримання заданих затримок між сигналами в каналах за рахунок зсуву відліків.

Практична реалізація нееквідистатної мікрофонної решітки по схемі 1-2-1 наведена на рис.2.1. Відстані між мікрофонами обрано 100 мм – 200 мм – 100 мм. При швидкості звуку 340 м/с на частоті 3300 Гц значення модуля 100 мм відповідає одному періоду довжини звукової хвилі. Отже, на всіх частотах, менших за 3300Гц, не будуть створюватись хибні промені діаграми направленості[9].

Обмеження спектру вхідних сигналів частотою до 3300 Гц можливе за рахунок застосування у всіх каналах цифрового нерекурсивного фільтра низька частота суттєво не впливає на якість розпізнавання.

Рисунок 2.1 — Конструкція 4-канальної мікрофонної решітки з центральним проміжком

Після фільтрації можливе зменшення частоти дискретизації до частоти 8000 Гц, що значно зменшить об’єм обчислень при визначенні амплітуд сигналів в координатних точках контрольованої території та пришвидшить виконання аудіодетекції сигналів[10].

2.2 Розрахункові співвідношення для сканування й обробки сигналів

Для розробки модулів сканування необхідно вирішити ряд геометричних задач, пов’язаних з розміщенням мікрофонних решіток A і B та усіх точок C (рис. 2.5) перетину променів їх діаграм направленості на контрольованій місцевості.

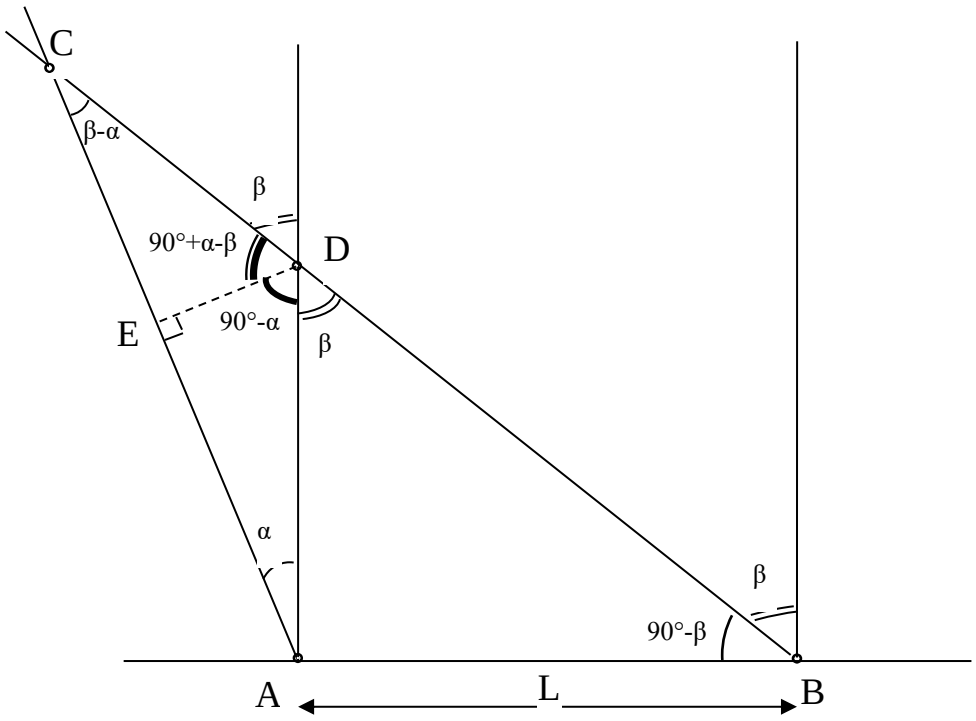


Рисунок 2.5 — Знаходження відстаней до точки C для «лівої» частини поля

Однією із задач які потрібно вирішити являється знаходження відстаней до точи перетину промінів.

Дано: $|AB|=L$ – відстань між мікрофонами, для прикладу, L=20 м;

α – кут напрямку променю з решітки в точці A, $\alpha \in [0°;75°]$;

β – кут з точки B, $\beta \in [0°;75°]$.

Знайти: відстані від решіток до точки C перетину променів $|AC|$, $|BC|$;

різницю відстаней $\Delta = |BC| - |AC|$.

Рішення задачі:

1) знаходимо суміжні і різницеві кути в трикутниках ABC, ADB, ADC на основі заданих кутів α і β (рис. 2.6);

2) $|BD| = L/\sin(\beta)$;

$|AD| = |BD| * \cos(\beta) = L/\tan(\beta)$;

3) $|AE| = |AD| * \cos(\alpha)$;

$|ED| = |AD| * \sin(\alpha)$;

4) $|CD| = |ED| / \sin(\beta-\alpha)$;

$|CE| = |CD| * \cos(\beta-\alpha) = |ED| / \tan(\beta-\alpha)$;

5) $|AC| = |AE| + |CE| = |AD| * \cos(\alpha) + |ED| / tg(\beta-\alpha) =$

$= |AD| * \cos(\alpha) + + |AD| * \sin(\alpha) / tg(\beta-\alpha) = |AD| * (\cos(\alpha) + \sin(\alpha) / tg(\beta-\alpha)) = L/tg(\beta) * (\cos(\alpha) + + \sin(\alpha) / tg(\beta-\alpha))$,

отже, відстань від решітки А до точки С є рівною $L/tg(\beta) * (\cos(\alpha)+\sin(\alpha)/ tg(\beta-\alpha))$;

6) $|BC| = |BD| + |CD| = L/\sin(\beta) + |ED| / \sin(\beta-\alpha) =$

$= L/\sin(\beta) + + |AD| * \sin(\alpha) / \sin(\beta-\alpha) = L/\sin(\beta) + L/tg(\beta) * \sin(\alpha)/\sin(\beta-\alpha) =$

$= L/\sin(\beta) * (1 + \cos(\beta) * \sin(\alpha)/\sin(\beta-\alpha))$,

отже, відстань від решітки А до точки С розраховується за формолою (2.2);

$|BC| = L/\sin(\beta) * (1 + \cos(\beta) * \sin(\alpha)/\sin(\beta-\alpha))$;

7) різниця відстаней від точки С до решіток А і В для врахування потрібної затримки звукового сигналу

$\Delta = |BC| - |AC|$

Перша задача полягає в розрахунку відстані до точки С перетину променів залежно від відстані L між решітками А і В та кутами повороту променів α і β відносно прямого напрямку. Різниця відстаней визначатиме затримку звукових сигналів між решітками.

Друга задача полягає в розрахунку координат X і У для усіх точок С перетину променів за заданим кроком зміни кутів α і β. Це необхідно для побудови акустичного поля звукових рівнів на контрольованій місцевості.

Для точок С, розміщених на «правій» частині контрольованого поля, розрахунки здійснюються згідно з рис. 2.6, що «симетричний» до попередньо розглянутого рис. 2.5. Кути α і β «замінюють» один одного. Знак затримки змінюється.

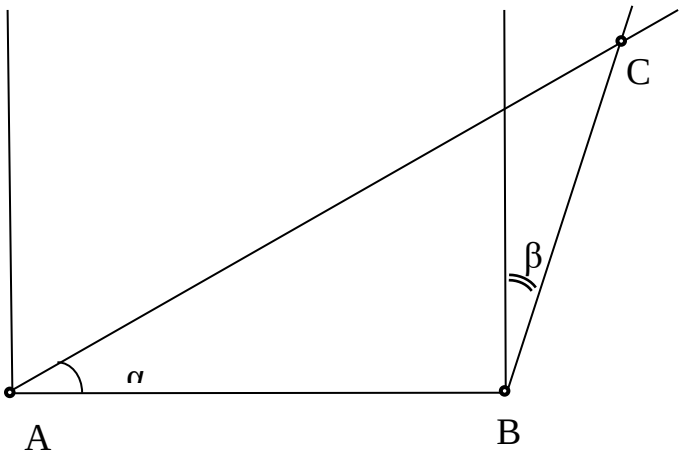


Рисунок 2.6 — Знаходження відстаней до точки С для «правої» частини поля

У випадку повороту променів решіток А і В на однакові кути, вони не перетинаються, а рішення згідно з рис. 2.6 спрощується.

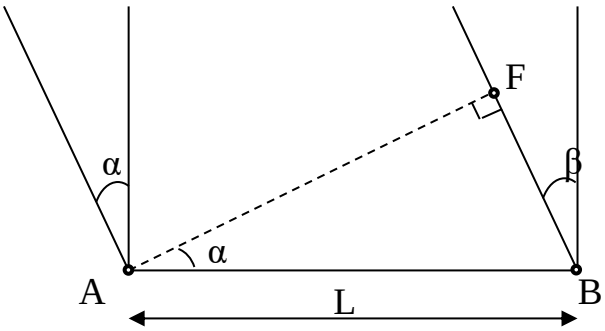


Рисунок 2.7 — Знаходження різниці відстаней при повороті на однакові кути

Якщо кути однакові α = β, то обчислюється тільки різниця відстаней – відрізок $|BF| = L*\sin(\alpha)$

На основі розрахованих співвідношень, для L= 20 м, і кроків кутів α і β по 5° в рамках від +75° («ліва» частина) до мінус 75° («права» частина) розраховано відстані $|AC|, |BC|$ і їх різницю Δ.

Другою задача являє собою розрахунок координатної матриці в значеннях X і У для кожної точки перетину променів.

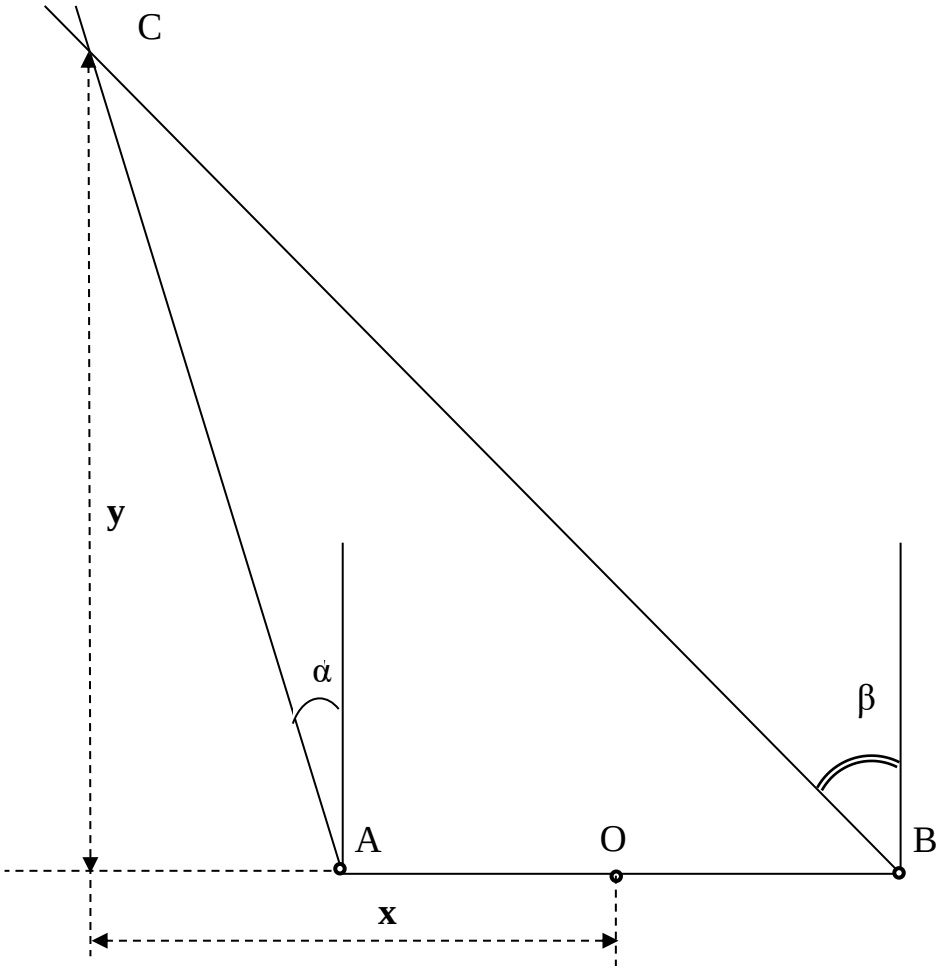


Рисунок 2.8 – Знаходження координат X і У точки С

За умовою задачі дано $|AB| = L = 20$ м, $|AO| = |OB| = 10$ м,

α, β, $|AC|$, $|BC|$

Потрібно знайти координати x,y кожної значущої точки на перехресті ліній з кутами α і β, в рішенні використовуються формули (2.5, 2.6);

Рішення задачі

1) $|AG| = |AC| * \sin(\alpha)$, де $X = |GO| = |AG| + L/2 = |AC| * \sin(\alpha) + L/2$;

2) $У = |GC| = |AC| * \cos(\alpha)$;

3) виконати п.1 і п.2 для всіх β [5°–75°] з кроком 5°;

4) для α= β точки повинні знаходитись на сторонах квадрата 250м х 250 м, що відповідає розмірам контрольованої території місцевості, як показано на рис. 2.9:

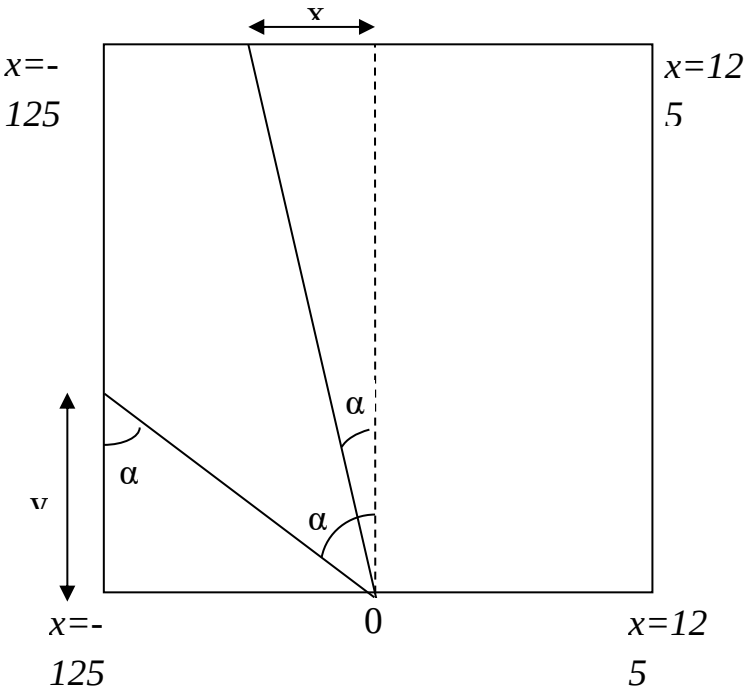


Рисунок 2.9 – Знаходження координат X і У точок, що знаходяться на сторонах квадрату контрольованої території

На основі розрахованих співвідношень, для L= 20 м і для кожної точки перетину променів α і β по 5° в рамках від плюс 75° («ліва» частина) до мінус 75° («права» частина) розраховано координати X і У усіх точок С [11].

3 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОБЧИСЛЕННЯ НИЗЬКОЧАСТОТНИХ СИГНАЛІВ ЗА ДОПОМОГОЮ CUDA-ЯДЕР. ВИКОРИСТАННЯ БІБЛІОТЕК ДЛЯ РОБОТИ ІЗ ЯДРАМА ГРАФІЧНОГО ПРОЦЕСОРА

3.1 Огляд технології CUDA

CUDA — це паралельна обчислювальна платформа та модель програмування, розроблена Nvidia для загальних обчислень на власних графічних процесорах (графічних процесорах). CUDA дозволяє розробникам пришвидшити обчислювальні програми, використовуючи потужність графічних процесорів для паралелізації частини обчислень. Хоча були запропоновані інші API для графічних процесорів, такі як OpenCL, і є конкурентоспроможні графічні процесори інших компаній, таких як AMD, комбінація графічних процесорів CUDA та Nvidia домінує в декількох областях застосування, включаючи глибоке навчання, і є основою для деяких найшвидші комп'ютери у світі. Відео карти, можливо, такі ж старі, як і ПК – тобто, якщо вважати монохромний дисплейний адаптер IBM 1981 року графічною картою. До 1988 року ви могли отримати 16-бітну 2D-карту VGA Wonder від ATI (компанія, яку врешті придбала компанія AMD). До 1996 року ви можете придбати 3D-графічний прискорювач у 3dfx Interactive, щоб змогли запускати шутер від першої особи Quake на повній швидкості. Також у 1996 році Nvidia почала намагатися конкурувати на ринку 3Dприскорювачів зі слабкими продуктами, але навчилася, і в 1999 році представила успішну GeForce 256, першу графічну карту, яку назвали GPU. Тоді основною причиною наявності графічного процесора була гра. Лише пізніше люди почали використовувати графічні процесори для математики, науки та техніки.

У 2003 році група дослідників під керівництвом Яна Бака представила Brook – першу широко прийняту модель програмування, яка розширила C за допомогою паралельних даних конструкцій. Пізніше Бак приєднався до Nvidia і очолив запуск CUDA в 2006 році – першого комерційного рішення для обчислень загального призначення на графічних процесорах.

Конкурент CUDA OpenCL був запущений Apple та Khronos Group в 2009 році, намагаючись забезпечити стандарт для різнорідних обчислень, який не обмежувався процесорами Intel / AMD з графічними процесорами Nvidia. Хоча OpenCL звучить привабливо через свою загальність, він не настільки успішно працював з CUDA на графічних процесорах Nvidia, і багато фрейм ворки глибокого навчання або не підтримують його, або підтримують лише як додаткову думку після виходу їх підтримки CUDA. Протягом багатьох років CUDA вдосконалював і розширював сферу застосування, більш-менш поступово завдяки вдосконаленим графічним процесорам Nvidia. Починаючи з версії 9.2 CUDA, використовуючи кілька серверних графічних процесорів P100, можна досягти в 50 разів поліпшення продуктивності порівняно з центральними процесорами[12].

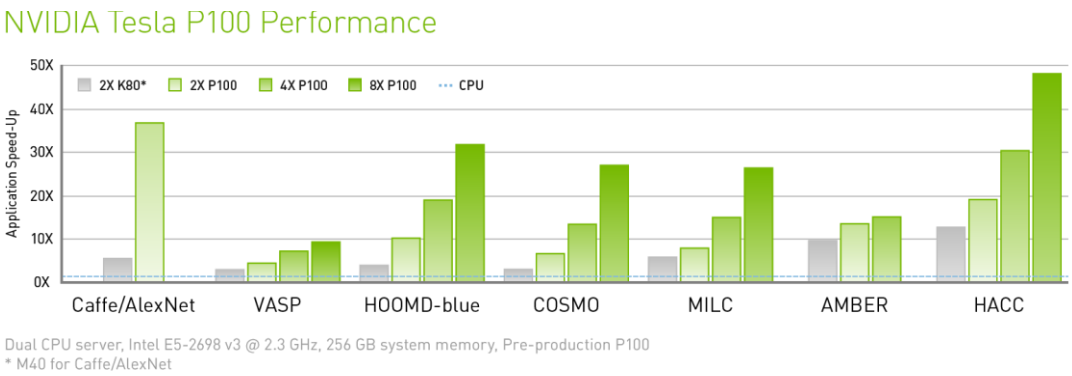


Рис.3.1 – Графік продуктивності процесора NVIDIA Tesla P100

Одно потокове збільшення продуктивності процесорів з часом, яке, за законом Мура, передбачалося подвоювати кожні 18 місяців, сповільнилося до 10 відсотків на рік (детально вказано на рисунку 3.2), оскільки виробники чіпів стикалися з фізичними обмеженнями, включаючи обмеження розміру роздільної здатності маски чипа та виходу чіпів під час виробничого процесу і обмеження тепла на тактових частотах під час роботи.

Графічні процесори CUDA та Nvidia були застосовані у багатьох областях, які потребують високих обчислювальних характеристик з плаваючою точкою, як це зображено на малюнку вище. Більш повний перелік включає:

— обчислювальні фінанси;

— моделювання клімату, погоди та океану;

— наука про дані та аналітика;

— поглиблене навчання та машинне навчання;

— оборона та розвідка;

— виробництво / AEC;

— медіа та розваги.

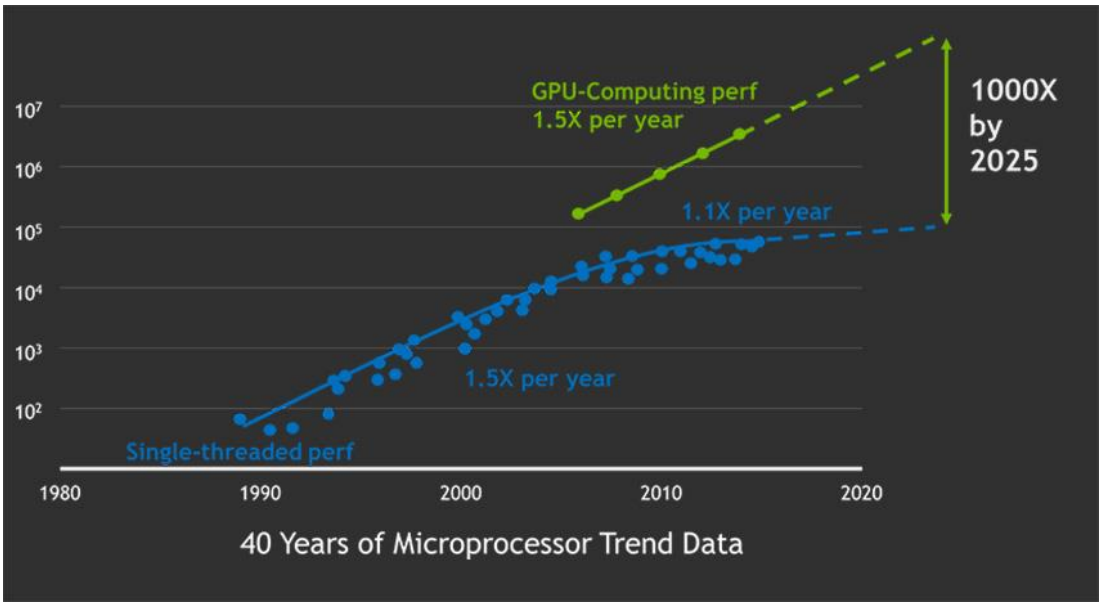


Рис.3.2 – Дані мікропроцесорів за 40 років

Глибоке навчання має велику потребу в швидкості обчислень. Наприклад, для підготовки моделей для Google Translate у 2016 році команди Google Brain та Google Translate провели сотні тижневих запусків TensorFlow з використанням графічних процесорів; вони придбали у Nvidia 2000 серверних графічних процесорів. Без графічних процесорів ці навчальні прогони зайняли б місяці, а не тиждень, щоб сходитися. Для розгортання цих моделей перекладу TensorFlow Google використовував новий спеціальний процесорний чіп, TPU. На додаток до TensorFlow, багато інших фреймворків DL покладаються на CUDA для підтримки графічного процесора, зокрема

Caffe2, CNTK, Databricks, H2O.ai, Keras, MXNet, PyTorch, Theano та Torch. У більшості випадків вони використовують бібліотеку cuDNN для обчислень глибокої нейронної мережі. Ця бібліотека настільки важлива для навчання фреймворків глибокого навчання, що всі фреймворки, що використовують задану версію cuDNN, мають по суті однакові показники продуктивності для еквівалентних випадків використання. Коли CUDA та cuDNN покращуються від версії до версії, усі основи глибокого навчання, які оновлюються до нової версії, бачать приріст продуктивності. Де продуктивність, як правило, відрізняється від фрейм ворку до того, наскільки якісно вони масштабуються до декількох графічних процесорів та декількох вузлів. Набір інструментів CUDA включає бібліотеки, засоби налагодження та оптимізації, компілятор, документацію та бібліотеку середовища виконання для розгортання ваших програм. Він має компоненти, які підтримують глибоке навчання, лінійну алгебру, обробку сигналів та паралельні алгоритми. Загалом, бібліотеки CUDA підтримують усі сімейства графічних процесорів Nvidia, але найкраще працюють на останньому поколінні, наприклад V100, який може бути в три рази швидшим за P100 для навчальних робочих навантажень (показано на рисунку 3.3). Використання однієї або декількох бібліотек – це найпростіший спосіб скористатися графічними процесорами, якщо алгоритми, які вам потрібні, були реалізовані у відповідній бібліотеці.

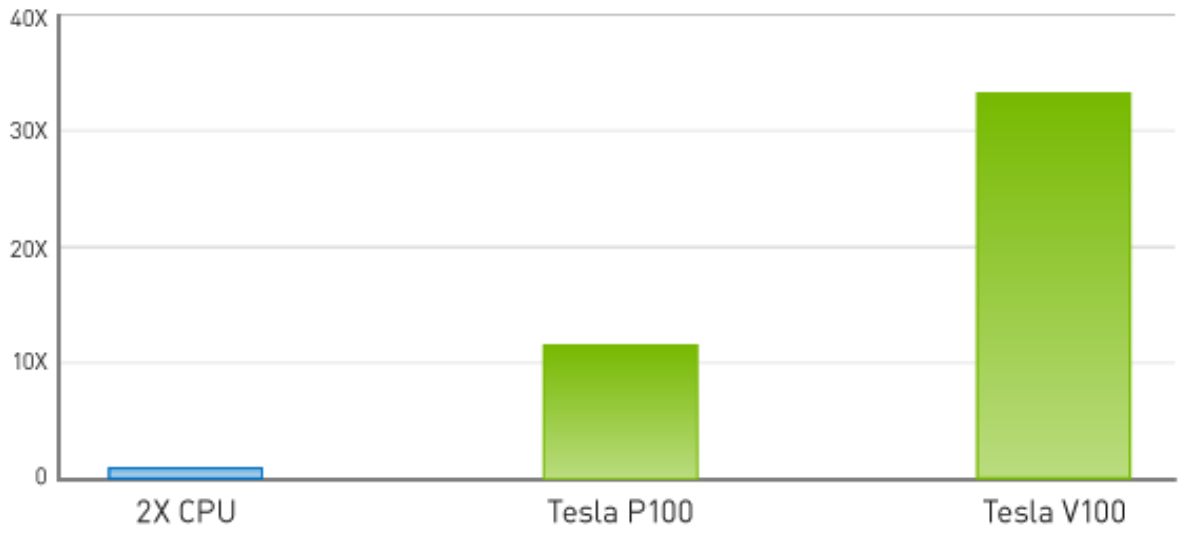


Рис.3.3 – Глибоке навчання нейронної мережі за декілька годин

У сфері глибокого навчання є три основні бібліотеки, прискорені GPU: cuDNN, яку вже згадували раніше як компонент GPU для більшості платформ глибокого навчання з відкритим кодом; TensorRT, який є високоефективним оптимізатором виводу глибокого навчання та середовищем виконання; та DeepStream, бібліотека відеовисновків. TensorRT допомагає оптимізувати нейромережеві моделі, відкалібрувати для меншої точності з високою точністю та розгорнути навчені моделі в хмарах, центрах обробки даних, вбудованих системах або платформах автомобільних продуктів.

Усі три бібліотеки для паралельних алгоритмів мають різне призначення. NCCL призначена для масштабування програм на декількох графічних процесорах і вузлах; nvGRAPH – для аналізу паралельних графіків; і Thrust – це бібліотека шаблонів C для CUDA на основі стандартної бібліотеки шаблонів C . Thrust забезпечує багату колекцію даних паралельних примітивів, таких як сканування, сортування та зменшення.

У деяких випадках замість еквівалентних функцій центрального процесора можна використовувати функцію CUDA, що випадає. Наприклад, підпрограми множення матриць GEMM від BLAS можна замінити версіями графічного процесора, просто зробивши посилання на бібліотеку NVBLAS.

3.2 Програмування з використанням CUDA технологій

При використанні традиційних API програміст не зважаючи на складність алгоритму завжди повинен ковфігурувати всі частини графічного конвеєру. Цей факт суттєво ускладнює використання GPU для вирішення завдань загального призначення, тому що навіть просте додавання двох матриць вимагає виконання ряду команд з підготовки. У результаті на кілька рядків шейдерної програми припадають сотні рядків додаткового коду.

Модель програмування, що використовується в CUDA відрізняється від традиційних API тим, що повністю приховує графічний конвеєр від програміста, й чим дозволяє йому писати програми у більш звичайному вигляді на розширеній варіації мови C.

Крім того, CUDA надає програмісту зручну модель. роботи із пам'яттю. Більше немає необхідності зберігати дані у 128-бітних текстурах, оскільки CUDA дозволяє читати дані безпосередньо з пам'яті відеокарти.

До складу NVIDIACUDA входять два API: високого рівня (CUDA Runtime API) та низького (CUDA Driver API) що продемонсторвано на рисунку 3.4. При необхідності задіяти низькорівневі функції графічного процесора програміст завжди може відмовитися від Runtime API на користь Driver API.

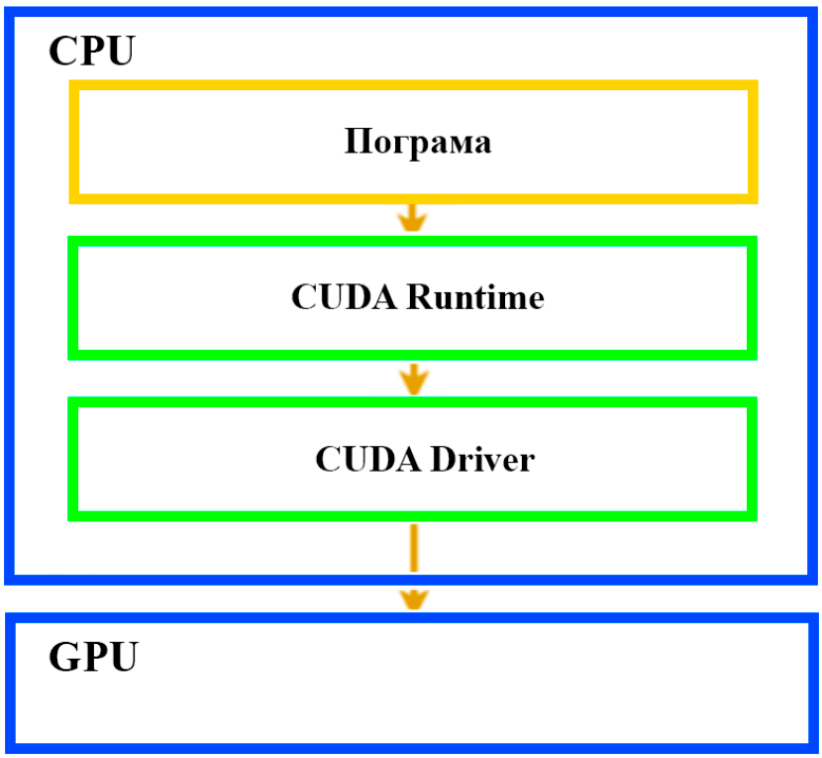


Рисунок 3.4 — Схема роботи CUDA API

Першим кроком при перенесенні існуючого алгоритму на CUDA неодмінно є його аналіз, мета якого полягає в пошуку місць», що потребують розпаралелювання. Як правило, в алгоритмі для CPU такі ділянки містяться в циклах або рекурсіях.

Повне перенесення алгоритму на GPU не є можливим, так як графічний процесор не має доступу ні до пам'яті комп'ютера, ні до пристроїв вводу/виводу. При виконанні програми CPU, як і раніше, відповідає за підготовку та постобробку даних, сама ж трудомістка робота лягає на GPU. Набір інструкцій на графічному процесорі називається ядром (kernel)[13].

Саме за формування та компіляцію ядер відповідає CPU. В свою чергу відеочіп просто приймає вже скомпільоване ядро і створює копії для кожного елемента даних. Головним плюсом розробки програмного забезпечення з використанням технології CUDA, являється те що кожне із сформованих ядер виконує поставлену задачу у власному потоці.

Потоки в GPU можуть виконуватися лише групами по 32 екземпляри. Але максимальним числом допустимим для використання в одній задачі є число рівне кількості CUDA-ядер відеокарти. Тому кожен такт апаратне забезпечення вибирає, яка із груп потоків буде виконана. Але якби в CPU подібне перемикання зайняло сотні тактів, то GPU робить це майже миттєво.

На відміну від шейдерів, де всі дані представлені у вигляді чотирьох компонентних векторів, дані в ядрі є скалярними. Таке уявлення є більш природнім більшості неграфічних завдань.

Кожне ядер виконується над сіткою блоків. У кожений момент час на GPU може виконуватися лише одна сітка. Подібне угруповання дозволяє досягти максимальної масштабованості. Якщо GPU недостатньо ресурсів для запуску всіх блоків, вони будуть виконуватися послідовно, один за одним[13]. Це дозволяє розробнику не замислюватися про потужність пристрою, на якому буде запущено програму. У кожному додатку, написаному на NVIDIACUDA незалежно від його призначення, можна виділити ряд загальних кроків:

— підготовка пам'яті;

— конфігурація сітки та блоків;

— запуск ядра;

— отримання результатів та звільнення пам'яті.

3.3 Огляд компілятора ILGPU для використання CUDA в коді C#

На даний момент CUDA підтримую розробку програмного забезпечення не на багатьох мова, тому для розробки програмного забезпечення мовою програмування

C# нам потрібно скористатись сторонніми завнішніми бібліотеками для роботи з CUDA-ядрами графічного процесора. Однією із таких бібліотек являється ILGPU.

ILGPU— це компілятор типу JIT(just-in-time) який призначений для використання в високопродуктивних програм з використанням обчислювальних потужностей GPU, написаних мовами на основі .Net. ILGPU створений на мові C# без будь-яких залежностей від інших бібліотек, що дозволяє розробляти програми з використанням GPU. Основною ідеєю компілятора є поєднання зручностей C++ AMP з високою продуктивністю CUDA. Функції в області дії над ядрами не є анотованими (наприклад, функції C# за замовчуванням) і ї дозволяється працювати із різними типами даних, принцип компіляції коду наведено на рисунку 3.5.

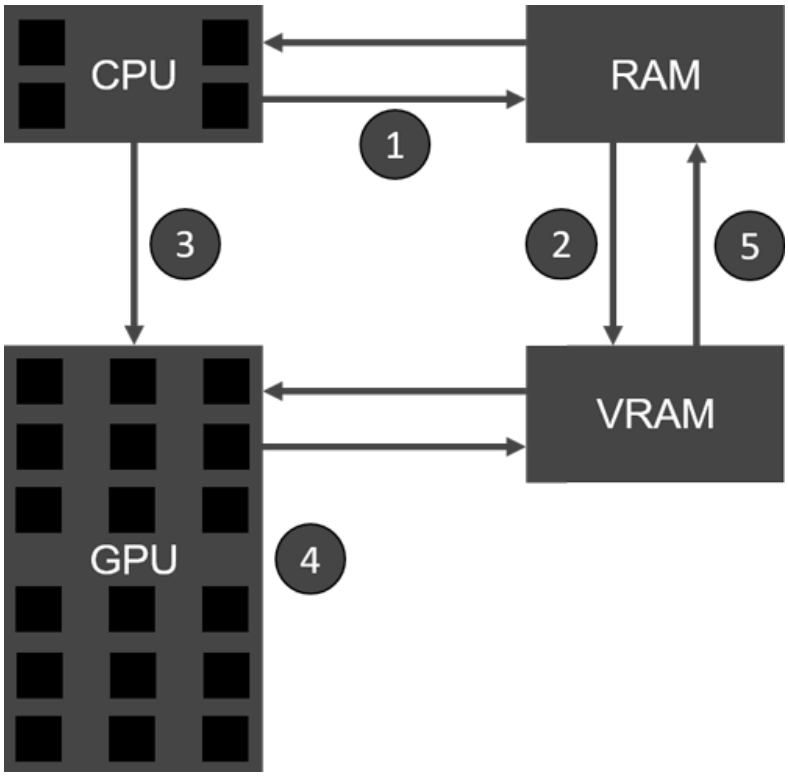


Рис.3.5 – Процес компіляції коду за допомогою ILGPU

Даний компілятор дозволяє виконання коду на графічному процесорі за допомогою NVIDIA Nsight Visual Studio Edition. Також реалізує розширенні функції C#, включаючи віртуальні функції та загальні засоби паралельного багатопоточного обчислення[15].

3.4 Огляд програмного середовища та мови програмування C#.

Для розробки програмного забезпечення для системи обробки низькочастотних сигналів було обрано програмне середовище Visual Studio 2022 яке максимізує зручність та продуктивність розробки за допомогою зручного та інтуїтивного інтерфейсу. Підтримує розробку програмного забезпечення для персональних комп'ютерів та портативних пристроїв, та мобільних пристроїв. Visual Studio дозволяє розробляти програмне забезпечення на одній із найпопулярніших мов програмування, а саме C#.

C # - об'єктно-орієнтована мова програмування. Розроблено в 1998-2001 роках групою інженерів компанії Microsoft під керівництвом Андерса Хейлсберг і Скотта Вільтаумота [10] як мову розробки додатків для платформи Microsoft .NET Framework. Згодом був стандартизований як ECMA-334 і ISO / IEC 23270.

C # відноситься до сім'ї мов з C-подібним синтаксисом, з них його синтаксис найбільш близький до C ++ і Java. Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів (в тому числі операторів явного і неявного приведення типу), делегати, атрибути, події, змінні, властивості, узагальнені типи і методи, ітератори, анонімні функції з підтримкою замикань, LINQ, виключення, коментарі в форматі XML[16].

У мові прийнята загальна система роботи з типами, починаючи від примітивів і закінчуючи складними, в тому числі, призначеними для користувача наборами. Застосовується єдиний набір операцій для обробки і зберігання значень типізації. Також можна використовувати посилальні типи користувача, що дозволить динамічно виділити пам'ять під об'єкт або зберігати спрощену структуру в мережі.

Мова програмування забороняє звернення до змінних, що не були ініційовані, що виключає можливість виконання безконтрольного приведення типів або виходу за межі певного масиву даних.

Перейнявши багато від своїх попередників - мов C ++, Delphi, Модула, Smalltalk і, особливо, Java - C #, спираючись на практику їх використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем,

наприклад, C # на відміну від C ++ не підтримує множинне успадкування класів (між тим допускається множинна реалізація інтерфейсів).

Основною його особлививістю є його швиткодія, та простота написання коду, але й основними недостатком, як було вказано раніше водсутність прямої підтримки технологією CUDA.

Після завантаження Visual Studio та його налаштування, відкриється вікно із можливістю створити новий проект або ж відкрити вже готовий. Після вибору «Create new project», відкриється вікно налаштувань проекту, зображене на рис.3.6.

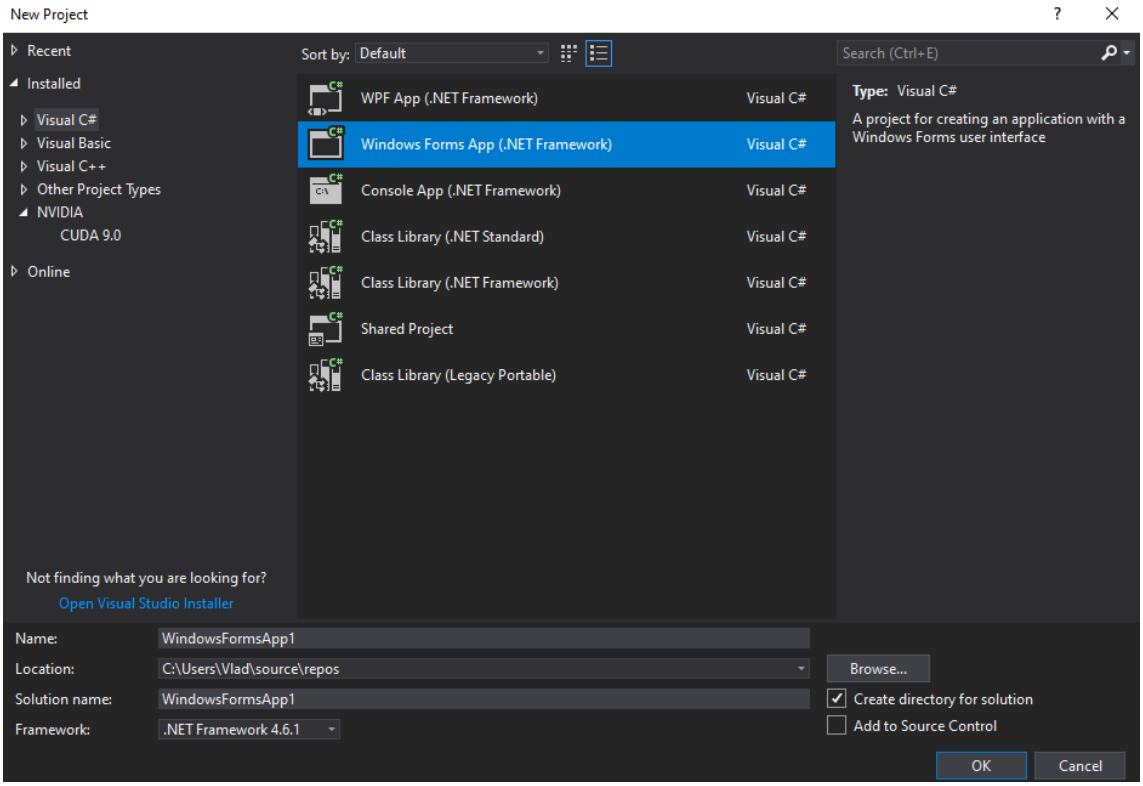


Рисунок 3.6 — Створення нового проекту

Нам буде доступний вибір між мовами програмування, типом проекту тощо. Але так як розробляється конмольне програмне забезпечення на мові C#, потрібно вибрати «Console App (.Net Framework)». Після вибору та натиснення клавіші «Ok», відкривається середовище розробки із базовим програмним кодом, приклад зображено на рисунку 3.7.

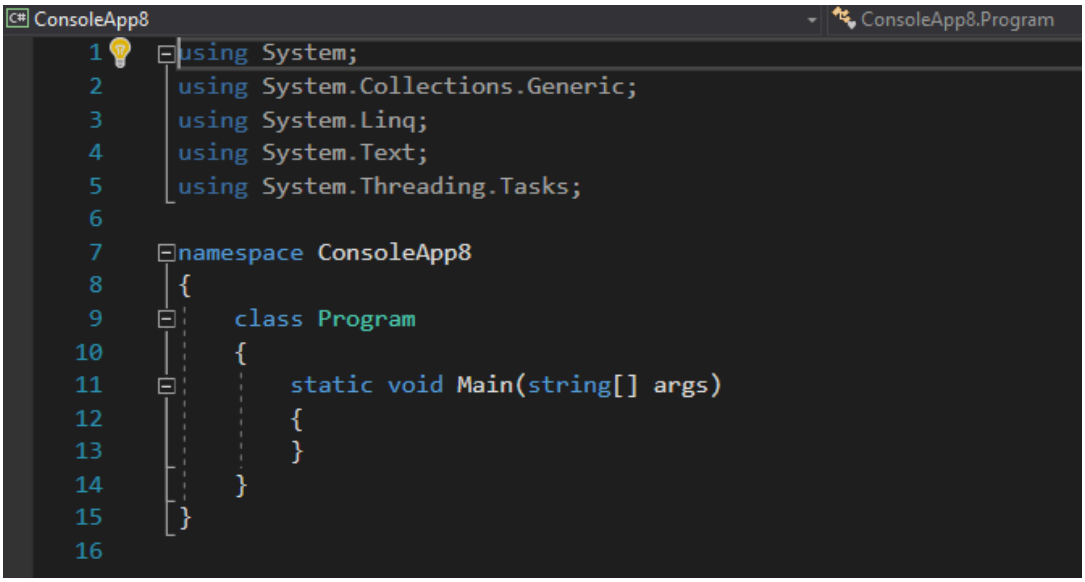


Рисунок 3.7 — Приклад створення нового проекту

3.5 Розробка програмного забезпечення для обчислення низькочастотних сигналів з використанням технології CUDA.

Під час розробки програмних засобів, розробка розпочинається із побудови блок-схеми алгоритму роботи програми. Блок-схеми - це не що інше, як графічне представлення даних або алгоритм для кращого розуміння коду візуально. Він відображає покрокові рішення проблеми, алгоритму чи процесу. Це наочний спосіб представлення кроків, які віддають перевагу більшість програмістів для правильної побудови алгоритму програми, таким чином він сприяє усуненню проблем в алгоритмі. Блок-схема зображує блоки, які вказують на послідовний перебіг процесу. Оскільки блок-схема є наочним зображенням процесу або алгоритму, його легко інтерпретувати та розуміти. Блок-схема розроблюваного програмного забезпечення зображена на рисунку 3.8.

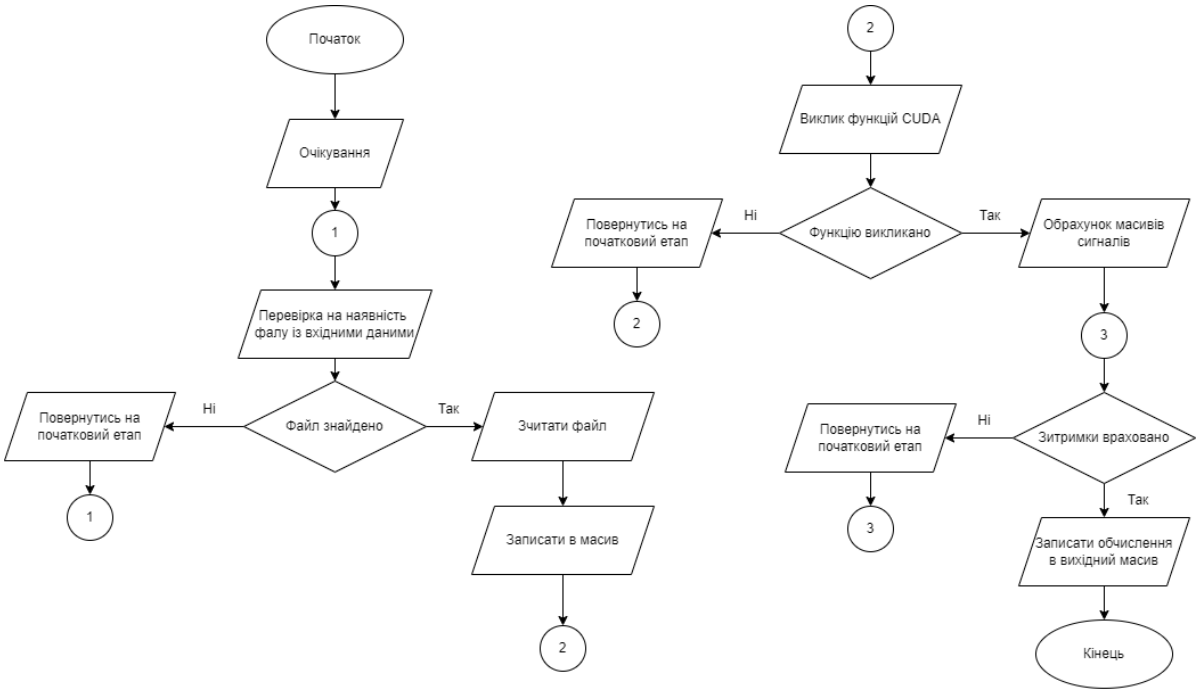


Рисунок 3.8 – Блок-схема роботи алгоритму

При поставленій задачі можна приступати до безпосередньої розробки програми. Основною проблемою при розробки програмного забезпечення в якому використовуються технології CUDA, являє собою відсутність потрібного мінімального GPU. Для цього першим кроком при написанні програми реалізовано перевірку наявних графічних прискорювачів продемонстровано на рисунку 3.9

```
Ссылок: 1
public static void CudaRun()
{
    using Context context = Context.Create(builder => builder.Cuda());
    foreach (Device d in context.GetCudaDevices())
    {
        using Accelerator accelerator = d.CreateAccelerator(context);
        Console.WriteLine(accelerator);
        Console.WriteLine(GetInfoString(accelerator));
    }

}
Ссылок: 1
private static string GetInfoString(Accelerator a)
{
    StringWriter infoString = new StringWriter();
    a.PrintInformation(infoString);
    return infoString.ToString();
}
```

Рисунок 3.9 — Виведеня інформації про GPU із підтримкою CUDA-ядер

Як результат роботи ми отримаємо виведення інформації в консоль про всі наявні прискорювачі, також кількістю доступних CUDA-ядер, розмір пам’яті графічного процесора, та версію драйвера, демонстрація роботи наведене на рисунку 3.10.

```
NVIDIA GeForce MX150 [Type: Cuda, WarpSize: 32, MaxNumThreadsPerGroup: 1024, MemorySize: 2147352576]
Device: NVIDIA GeForce MX150
  Accelerator Type:                         Cuda
  Cuda device id:                           0
  Cuda driver version:                      11.7
  Cuda architecture:                        SM_61
  Instruction set:                          7.6
  Clock rate:                               1531 MHz
  Memory clock rate:                        3004 MHz
  Memory bus width:                         64-bit
  Warp size:                                32
  Number of multiprocessors:                3
  Max number of threads/multiprocessor:     2048
  Max number of threads/group:              1024
  Max number of total threads:              6144
  Max dimension of a group size:            (1024, 1024, 64)
  Max dimension of a grid size:             (2147483647, 65535, 65535)
  Total amount of global memory:            2147352576 bytes, 2047 MB
  Total amount of constant memory:          65536 bytes, 64 KB
  Total amount of shared memory per group: 49152 bytes, 48 KB
  Total amount of shared memory per mp:     98304 bytes, 96 KB
  L2 cache size:                            524288 bytes, 512 KB
  Max memory pitch:                         2147483647 bytes
  Total number of registers per mp:         65536
  Total number of registers per group:      65536
  Concurrent copy and kernel execution:     True, with 5 copy engines
  Driver mode:                              WDDM
  Has ECC support:                          False
  Supports managed memory:                  True
  Supports compute preemption:              True
  PCI domain id / bus id / device id:       0 / 1 / 0
  NVML PCI bus id:                          0000:01:00.0
```

Рисунок 3.10 — Виведення інформації про наявний графічний процесор з підтримкою CUDA-ядер

Далі потрібно конвертувати звуковий файл в масив даних, цього було реалізовано метод ReadToByte. Даний метод відкриває звукувий файл типу .wav, після чого зчитує його значення в масив байтів. Масиви зберігаються до окремих текстових вайлів для подольшого їх опрацювання, код продемонстовено в додатку Г .

Далі написання методу Reader, у якому реалізовано зчитування перетвореного звукового файлу, з подальшим записом кожного каналу у одновимірний масив даних. На рисунку 3.11 зображено частина його вмісту, повний лістинг буде приставлено в додатку Ґ

Для зчитування та запису даних із файлу в масив потрібно скористатись простором імен System.IO. Простір імен System.IO складається з класів, структур, делегатів і перерахувань, пов'язаних з IO. Ці класи можна використовувати для читання та запису даних у файли або потоки даних. Він також містить класи для підтримки файлів і каталогів.

```
public static void Reader()
{
    reader1 = new StreamReader("Kanal1.txt");
    reader2 = new StreamReader("Kanal2.txt");
    reader3 = new StreamReader("Kanal3.txt");
    reader4 = new StreamReader("Kanal4.txt");
    reader5 = new StreamReader("zatrumku.txt");

    file = reader1.ReadToEnd();
    WorkWithFile(file, list, out kanal1);

    file = reader2.ReadToEnd();
    WorkWithFile(file, list, out kanal2);

    file = reader3.ReadToEnd();
    WorkWithFile(file, list, out kanal3);

    file = reader4.ReadToEnd();
    WorkWithFile(file, list, out kanal4);

    file = reader5.ReadToEnd();
    WorkWithFile(file, list, out zatrumku);
```

Рисунок 3.11 – Метод Reader

Для зчитування та запису даних із файлу в масив потрібно скористатись простором імен System.IO. Простір імен System.IO складається з класів, структур, делегатів і перерахувань, пов'язаних з IO. Ці класи можна використовувати для читання та запису даних у файли або потоки даних. Він також містить класи для підтримки файлів і каталогів.

Основним класом який використовується для зчитування даних із файлу являється StreamReader. StreamReader використовується для читання символів у потік у вказаному його кодуванні. Метод Read зчитує наступний символ або наступний набір символів з вхідного потоку. StreamReader успадкована від TextReader, який надає методи для читання символу, блоку, рядка або всього вмісту. Також було оголошено змінні для запису в них одновимірних масивів даних які було зчитано із файлів.

Далі було реалізовано було реалізовано метод WorkWithFile. В даному методі нам потрібно пройтись по всім символам файлу. Даний метод перевіряє всі символи які було зчитано та зрівнює їх із ASCII кодом символів. Числа формуються в список типу String, який по закінченню роботи методу конвертується в масив типу int, приклад коду даного методу наведено в рис. 3.12.

```
public static void WorkWithFile(string name, List<string> list, out int[] kanal)
{
    tsufra = false;
    count = 0;
    list.Clear();

    for (int i = 0; i < name.Length; i++)
    {
        if (Convert.ToInt32(name[i]) >= 48 && Convert.ToInt32(name[i]) <= 57)
        {
            if (tsufra == false)
            {
                list.Add(name[i].ToString());
            }
            else
            {
                list[count] += name[i];
            }
            tsufra = true;
        }
        else
        {
            if (tsufra == true)
            {
                count++;
            }
            tsufra = false;
        }
    }
```

Рисунок 3.12 — Реалізація методу WorkWithFile

Далі нам потрібно завантажити створені масиви до пам'яті графічного процесора, для цього потрібно скористатись об’єктом класу ArrayView1D. Він приставляє собою виділення області пам’яті заданого типу значень на певних прискорювачах. Дані можна копіювати в будь який прискорювач, та з нього за допомогою операції

синхронізації або асинхронізації копіювання з використанням методу Streams. ArrayView напряму взаємодіє із ядром графічного процесора. Компілятор ILGPU підтримую лінійні типу 1D, та 2D, 3D буферів. Але для даної задачі було використано буфер типу 1D, так як потрібно працювати із одновимірними масивами рис. 3.13.

```
using Context context = Context.Create(builder => builder.Cuda());
Accelerator accelerator = context.CreateCudaAccelerator(0);

ArrayView1D<int, Stride1D.Dense> chanel1 = accelerator.Allocate1D(kanal1);
ArrayView1D<int, Stride1D.Dense> chanel2 = accelerator.Allocate1D(kanal2);
ArrayView1D<int, Stride1D.Dense> chanel3 = accelerator.Allocate1D(kanal3);
ArrayView1D<int, Stride1D.Dense> chanel4 = accelerator.Allocate1D(kanal4);
ArrayView1D<int, Stride1D.Dense> zatrumka = accelerator.Allocate1D(zatrumku);
```

Рисунок 3.13 — Завантаження масивів даних до пам’яті графічного процесора

В даній частині спочатку було викликано об’єкт Context. Всі класи та функції компілятора ILGPU покладаються на екземпляр ILCPU.Context. Даний об’єкт слугує інтерфейсом для компілятора. Об’єкт Context, а також більшість екземплярів класів, потребують виклик об’єкту dispose що запобігає втраті пам’яті. Accelerator являє собою апаратній або програмний графічний процесор. В даному коді представлено Accelerator типу CUDA, в свою чергу він вказує що системі потрібно використовувати графічний процесор. Але головним його не достатком слугує те що для використання даного об’єкту потрібний графічний процесор серії Nvidia GTX 980 або вище, та встановлене програмне забезпечення CUDA не нижче чим 10 версії.

Наступним кроком в розробці програмного забезпечення, слугує визначення функції ядра. Вони мають деякі обмеження, хоча під час проведення ними математичних операції все працює без проблем, так як об’єкт ArrayViews приймає в себе значення переданого масиву, і подальші операції виконуються за допомогою них рис 3.14.

```
static void Kernel(Index1D i, ArrayView<int> kanal1, ArrayView<int> output)
{
    output[i] = kanal1[i % kanal1.Length];
}
```

Рисунок 3.14 — Приклад реалізації ядра графічного процесора

Під час реалізації ядра першим параметром передається індекс. Ядро завжди виконує ітерацію по певному екстенту. В більшості довжина вихідного MemoryBuffer. Іншими його параметрами можуть бути структури або ArrayView. Максимум при формуванні ядра дозволяється задавати 19 параметрів.

Останнім кроком є ралізація методу MathOperation. Даний метод відповідає за проведення математичних операцій, частина коду наного медоту представлена на рисунку 3.15

```
Ссылок: 1
public static void MathOperation(out int[,] results)
{
    int lenghtKanal = chane1.Length;
    int lenghtZatrumku = zatrum.Length;

    results = new int[lenghtKanal, lenghtZatrumku];


    for (int i = 0; i < lenghtKanal;i++)
    {
        for (int j = 0; j < lenghtZatrumku; j++)
        {
            int koeficientA = i - (j * 2);
            int koeficientB = i - j;
            int koeficientC = i + j;
            int koeficientD = i + (j * 2);

            int A = koeficientA >= 0 && koeficientA < lenghtKanal ? kanal1[koeficientA] : 0;
            int B = koeficientB >= 0 && koeficientB < lenghtKanal ? kanal1[koeficientB] : 0;
            int C = koeficientC >= 0 && koeficientC < lenghtKanal ? kanal1[koeficientC] : 0;
            int D = koeficientD >= 0 && koeficientD < lenghtKanal ? kanal1[koeficientD] : 0;

            results[i, j] = (A + B + C + D) / 4;
        }
    }
```

Рисунок 3.15 — Реалізація методу MathOperation

Оснособновний алгоритм обчислення поставленої зададачі задається за формолою (3.1)

$$result = (A[i - j*2] + B[i - j] + A[i + j] + B[i + j*2]) / 4 \qquad (3.1)$$

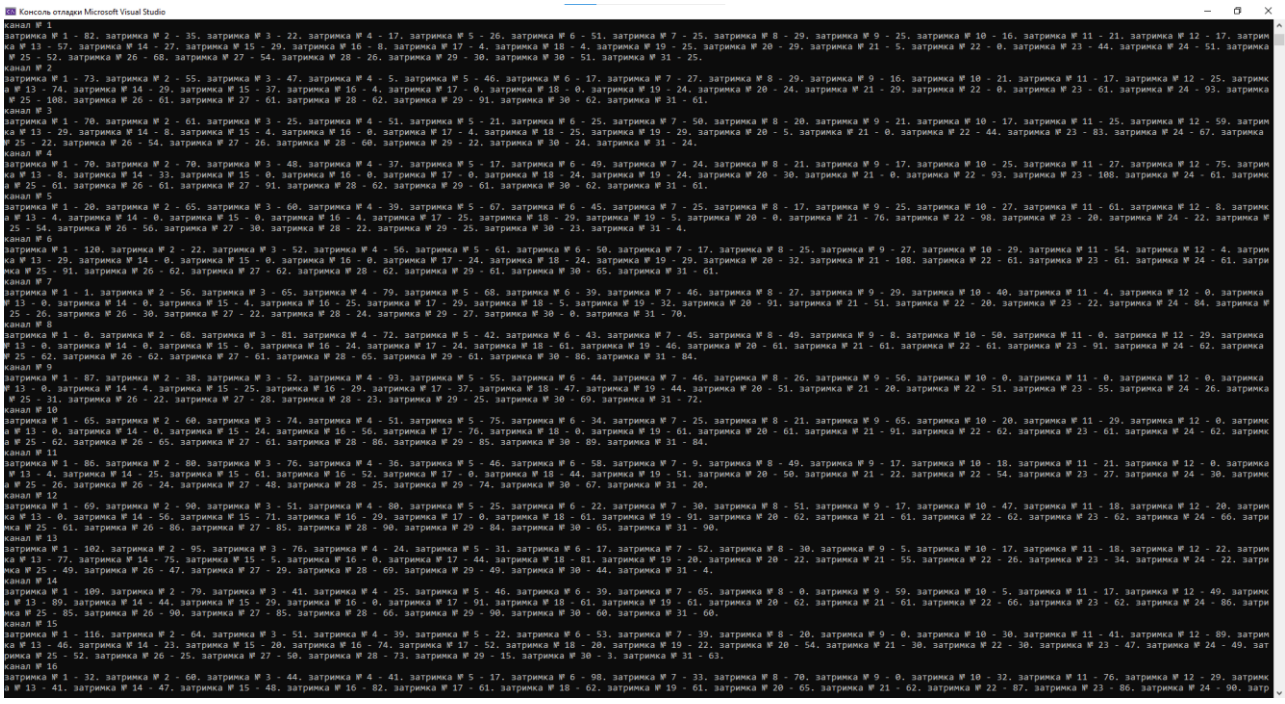
де A, B, C, D — масиви вхідних даних;

i — індекс елементу масиву;

j — значення заданої затримки;

Але при реалізації заданого алгоритму виникла проблема, що положення індесу виходить за рамки заданих масивів. В цьому випадку було добавлену тернарну операцію для перевірки значення. В випадку виходу за рамку масиву, заміняємо значення нулем.

В результаті роботи даного методу ми отримуємо кількість масивів рівну кількості затримок, а кількість елементів кожного масиву рівна кількості елементів масивів вхідних даних.



Рисунко 3.16— Результат роботи програми

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено комп’ютерну аналого-цифрову систему опрацювання низькочастотних сигналів з використанням CUDA-технологій.

Розглянуто існуючі методи і засоби акустичної локації та ідентифікації об’єктів на місцевості. Оглянуто структурні та обчислювальні методи апаратних та програмних засобів акустичної локації та ідентифікації. Досліджена структурна схема, склад та принцип роботи аналого-цифрових систем. Проаналізовано методи побудови і принципи функціонування мікрофонних решіток.

Було проаналізовано технології обробки низькочастотних звукових сигналів при багатоканальному скануванні в системі мікрофонних решіток. Досліджено конструктивно-схемотехнічні рішення мікрофонних решіток. Також розраховані співвідношення для сканування й обробки сигналів.

Розроблене програмне забезпечення для обчислення низькочастотних сигналів за допомогою технологій CUDA-ядер. Використано бібліотеки для роботи із ядрами графічного процесора. Оглянуто компілятор ILGPU для використання CUDA-ядер в коді C#.

На остаток протестоване розроблене програмне забезпечення на наявність неполадок та помилок в обрахунках.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Комп'ютерна аналого-цифрова система опрацювання низькочастотних сигналів з використанням CUDA-технологій [Текст] / Д. А. Савчук // LI Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2022): Тез. доп. — Вінниця, 2022. — 1. Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/14885 (дата звернення 09.06.22).

2. Арнаутов О.А., Момот Р.В., Худов Г.В. Методи визначення координат об’єктів в системах пасивної локації // Системи озброєння і військова техніка. – 2012. – № 3 (31). – С. 111–113.

3. Азаров О.Д. Аналого-цифрове порозрядне перетворення на основі надлишкових позиційних систем числення з ваговою надлишковістю: Монографія / Азаров О.Д. – Вінниця.– ВНТУ, 2010. – 232 с.

4. Крупельницький Л.В., Азаров О.Д. Аналого-цифрові пристрої систем, що самокоригуються, для вимірювань і обробляння низькочастотних сигналів: Монографія / Під ред. О.Д. Азарова.– УНІВЕРСУМ-Вінниця, 2005.– 167 с.

5. Ткаченко О.М. Метод швидкого пошуку найближчого сусіда з обчисленням відстані за зваженою евклідовою метрикою / О.М. Ткаченко, О.Ф. Грійо Тукало // Вісник Вінницького політехнічного інституту. – Вінниця: ВНТУ.– 2013.– №1.– С.116–122.

6. Методи та засоби для визначення напрямку та для ідентифікації джерел звуків на місцевості / Ткаченко О. М.; Крупельницький Л. В.; Дерев'яга Б.С.; Зінчук Р. С. // Матеріали XLV Науково-технічної конференції ВНТУ, Вінниця, 23-24 березня 2016 р.

7. Л.В. Крупельницький Характеристики і структури багатоканальних АЦ-систем, що самокоригуються, для аналізу аудіо сигналів // Тези доповідей П’ятої Міжнародної науково-практичної конференції "Методи та засоби кодування, захисту й ущільнення інформації". –Україна.– Вінниця, 19-21 квітня 2016 р. – Вінниця: ВНТУ, 2016. – С. 129–133.

8. Ткаченко О.М. Пошук найближчого вектора у кодових книгах на основі бінарного дерева / О.М. Ткаченко, О.Ф. Грійо Тукало // Інформаційні технології та комп'ютерна інженерія: Міжнародний науково-технічний журнал.– Вінниця, 2014.– №3(31).– С. 67–55.

9. Степанова Т.М. Етапи розвитку мікрофонних решіток для комп'ютерних систем акустичної локації // Матеріали XLVII Науково-технічної конференція ВНТУ , Вінниця, 21-23 березня 2018 р.

10. Гарнага В.А. Параметри мікрофонних акустичних решіток і попередніх підсилювачів аудіосигналів // Матеріали XLVII Науково-технічної конференція ВНТУ , Вінниця, 21-23 березня 2018 р

11. Крупельницький Л.В., Азаров О.Д. Аналого-цифрові пристрої систем, що самокоригуються, для вимірювань і обробляння низькочастотних сигналів: Монографія / Під ред. О.Д. Азарова.– УНІВЕРСУМ-Вінниця, 2005.– 167 с.

12. NVIDIA DOCUMENTANION [Електронний ресурс] — Режим доступу до ресурсу: https://docs.nvidia.com/cuda/

13. Мова програмування C# і платформа .NET [Електронний ресурс] — Режим доступу до ресурсу: https://metanit.com/sharp/tutorial/1.1.php

14. CUDA Programming Guide [Електронний ресурс] — Режим доступу до ресурсу: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

15. ILGPU Documentation [Електронний ресурс] — Режим доступу до ресурсу: https://www.ilgpu.net/docs/ReadMe

ДОДАТОК А
Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп’ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

_____ проф., д.т.н. О.Д. Азаров

«___»_____2021 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання дипломної роботи

«Комп’ютерна аналого-цифрова система опрацювання низькочастотних сигналів з використанням CUDA-технологій»

08-23.БДР.012.00.000 ПЗ

Науковий керівник к.т.н., доц. каф. ОТ

_____ Крупельницький Л.В.

Студента групи 1КІ-18б

_____ Савчук Д.А.

Вінниця 2022

1 Підстава для виконання бакалаврської дипломної роботи (БДР)

Підставою для розробки даної бакалаврської дипломної роботи є наказ ВНТУ № від «__» ___ 2022 року та рішення засідання кафедри обчислювальної техніки (протокол №_1_ від «_____»__ 2022 року).

2 Вихідні дані для виконання БДР

Тридцять два масива даних опрацьованого звукового сигналу із урахуванням затримок з частотою дискритизації 48 кГц. розрядністю 16-24 двійкових розряди.

3 Перелік задач

Перелік задач, що повинні бути виконані:

—огляд інуючих методів і засобів акустичної локації та індетифікації об’єктів на місцевості;

—аналіз технології обробки низькочастотних звукових сигналів при багатоканальному скануванні в системі мікрофонної решітки;

—використання CUDA-ядер або бібліотек для роботи із ними в мові програмування C#.

4 Перелік матеріалів, що подаються до захисту БДР

До захисту подається: пояснювальна записка БДР, графічні і ілюстративні матеріали, протокол попереднього захисту БДР на кафедрі, відгук наукового керівника, відгук рецензента, анотації до БДР українською та іноземною мовами, довідка про відповідність оформлення БДР діючим вимогам.

5 Порядок контролю виконня та захисту БДР

5.1 Робота виконується в три етапи, таблиця А.1

Таблиця А.1 — Етапи виконання роботи

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Інформаційний пошук та огляд інформаційних джерел	12.09.2021	25.11.2021	Розділи 1 та 2
2	Дослідження підходів та розробка схемної реалізації	26.11.2021	04.02.2022	Чернетки матеріалів
3	Підготока матеріалів пояснювальної записки	05.02.2022	28.03.2022	Пояснювальна записка

6 Обов’язковий ілюстративний матеріал

Перелік обов’язкового ілюстративного матеріалу:

До обов’язкового ілюстративного матеріалу конструкція мікрофонних решіток стаціонартного та портативного типу, структрурна схема АЦ-системи аудіолокації та ідентифікації об’єктів на місцевості.

7 Порядок контролю та прийому

Виконання етапів графічної та розрахункової документації БДР контролюється науковим керівником згідно зі встановленими термінами. Захист БДР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання БДР

При оформлюванні БДР використовуються:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104-2006 «Едина система конструкторської документації. Основні написи»;

— документами на які посилаються у вище вказаних.

Технічне завдання до виконання отримав _____ Савчук Д.А.

ДОДАТОК Б

Структрурна схема АЦ-системи аудіолокації та ідентифікації об’єктів на місцевості

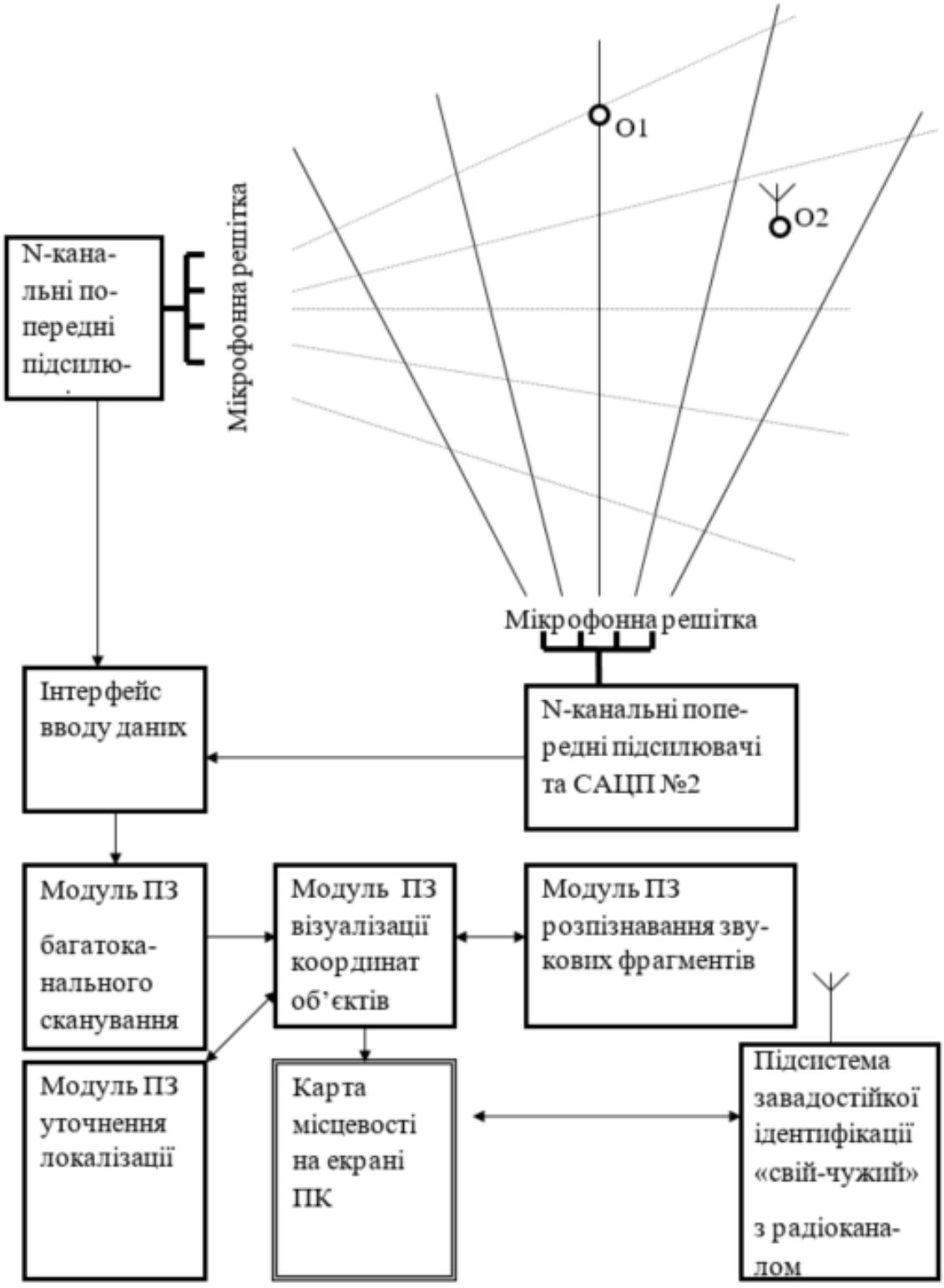


Рисунок Б — С труктрурна схема АЦ-системи аудіолокації та ідентифікації об’єктів на місцевості

ДОДАТОК В

Констркукція мікрофонних решіток стаціонарного та портативного типу

Рисунок В.1 — Зовнішній вигляд АЦ- системи портативного варіанту

Рисунок В.2 — Зовнішній вигляд АЦ- системи стаціонарного варіанту

ДОДАТОК Г

Лістинг класу для конвертації wav-файлу в масив даних

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace ConsoleApp10
{
   class Program
   {
      static void Main(string[] args)
      {
         ReadToByte();
      }



      static byte[] buffer;
      public static void ReadToByte()
      {
         buffer = File.ReadAllBytes("C:/Users/Vlad/Desktop/Diplom/Audio/Kanal4.wav");

         StreamWriter str = new StreamWriter("C:/Users/Vlad/Desktop/Diplom/Au-
dio/Kanal4.txt");

         for (int i = 0; i < buffer.Length; i++)
         {
```

```
            Console.Write($"{buffer[i]} ");
            str.WriteLine(buffer[i]);
        }

        Console.WriteLine();
    }
  }
}
```

ДОДАТОК Ґ

Лістинг основного коду програми

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Net.Mime;
using ILGPU;
using ILGPU.Runtime;
using ILGPU.Runtime.Cuda;

namespace CudaDiplome
{
   namespace Dimon
   {
      class MyClass
      {
         static bool tsufra;
         static int count;

         static string file;


         static byte[] kanal1 = new byte[] { };// масив інтів 1
         static byte[] kanal2 = new byte[] { };
         static byte[] kanal3 = new byte[] { };
```

```
        static byte[] kanal4 = new byte[] { };
        static byte[] zatrumku = new byte[] { };

        static byte[,] results = new byte[,] { };

        static List<string> list = new List<string>();//сначала значення записуються в ліст, а з нього конвертяться в масив інтів 3

        static StreamReader reader1;//створення стрім рідерів 4
        static StreamReader reader2;
        static StreamReader reader3;
        static StreamReader reader4;
        static StreamReader reader5;

        static byte[] chane2 = new byte[] { };
        static byte[] chane3 = new byte[] { };
        static byte[] chane4 = new byte[] { };
        static byte[] chane1 = new byte[] { };
        static byte[] zatrum = new byte[] { };

        //static int[] chane1

        /*static void Kernel(Index1D i, ArrayView<int> data, ArrayView<int> output)
        {
            output[i] = data[i % data.Length];
        }*/
        public static void CudaRun()
        {
            using Context context = Context.Create(builder => builder.Cuda());
```

```
            foreach (Device d in context.GetCudaDevices())
            {
                using Accelerator accelerator = d.CreateAccelerator(context);
                Console.WriteLine(accelerator);
                Console.WriteLine(GetInfoString(accelerator));
            }

        }
        private static string GetInfoString(Accelerator a)
        {
            StringWriter infoString = new StringWriter();
            a.PrintInformation(infoString);
            return infoString.ToString();
        }

        public static void Init()
        {
            reader1 = new StreamReader("Kanal1.txt");//передача в конструктор путі до
фалу 5
            reader2 = new StreamReader("Kanal2.txt");
            reader3 = new StreamReader("Kanal3.txt");
            reader4 = new StreamReader("Kanal4.txt");
            reader5 = new StreamReader("zatrumku.txt");

            file = reader1.ReadToEnd();// зчитуємо строку із фалу та записоємо в строку
6
            WorkWithFile(file, list, out kanal1);// передаємо строку до методу
WorkWithFile, що на виході дає нам масив Інт 7
```

```
    file = reader2.ReadToEnd();
    WorkWithFile(file, list, out kanal2);

    file = reader3.ReadToEnd();
    WorkWithFile(file, list, out kanal3);

    file = reader4.ReadToEnd();
    WorkWithFile(file, list, out kanal4);

    file = reader5.ReadToEnd();
    WorkWithFile(file, list, out zatrumku);

    ShowElementsInMasuv(kanal1);
    ShowElementsInMasuv(kanal2);
    ShowElementsInMasuv(kanal3);
    ShowElementsInMasuv(kanal4);
    Console.Write("\nzatrumku: ");
    ShowElementsInMasuv(zatrumku);

    Paralel();
    //MathOperation(out results);
    Console.WriteLine();
    //ShowResults(results);

}

static void Kernel(Index1D i, ArrayView<int> kanal1, ArrayView<int> output)
{
    output[i] = kanal1[i % kanal1.Length];
```

```
        }

        public static void LIMain()
        {
            using Context context = Context.Create(builder => builder.Cuda());
            Accelerator accelerator = context.CreateCudaAccelerator(0);

            ArrayView1D<int, Stride1D.Dense> chanel1 = accelerator.Allocate1D(kanal1);
            ArrayView1D<int, Stride1D.Dense> chanel2 = accelerator.Allocate1D(kanal2);
            ArrayView1D<int, Stride1D.Dense> chanel3 = accelerator.Allocate1D(kanal3);
            ArrayView1D<int, Stride1D.Dense> chanel4 = accelerator.Allocate1D(kanal4);
            ArrayView1D<int, Stride1D.Dense> zatrumka =
accelerator.Allocate1D(zatrumku);

            Action<Index1D, ArrayView<int>, ArrayView<int>> loadedKernel =
accelerator
                .LoadAutoGroupedStreamKernel<Index1D, ArrayView<int>,
ArrayView<int>>(Kernel);

            //loadedKernel((int)chanel1.Length, chanel1, chanel1);
            //loadedKernel((int)chanel2.Length, chanel2, chanel2);
            // loadedKernel((int)chanel3.Length, chanel3, chanel3);
            //loadedKernel((int)chanel4.Length, chanel4, chanel4);
            // loadedKernel((int)zatrumka.Length, zatrumka, chanel4);
            //accelerator.Synchronize();


            var buffer1 = accelerator.Allocate1D<byte>(kanal1);
            chane1 = buffer1.GetAsArray1D();
```

```
            var buffer2 = accelerator.Allocate1D<byte>(kanal2);
            chane2 = buffer2.GetAsArray1D();

            var buffer3 = accelerator.Allocate1D<byte>(kanal3);
            chane3 = buffer3.GetAsArray1D();

            var buffer4 = accelerator.Allocate1D<byte>(kanal4);
            chane4 = buffer4.GetAsArray1D();

            var buffer5 = accelerator.Allocate1D<byte>(zatrumku);
            zatrum = buffer5.GetAsArray1D();

            /*for (int i = 0; i < chanel1.Length; i++)
            {
                Console.WriteLine($"Math results: {chanel1[i]} {chanel2[i]} {chanel3[i]}
{chanel4[i]}");
            }*/

          /* int lenghtKanal = chane1.Length;
           int lenghtZatrumku = zatrum.Length;

           results = new int[lenghtKanal, lenghtZatrumku];


           for (int i = 0; i < lenghtKanal; i++)
           {
              for (int j = 0; j < lenghtZatrumku; j++)
              {
```

```
                int koeficientA = i - (j * 2);
                int koeficientB = i - j;
                int koeficientC = i + j;
                int koeficientD = i + (j * 2);

                int A = koeficientA >= 0 && koeficientA < lenghtKanal ?
kanal1[koeficientA] : 0;
                int B = koeficientB >= 0 && koeficientB < lenghtKanal ?
kanal1[koeficientB] : 0;
                int C = koeficientC >= 0 && koeficientC < lenghtKanal ?
kanal1[koeficientC] : 0;
                int D = koeficientD >= 0 && koeficientD < lenghtKanal ?
kanal1[koeficientD] : 0;

                results[i, j] = (A + B + C + D) / 4;
            }
        }
        */

        //int lenthKanal = chanel1.Length;

        /*// Initialize ILGPU.
        Context context = Context.CreateDefault();
        Accelerator accelerator = context.GetPreferredDevice(preferCPU:
false).CreateAccelerator(context);

        // Load the data.
        MemoryBuffer1D<int, Stride1D.Dense> chanel1 =
accelerator.Allocate1D(kanal1);
```

```
        MemoryBuffer1D<int, Stride1D.Dense> chanel2 =
accelerator.Allocate1D(kanal2);
        MemoryBuffer1D<int, Stride1D.Dense> chanel3 =
accelerator.Allocate1D(kanal3);
        MemoryBuffer1D<int, Stride1D.Dense> chanel4 =
accelerator.Allocate1D(kanal4);
        MemoryBuffer1D<int, Stride1D.Dense> zatrumka =
accelerator.Allocate1D(zatrumku);

        MemoryBuffer1D<int, Stride1D.Dense> deviceOutput =
accelerator.Allocate1D<int>(1000_000);

        // load / precompile the kernel
        Action<Index1D, ArrayView<int>, ArrayView<int>> loadedKernel =
accelerator.LoadAutoGroupedStreamKernel<Index1D, ArrayView<int>,
ArrayView<int>>(Kernel);

        // finish compiling and tell the accelerator to start computing the kernel
        loadedKernel((int)deviceOutput.Length, chanel1.View, deviceOutput.View);
        loadedKernel((int)deviceOutput.Length, chanel2.View, deviceOutput.View);
        loadedKernel((int)deviceOutput.Length, chanel3.View, deviceOutput.View);
        loadedKernel((int)deviceOutput.Length, chanel4.View, deviceOutput.View);
        loadedKernel((int)deviceOutput.Length, zatrumka.View, deviceOutput.View);

        // wait for the accelerator to be finished with whatever it's doing
        // in this case it just waits for the kernel to finish.
        accelerator.Synchronize();*/

    }
```

```
public static void WorkWithFile(string name, List<string> list, out byte[] kanal)// метод записує в масив інтів значення із файлу 2.. в тг
        {
        tsufra = false;
        count = 0;
        list.Clear();

        for (int i = 0; i < name.Length; i++)
        {
           if (Convert.ToInt32(name[i]) >= 48 && Convert.ToInt32(name[i]) <= 57)
           {
              if (tsufra == false)
              {
                 list.Add(name[i].ToString());
              }
              else
              {
                 list[count] += name[i];
              }
              tsufra = true;
           }
           else
           {
              if (tsufra == true)
              {
                 count++;
              }
              tsufra = false;
```

```
        }
    }

    kanal = new byte[list.Count];

    for (int i = 0; i < list.Count; i++)
    {
        kanal[i] = Convert.ToByte(list[i]);
    }
}

public static void ShowElementsInMasuv(byte[] masuv)
{
    for (int i = 0; i < masuv.Length; i++)
    {
        Console.Write($"{masuv[i]}; ");
    }
    Console.WriteLine();
}

public static void ShowResults(byte[,] results)
{
    int rows = results.GetUpperBound(0) + 1;
    int columns = results.Length / rows;

    string res = string.Empty;

    for (int i = 0; i < rows/10; i++)
    {
```

```
            res += $"\nканал № {i+1}\n";
            for (int j = 0; j < columns; j++)
            {
                res += $"затримка № {j+1} - {results[i, j]}. ";
            }
        }
        Console.WriteLine(res);
    }

    public static void Paralel()
    {
        int processors_count = Environment.ProcessorCount;
        results = new byte[kanal1.Length, zatrumku.Length];

        //for (int i = 1; i< processors_count+1; i++)
        //{
        //    Thread thread = new Thread(() =>
SomeMethod(i,processors_count,kanal1.Length));
        //}
        Thread thread = new Thread(() =>MathOperation_2(0,2500));
        Thread thread_1 = new Thread(() => MathOperation_2(2500,5000));
        Thread thread_2 = new Thread(() => MathOperation_2(5000,7500));
        Thread thread_3 = new Thread(() => MathOperation_2(7500,10000));
        thread.Start();
        thread_1.Start();
        thread_2.Start();
        thread_3.Start();

        //Thread.Sleep(1000);
```

```
    thread_3.Join();
    ShowResults(results);
}
public static void SomeMethod(int number,int processor_count, int lenght)
{
    int count = lenght / processor_count;

    int index_1 = number * count - count;
    int index_2 = number * count;

    MathOperation_2(index_1, index_2);
}

public static void MathOperation_2(int index_1, int index_2)
{
    int lenghtKanal = kanal1.Length; // було kanal1.Length
    int lenghtZatrumku = zatrumku.Length;
    int
    for (int i = index_1; i < index_2; i++)
    {
        for (int j = 0; j < lenghtZatrumku; j++)
        {
            int koeficientA = i - (j * 2);
            int koeficientB = i - j;
            int koeficientC = i + j;
            int koeficientD = i + (j * 2);
```

```
                int A = koeficientA >= 0 && koeficientA < lenghtKanal ?
kanal1[koeficientA] : 0;
                int B = koeficientB >= 0 && koeficientB < lenghtKanal ?
kanal1[koeficientB] : 0;
                int C = koeficientC >= 0 && koeficientC < lenghtKanal ?
kanal1[koeficientC] : 0;
                int D = koeficientD >= 0 && koeficientD < lenghtKanal ?
kanal1[koeficientD] : 0;

                results[i, j] = (byte)((A + B + C + D) / 4);
            }
        }
    }

    public static void MathOperation(out byte[,] results)
    {
        //int lenghtKanal = chane1.Length;
        int lenghtKanal = kanal1.Length; // було kanal1.Length
        int lenghtZatrumku =  zatrumku.Length; // було zatrumku.Length
        //int lenghtZatrumku = zatrum.Length;

        results = new byte[lenghtKanal, lenghtZatrumku];


        for (int i = 0; i < lenghtKanal;i++)
        {
            for (int j = 0; j < lenghtZatrumku; j++)
            {
                int koeficientA = i - (j * 2);
```

```
                int koeficientB = i - j;
                int koeficientC = i + j;
                int koeficientD = i + (j * 2);

                int A = koeficientA >= 0 && koeficientA < lenghtKanal ?
kanal1[koeficientA] : 0;
                int B = koeficientB >= 0 && koeficientB < lenghtKanal ?
kanal1[koeficientB] : 0;
                int C = koeficientC >= 0 && koeficientC < lenghtKanal ?
kanal1[koeficientC] : 0;
                int D = koeficientD >= 0 && koeficientD < lenghtKanal ?
kanal1[koeficientD] : 0;

                results[i, j] = (byte)((A + B + C + D) / 4);
            }
        }

    }
}
class Program
{
    public static void Main(string[] args)
    {
        MyClass.CudaRun();
        MyClass.Init();
        MyClass.LIMain();
    }
}
} }
```

ДОДАТОК Е

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА
НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Комп'ютерна аналого-цифрова система опрацювання низькочастотних си-гналів з використанням CUDA-технологій

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 93,2% Схожість 6,8%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ _____
(підпис) (прізвище, ініціали)

Керівник роботи _____ _____
(підпис) (прізвище, ініціали)