Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інфокомунікацій,радіоелектроніки та наносистем
(повне найменування інституту, назва факультету (відділення))

Кафедра радіотехніки
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка
до магістерської кваліфікаційної роботи

«Магістр»
(освітньо-кваліфікаційний рівень)

на тему: **«Розробка автономного пристрою на Nvidia для задач штучного інтелекту»**

Виконав: студент 2-го курсу, групи РТ-19м
спеціальності 172 – Телекомунікації та радіотехніка_Освітня програма - Радіотехніка_
(шифр і назва напряму підготовки, спеціальності)
Дука В.А.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. РТ
Гаврілов Д.В.
(прізвище та ініціали)
«____» _____ 2020 р.

Рецензент: к.т.н., професор
Васильківський М.В.
(прізвище та ініціали)
«____» _____ 2020 р.

Вінниця ВНТУ - 2020 рік

Вінницький національний технічний університет
Факультет Інфокомунікацій , радіоелектроніки та наносистем
Кафедра Радіотехніки
Освітньо-кваліфікаційний рівень Магістр
Спеціальність 172 – Телекомунікації та радіотехніка
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри РТ
д.т.н., професор О.В. Осадчук
"___" _____20__ року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Дуці Владиславу Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка автономного пристрою на Nvidia для задач штучного інтелекту»

керівник роботи Гаврілов Дмитро Володимирович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від 25.09.2020 року №214

2. Строк подання студентом роботи 10.12.2020 року.

3. Вихідні дані: кількість ядер NVIDIA CUDA 128; час обробки, не більше 1 мс;споживана потужність, не більше 5 ВА; живлення пристрою має відбуватися від джерела живлення ЕОМ з напругами ± 5В; вид обміну інформацією паралельний; час готовності до роботи, не більше 0,3 хв; память LPDDR4, 64-біт 4 Гб; Флеш-пам'ять eMMC 16 Гб; інформаційна ємність даних 270 біт; вид передачі даних неперервний; маса пристрою, не більше 0,2 кг; габаритні розміри плати, не більше 70×45×50 мм

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Вступ. Техніко-економічне обґрунтування. Аналіз існуючих рішень і платформ. Архітектура системи, що розробляється. Програмна та апаратна реалізація. Результати моделювання. Цивільна оборона. Безпека життєдіяльності. Організаційно-економічний розділ. Висновки.

5.Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): Технічне завдання. Нейронні мережі. Організація пам'яті в CUDA. Архітектура апаратної платформи Nvidia. Опис алгоритму розрахунку схем адвекції. Апаратна реалізація. Результати моделювання. Лістинги програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Основна частина	к.т.н., доцент Гаврілов Д.В.		
Охорона праці та безпека в надзвичайних ситуаціях	к.т.н., доцент Березюк О. В.		
Економічна частина	к.т.н., доцент Адлер О. О.		

7. Дата видачі завдання 03.09.2020 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Огляд літературних джерел. Вибір та узгодження теми МКР	03.09.2020-14.09.2020	
2.	Аналіз літературних джерел. Попередня розробка основних розділів	15.09.2020-21.09.2020	
3.	Затвердження теми. Розробка технічного завдання	21.09.2020-25.09.2020	
4.	Аналіз вирішення поставленої задачі. Розробка структурної схеми	26.09.2020-09.10.2020	
5.	Електричні розрахунки. Експериментальне дослідження	10.10.2020-25.10.2020	
6.	Розділ моделювання	26.10.2020-04.11.2020	
7.	Розробка графічної частини МКР	05.11.2020-15.11.2020	
8.	Аналіз економічної ефективності розробки	16.11.2020-19.11.2020	
9.	Охорона праці (ОП)	19.11.2020-22.11.2020	
10	Оформлення пояснювальної записки та графічної частини	23.11.2020-29.11.2020	
11.	Нормоконтроль	30.11.2020-01.12.2020	
12.	Попередній захист МКР, доопрацювання, рецензування МКР	02.12.2020-04.12.2020	

Студент _____ Дука В.А.
(підпис)

Керівник роботи _____ Гаврілов Д.В.
(підпис)

РЕФЕРАТ

УДК 621.396

Дука В.А. Розробка автономного пристрою на Nvidia для задач штучного інтелекту / Магістерська кваліфікаційна робота / Вінниця: ВНТУ, 2020, – 132 с. Укр. мовою.

Бібліограф. найменувань 24, ілюстрацій 41, таблиць 37.

Метою **магістерської кваліфікаційної роботи** є проектування та розробка автономного пристрою на Nvidia для задач штучного інтелекту, із метою подальшого виготовлення.

Методи дослідження - застосування сучасних САПР, порівняльний аналіз, використання існуючих баз даних для створення конструкції приладу.

Результатом розробки є частина технічної документації, необхідна для виготовлення та експлуатації вимірювальних приладів.

В **магістерській кваліфікаційній роботі** наводиться необхідний матеріал для підтвердження актуальності розробки, наводиться принцип дії із висвітленням наукової новизни, обраховані основні електричні, конструктивні та технологічні характеристики виробу, а також електромагнітна сумісність та захист. Наведене обґрунтування загального конструкторського рішення, обґрунтування вибору комплектуючих та матеріалів.

Проведено техніко-економічне обґрунтування доцільності розробки. Розраховано економічний ефект від розробки та впровадження пристрою. Розглянуті питання безпеки життєдіяльності під час виготовлення пристрою та стійкості його роботи при дії електромагнітного імпульсу та іонізуючого випромінювання.

Ключові слова: штучний інтелект, Nvidia, Cuda, Jetson.

ABSTRACT

UDC 621.396

Duka VA Development of an autonomous device on Nvidia for artificial intelligence tasks / Master's thesis / Vinnytsia: VNTU, 2020, - 132 p. Ukr. language.

Bibliographer. names 24, illustrations 41, tables 37.

The purpose of the master's thesis is to design and develop a standalone device on Nvidia for artificial intelligence tasks, with the aim of further manufacturing.

Research methods - the use of modern CAD, comparative analysis, the use of existing databases to create the design of the device.

The result of the development is a part of the technical documentation required for the manufacture and operation of measuring instruments.

The master's qualification work provides the necessary material to confirm the relevance of the development, provides the principle of operation with coverage of scientific novelty, calculated the basic electrical, structural and technological characteristics of the product, as well as electromagnetic compatibility and protection. The substantiation of the general design decision, the substantiation of a choice of accessories and materials is resulted.

Feasibility study of development expediency is carried out. The economic effect of the development and implementation of the device is calculated. The issues of life safety during the manufacture of the device and the stability of its operation under the action of electromagnetic pulse and ionizing radiation are considered.

Keywords: artificial intelligence, Nvidia, Cuda, Jetson.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

- **GPGPU** (***General - purpose graphics processing units)*** - техніка використання графічного процесора для загальних обчислень, які зазвичай проводить центральний процесор

- **GPU** *(graphics processing unit)* - графічний процесор

- **CPU** *(central processing unit)* - центральний процесор

- **SIMD** *(Single Instruction stream Multiple Data stream)* метод обчислень, що означає, що ядра GPU виконують один і той же набір інструкцій для кожного екземпляра даних.

- **GFlops -** позасистемна одиниця, використовувана для виміру продуктивності комп'ютерів, показує, скільки операцій з плаваючою комою в секунду виконує ця обчислювальна система.

- **CUDA** *(англ. Compute Unified Device Architecture)* — розроблена компанією NVIDIA програмно-апаратна архітектура, що дозволяє проводити обчислення з використанням графічних процесорів NVIDIA, що підтримують технологію GPGPU.

- **OpenCL** *(від англ. Open Computing Language — відкрита мова обчислень)* — фреймворк для написання комп'ютерних програм, пов'язаних з паралельними обчисленнями на різних графічних (англ. GPU) і центральних процесорах (англ. CPU).

ВСТУП

Актуальність теми. Людство генерує різну інформацію, і з кожним днем, швидкість і об'єми створюваної інформації тільки збільшуються. На сьогодні інформаційна сфера є невід'ємною частиною життя практично будь-якої людини. Проте, кількість існуючої сьогодні інформації, і інформації, дійсно потрібної людині, сильно відрізняється, тобто із загальної кількості усієї доступної інформації людині необхідно відокремити саме те, що йому треба. Звичайний людський мозок не здатний обробити і малої долі всього, що існує сьогодні, звідси ми отримуємо першу проблему – проблему отримання, вибору, аналізу інформації.

Щоб спробувати розв'язати цю проблему, людство винаходило різні пристрої, пристосування, що дозволяють зменшити навантаження на людину. Вінцем цих винаходів стали ЕОМ, що дозволяють зберігати і обробляти величезні об'єми різної інформації. Проте, сама інформація не така важлива, як зрозуміти, що робити, маючи цю інформацію. Потрібні системи аналізу даних і ухвалення рішень на їх основі. Ця концепція систем ухвалення рішень дозволяє працювати з великою кількістю даних, аналізуючи вхідну інформацію, і що видає в якості результату певну дію або порядок дій, необхідних людині. Ці системи отримали величезне поширення в різних сферах, в яких потрібне оперативне отримання кінцевих даних, наприклад в економіці, медицині, в сферах моделювання різних систем і процесів, а так само в системах контролю роботи підприємств і системах безпеки.

Окрім отримання необхідного результату, системи ухвалення рішень можуть надавати користувачеві різні прогнози, а так само, у випадках некоректних, або не повністю відповідних даних, видавати зразкові дані, понад усе відповідні для ситуації, що склалася. Так людина дістає можливість оцінити отримані результати і прийняти оптимальне рішення.

На сьогодні, на жаль, не усі завдання можливо точно формалізувати, і створити алгоритм для конкретного завдання, оскільки частенько завдання не

можна алгоритмізувати, або ж виконання розрахунків для отримання точного значення займуть набагато більше часу, ніж необхідно. У ситуаціях такого типу, замість точного рішення, правильніше буде оцінити найбільш важливі критерії, виділити ключові значення і присвоїти їм найбільш відповідні зразкові значення, залежно від стану усієї системи. При цьому варто відмітити, що показники стану системи потрібні не лише для цього моменту, але і минулі стани, щоб шляхом аналізу, зрозуміти, як система прийшла в цей стан, які були для цього передумови, а так само «запам'ятати» наслідки, щоб використати цю конфігурацію значень надалі. Для будь-якої системи можна спробувати виділити циклічність станів, завдяки чому подальший аналіз проходитиме простіше і швидше.

Проте, не дивлячись на таке широке охоплення виконуваних завдань, системи ухвалення рішень по своїй специфіці не є автономними і не можуть працювати без участі людини. Все одно потрібне втручання людини. Для створення повністю автономної системи потрібний апарат, математична модель, що дозволяє аналізувати інформацію і приймати рішення, з урахуванням навчання цієї системи. Таким критеріям відповідають нейронні мережі. Створені як комп'ютерна симуляція роботи мозку, завдяки можливості навчання, нейронні мережі можуть адаптуватися до нових даних і різних змін, приймаючи при цьому вірні рішення при повній автономії. Таким чином, нейронні мережі дозволяють добитися високої точності отримуваних результатів при різних конфігураціях вхідних даних, що змінюються.

Створення нейронної мережі відбувається в 2 етапи, це створення нейронів і зв'язків між ними, а потім, процес навчання мережі, проте найбільш важливий саме процес навчання. Нейронні мережі за способом навчання діляться на 2 основні класи:

- навчаються з учителем;
- навчаються без учителя.

Також, нейронні мережі можна класифікувати залежно від поставлених

завдань, наприклад за типом даних, що вводяться, або по виду вихідній інформації. Так одні з найпоширеніших мереж – мережі, що аналізують зображення, або, наприклад, системи розпізнавання рукописного тексту. У цій роботі будуть розглянуті логічні нейронні мережі. Базисом для таких мереж є алгебра логіки. Запропонований Жаком Ербраном і розроблений Джоном Робінсоном, алгоритм припускає доведення теореми методом «від осоружного», через пошук протиріччя у вхідній інформації. Спочатку метод був реалізований перебором безлічі вхідних даних до тих пір, поки не знаходилося шукане протиріччя, або цикл не зупинявся. Проте, коли Японія оголосила про старт проекту «ЕОМ п'ятого покоління», для реалізації логічних систем був перероблений метод резолюції для паралельної реалізації. Цей метод утілив Ehud Shapiro (Ехуд Шапіро) в проекті «Flat Concurrent PROLOG», паралельній реалізації логічної мови програмування «PROLOG». Розпаралелювання цього методу дозволяє провести резолюцію відразу над усією вхідною множиною за 1 раз. Таким чином, системи логічного виведення отримали своє паралельне втілення, у тому числі у вигляді логічних нейронних мереж.

Нейронна мережа є паралельною структурою, що полягає, в мінімальній конфігурації, з вхідного і вихідного шару. Навчання і робота нейронної мережі відбувається за схемою SIMD (Single Instruction – Multiple Data), що означає виконання однієї функції(інструкції) на деякому наборі різних вхідних даних. Таким чином, робота нейронної мережі безпосередньо залежить від використовуваних алгоритмів роботи, і природно, при реалізації паралельних алгоритмів навчання і роботи, швидкість виконання поставлених завдань буде набагато більше.

Для реалізації будь-якого паралельного алгоритму, будь-яких паралельних процесів, потрібна відповідна архітектура ЕОМ, що дозволяє одночасне виконання завдань. Відповідними конфігураціями для реалізації паралельної роботи є багатоядерні, багатопроцесорні обчислювальні машини і комплекси, а так само архітектура, що дозволяє використати графічне ядро

для обчислень, такі як CUDA (Nvidia), C++ AMP (Microsoft), OpenCL, OpenACC і AMD FireStream. Використання процесора апаратної платформи Nvidia для обчислень відрізняється від використання декількох ядер центрального процесора лише тим, що апаратна платформа Nvidia має в рази більшу кількість ядер, тобто мають більше можливостей для паралелізму виконуваних завдань. Проте варто відмітити той факт, що при великій кількості ядер, кожне ядро саме по собі досить слабке, тому виконувати на апаратних платформах Nvidia варто тільки процеси, які не потребують великої продуктивності.

Процеси навчання і роботи мережі за своєю суттю не є дуже вимогливими до системи, а швидкодія залежатиме лише від кількості оброблюваної інформації. Таким чином, логічні нейронні мережі можуть бути реалізовані з використанням технології CUDA для того, щоб досягти максимальної продуктивності.

Метою роботи є реалізація логічної нейронної мережі з використанням паралельного методу логічного слідування, виконана на мові C++ із застосуванням CUDA технології.

Об'єктом дослідження є розробка теоретичних засад, методів та засобів для задач штучного інтелекту на основі автономного пристрою на Nvidia.

Предметом дослідження – є автономні пристрої на Nvidia для задач штучного інтелекту.

В магістерській кваліфікаційній роботі для досягнення поставленої мети **розв'язуються такі завдання**:

1. Розглянути моделі нейронних мереж, їх реалізації і основні ідеї.

2.Визначити клас логічних нейронних мереж і виконуваних ними завдань.

3. Розглянути послідовний і паралельний методи резолюції, як засоби логічного виведення.

4. Встановити залежність між логічним слідуванням і логічним виведенням.

5. Проаналізувати отриману інформацію, і на її основі викласти ідеї паралельної реалізації логічної нейронної мережі і методу логічного слідування.

6. Реалізувати логічну нейронну мережу для завдання логічного слідування.

Методи дослідження ґрунтуються на використанні:

а) основних положень теорії функції комплексної змінної (створення математичних моделей);

б) диференціального та інтегрального числення (створення математичних моделей);

в) методів розрахунку лінійних електричних кіл з використанням матриць (електричні розрахунки);

г) ЕОМ для обрахунків та проведення моделювання.

Наукова новизна одержаних результатів

Наукова новизна роботи полягає в отриманні таких результатів:

1. Запропоновано новий підхід при побудові автономного пристрою на Nvidia для задач штучного інтелекту з застосуванням спеціалізованих мікросхем.

2. За рахунок використання сучасної елементної бази вдалося вдосконалити автономний пристрій на Nvidia для задач штучного інтелекту.

3. У порівнянні з іншими серійними платформами для задач штучного інтелекту, які побудовані на іншій елементній базі, пристрій побудований з застосуванням швидкодіючих цифрових пристроїв, що дає змогу добитися високої надійності та більшої стабільності процесі роботи.

4. Удосконалено математичні моделі елементів схеми, що застосовуються при моделювання у САПР, які, на відміну від існуючих, враховують зміни температури та стабільності напруги живлення.

5. Проведені експериментальні дослідження підтвердили вірність електричних розрахунків схем пристрою з похибкою, що відповідає вимогам технічного завдання.

Практичне значення отриманих результатів полягає у тому, що нейромережеві технології є передовими у багатьох областях, проте для отримання максимальної ефективності, необхідно використати новіші методи реалізації, такі як паралельне програмування, а зокрема, реалізації паралельної роботи центрального процесора і графічного ядра.

Звичайному призначеному для користувача комп'ютеру з двох або чотириядерним CPU може знадобитися для таких розрахунків декілька днів, а іноді тижнів або місяців. Найчастіше нам необхідно оперативно отримувати результати обробки (наприклад, для обробки сейсмічних даних або кодування відео в режимі online).

Деякі завдання наукового характеру здатні розрахувати негайно лише одиниці суперкомп'ютерів у всьому світі, а найбільш складні вирішуються за допомогою об'єднання високопродуктивних серверних станцій в кластери. Проте вартість такого устаткування занадто висока.

При виконанні цих операцій використовують паралельні обчислення, розподіляючи навантаження між ресурсами CPU і процесорами комп'ютерів кластера.

Розробники пристроїв знайшли вихід з такої ситуації. Вони дозволили збільшити на декілька порядків обчислювальні можливості комп'ютерів, пропорційно зменшивши витрати на програмне забезпечення.

Особистий внесок. Рішенням задач штучного інтелекту є технологія GPGPU, що дозволяє використати ресурси платформи для неграфічних обчислень. І сьогодні ця технологія стає усе більш актуальною. Основними конкурентами-виробниками платформ з апаратною підтримкою GPGPU є NVIDIA з комплексом CUDA и AMD з ATI Stream Technology.

CUDA підтримується в усіх будовах NVIDIA, починаючи з простих призначених для користувача, не кажучи вже про спеціалізовані потужні

пристрої. А також прискорення за допомогою CUDA вже використовують багато призначених для користувача застосувань. І такі технології припадають на кожен персональний комп'ютер.

Таким чином, актуальність цієї роботи полягає в розгляді паралельної реалізації алгоритмів з використанням технології CUDA.

Тому програмування, з використанням ресурсів платформи неймовірно актуально. Ми отримуємо велику продуктивність за мінімальну ціну. Мета роботи : за допомогою технології CUDA максимально прискорити алгоритм отримання позитивно певних схем адвекції шляхом нелінійного перенормування потоків і тим самим показати переваги використання вищезгаданої технології для складних обчислень.

1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ І ПЛАТФОРМ

1.1 Штучні нейронні мережі

Уперше нейронні мережі були згадані в роботі Уорена Мак-Калока і Уолтера Пітса «Logical calculas of the ideas immanent in nervous activity» («Логічне обчислення ідей, що відносяться до нервової активності»). У статті було формалізовано поняття нейронної мережі і описана робота нейронів формулами математики і алгебри логіки, виходячи зі знань про будову нервової системи. Було відмічено, що штучний нейрон є простим логічним пристроєм, проте без навчання, не здатний виконувати ніяких функцій. Алгоритм навчання був запропонований Дональдом Хебом в книзі «Організація поведінки : нейропсихологічна теорія» і припускав, що при частій стимуляції нервової системи утворюються нейронні структури.

Перший одношаровий перцептрон з'явився в 1958 році, завдяки Френку Розенблату і мав можливість виконувати завдання класифікації, через що набув популярності в різних сферах. Після публікації книги «Перцептрони» Марвином Мінським і Сеймуром Папертом в 1969 році, в якій детально наводився доказ неможливості навчання універсального перцептрона, і описувалися класи завдань, які перцептрон не міг вирішити, інтерес до нейронних мереж практично зник.

Дослідження в цій сфері продовжилися, і в 1972 році Теуво Кохонен і Джеймс Андерсон незалежно пропонують новий тип нейронних мереж. Так, до 1982 року, нейромережеві технології були в занепаді. Проте, модель мережі із зворотними зв'язками, створена Джоном Хопфілдом, показала нові можливості застосування нейронних мереж. В цей же час Теуво Кохонен запропонував моделі нейронної мережі, з можливістю навчання без учителя. Через 4 роки, двома незалежними групами учених, що складаються з Девіда Румельхарта з Джеффрі Хінтоном і Рональдом Вільямсом, і С.І. Барцева з В.А Охоніним, був наново вивчений і істотно розвинений метод зворотного

поширення помилки. З цієї миті в науковій сфері знову з'явився інтерес до розвитку технологій нейронних мереж, а завдяки широкому поширенню персональних комп'ютерів, дослідження в цій області вкрай прискорилися.

На сьогодні ми маємо більше двадцяти різних моделей нейронних мереж, які отримали застосування у багатьох сферах сучасного життя. Для завдань аналізу великої кількості даних, для трудноформалізованих і завдань, що не формалізуються, для завдань з комплексними алгоритмами і алгоритмами з неявними параметрами, можливе застосування нейронних мереж. При необхідності одночасного аналізу великого числа даних, виділенні тенденцій і прогнозування залежно від отриманої інформації, як, наприклад, на ринку паперів або валютному ринку, нейронні мережі отримали дуже широке поширення.

Проте, не дивлячись на усі можливості нейронних мереж і їх продуктивність, людина в рази більше адаптивна і здатна розпізнавати і обробляти речі, що абсолютно не формалізуються, такі як емоції, інтонацію, поведінку. Але штучні нейронні мережі мають значну перевагу перед людським мозком, завдяки здатності обробляти більший об'єм однотипної інформації, із-за своєї структури. Цей процес має перевагу за рахунок паралелізування виконуваних завдань, розбитих на простіші функції аж до простих бінарних операцій. Ще одним значимим фактом є те, що нейронні мережі не проводять занадто складних обчислень і ірраціональних операцій, характерних для послідовної архітектури.

Виходячи з цього, архітектуру нейронної мережі можна характеризувати схемою SIMD (Single Instruction - Multiple Data), тобто виконання однієї функції для безлічі даних. За такою схемою реалізується обчислювальний процес на графічних картах, а отже, архітектура графічної карти повністю підходить для реалізації на ній нейронних мереж. Проте перш ніж перейти до реалізації, слід розглянути її пристрій і алгоритми її роботи.

Структура штучного нейрона аналогічна біологічному. Нейрон має декілька входів, по яких сигнали поступають в сам нейрон. Вхідні сигнали

можуть бути отримані як із зовнішнього джерела, так і від інших нейронів. Кожен вхід нейрона має свою вагу, тому сигнал, що поступає, потрапляє в оброблювальний центр нейрона з деяким коефіцієнтом. Далі, після деяких перетворень, нейрон передає на вихід всього один сигнал. Отриманий вихідний сигнал може поступати або на наступний нейрон, або бути кінцевим результатом роботи для усієї мережі. Ця схема представлена на рисунку 1.1

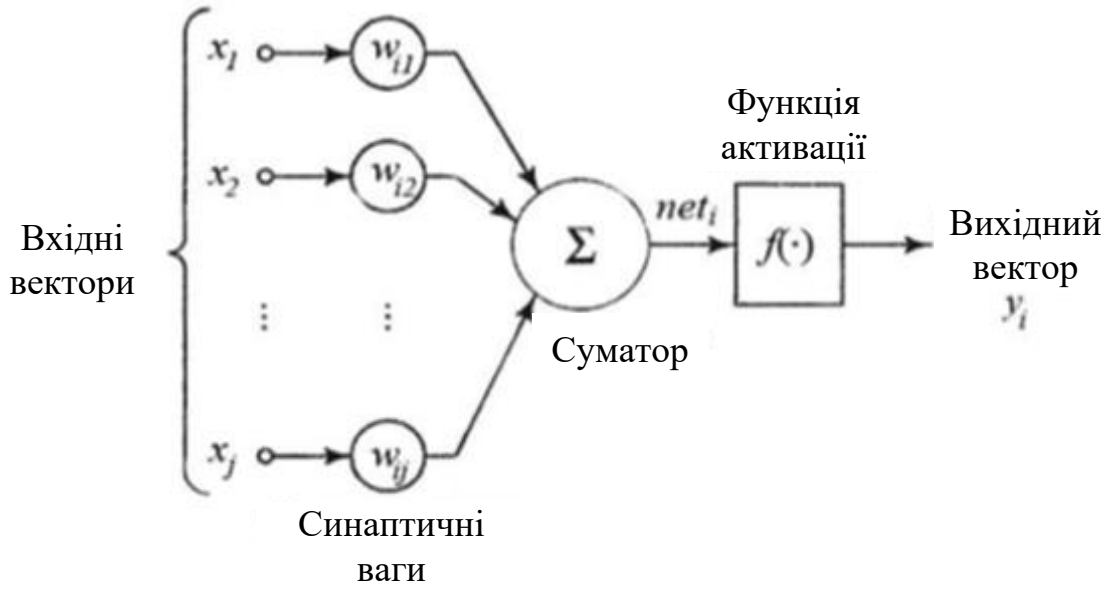


Рисунок 1.1 – Схема роботи штучного нейрона

Таким чином, нейрон обчислює вхідне значення базисною функцією

$$\mathrm{Neti}(\omega_{ij}, x\,j)$$

де значення x j - значення вхідних векторів, а ω ij - відповідні синаптичні ваги.

Базисні функції, що відповідають за обчислення вихідного значення, можуть дуже сильно розрізнятися, проте серед загальної множини можна виділити декілька, використовуваних найчастіше:

а) Лінійна: функція виконує операцію складання типу :

$$\text{Neti}\ (\omega\ \text{ij}, x\ \text{j}\ ) = \sum_{j=1}^{n} x_j \times \omega_{\text{ij}}$$

б) Радіальна: значення обчислюється за формулою відстані від вхідного значення до деякого « шаблонного»:

$$\text{Neti}\ (\omega\ \text{ij}, x\ \text{j}\ ) = \sqrt{\sum_{j=1}^{n} (x_j - \omega_{\text{ij}})^2}$$

В цілому, обмежень на вибір базисної функції немає. Підбір функції відбувається залежно від завдання, для отримання адекватної моделі.

Після виконання перетворення величини вхідного сигналу, нейрон посилає це значення на вихід. Вихідне значення підлягає аналізу активаційною функцією, що є свого роду фільтром. Активаційна функція оцінює отримане значення, і у разі задоволення певним умовам, значення буде передано далі. Основними активаційними функціями є:

а) Лінійні:

$$\text{F(net)} = \begin{cases} min, net \leq min \\ net, min < net < max \\ max, net \geq max \end{cases},$$

де net - поточне значення, min і max - деякі вибрані порогові значення.

б) Порогові функції типу «стрибок»:

$\text{F(net)} = \begin{cases} 0, net \leq 0 \\ 1, net > 0 \end{cases}$, в даному випадку значення порогу вибране рівним 0, проте можна виставити це значення рівним будь-кому, що підходить під модель, що реалізовується. Одна з поширених функцій такого типу:

$$\text{F(net)} = \text{sign(net)}, \text{sign(net)} = \begin{cases} -1, net < 0 \\ 0, net = 0 \\ 1, net > 0 \end{cases}$$

в) Сигмоїдальні функції:

$$F(net) = \frac{1}{1+e^{-k\times net}},$$

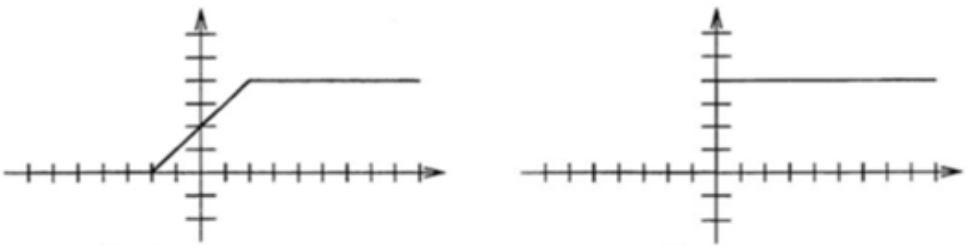
де « k» - коефіцієнт нахилу функції. Зазвичай прийнято брати k = 1. При виборі нескінченно великого значення k, функція вироджується в порогову.

д) Функція Гауса :

$$F(net) = e^{\frac{-(S-R)^2}{2\sigma^2}},$$

де S = ||x - c|| - « відстань» між значеннями вхідного вектору, і деякого заданого шаблону; R - параметр, що відповідає за зрушення функції по осі абсцис; σ - параметр, що визначає швидкість зміни значення функції, при віддаленні від центру.

Графічно, перераховані види функцій, відображені на рис. 1.2, а...г:



а) б)

в) г)

Рисунок 1.2 – Види функцій: а) лінійна, б) порогова, в) сигмоїдальна, г) Гаус

Окрім приведених активаційних функцій, існує безліч їх модифікацій і функцій інших видів. У деяких мережах активаційна функція не використовується, і вихідне значення обчислюється як F(net) = net.

Для реалізації структури мережі, необхідно ввести зв'язки між нейронами. Основні види зв'язків, використовуваних в нейронних мережах, наступні:

а) Прямий зв'язок:

Процес передачі вихідного сигналу відбувається послідовно, з виходу одного шару, на вхід наступного.

б) Зворотний зв'язок:

Вихідні сигнали передаються на вхід попереднього шару, що дозволяє проводити рекурентні операції на нейронних мережах.

в) Бічний зв'язок:

Вихідні сигнали передаються усередині шару від нейрона до нейрона.

Перш ніж почати створювати нейронну мережу, проводиться довгий аналіз завдання, в процесі якого визначаються топологія і розміри мережі, передатні і активаційні функції і інші параметри. Завершивши цей етап формалізації і перенесення поставленого завдання на модель нейронної мережі, необхідно почати процес навчання.

Навчання є ключовим етапом під час створення нейронної мережі, і є процесом налаштування параметрів, наприклад вибором вагів синоптичних зв'язків і порогів в активаційних функціях. Прийнято розрізняти 3 види способів навчання :

а) Навчання з учителем;

б) Навчання без учителя;

в) Навчання по алгоритму з фіксованими вагами.

Процес навчання з учителем полягає в тому, що нейронній мережі передаються вхідні дані, і набір заздалегідь вичислених вихідних значень. Таким чином, з отриманих значень складаються повчальні пари, а вихідна

функція проводить коригування, щоб з максимальною точністю наблизити реальний результат до очікуваного вихідного значенню. Система продовжуватиме навчання до тих пір, поки результати не досягнуть певної точності. Основним алгоритмом методу навчання з учителем є метод зворотного поширення помилки, що дозволяє рекурентно коригувати параметри мережі.

Для алгоритму навчання без учителя, характерна передача тільки вхідних даних. Алгоритм є ітеративним, а процес коригування закінчується у разі, якщо однакові входи відповідають однаковим виходам. Прикладом нейронної мережі, що навчається без учителя, служать карти, що самостійно організуються.

При навчанні нейронної мережі алгоритмом з фіксованими вагами, мережа визначає ваги по заздалегідь заданих методах розрахунку, обчислюючи їх по вхідних векторах.

В цілому, якщо коротко порівняти алгоритми, то результати, що отримуються шляхом навчання без учителя, є реалістичнішими, оскільки в цих процесах не йде порівняння із заданим результатом.

Навчання нейронної мережі витрачає багато часу, і чим складніше структура, тим довше проходитиме навчання. Для прискорення цього процесу існує лише одне рішення - поліпшення самого алгоритму навчання. Найбільш ефективною, з точки зору тимчасових витрат модифікацією для процесів навчання є розпаралелювання алгоритму.

Після завершення стадії навчання, структура нейронної мережі фіксується, тобто подальших змін вагів і зв'язків не відбувається, мережа переходить в робочий режим.

1.2 Логічні нейронні мережі

Першим кроком для створення нейронної мережі є аналіз вибраного завдання, яке ця мережа виконуватиме, тобто за встановленими вимогами

вибирається деякий стандартний набір функцій. Залежно від поставлених цілей, прийнято класифікувати нейронні мережі. Для систем аналізу зображень, завдань автоматизації управління, економічної сфери, існують типові нейронні мережі, які далі модифікуються під конкретно поставлене завдання. У цій роботі буде розглянутий клас логічних нейронних мереж.

Для того, щоб визначити, в чому полягає суть логічних нейронних мереж, розглянемо приклад завдання, що стоїть перед цією системою. Припустимо, що у нас є деяка безліч висловлювань, по якій шляхом неарифметичних перетворень має бути отриманий результат. Для роботи з подібними множинами, необхідно їх формалізувати, представивши усі висловлювання у вигляді предикатів. Так дістаємо можливість обробки предикатів за допомогою операцій алгебри логіки. Рішення в цьому завданні залежить від істинності або хибності висловлювань вхідної великої кількості. Виходячи з цього, логічну нейронну мережу можна охарактеризувати як нейронну мережу, базисом для якої є булева алгебра.

Продовжимо розгляд поставленого завдання для визначення функціонала і виду мережі. Якщо узяти сукупність усіх подій, і виділити серед них ті висловлювання, які повністю покривають смисловий діапазон, отримаємо нову множину. У тому випадку, якщо кожна допустима ситуація характеризується тим, що істинне значення приймає тільки одне висловлювання цієї сукупності, отриману множину називатиметься вичерпною безліччю подій. На основі отриманої множини, ввівши деяку ієрархію, будуватимемо логічні ланцюжки, що дозволяють однозначно отримати результат. Систему з таких логічних шляхів, у разі її несуперечності і повноти, можна розглядати як систему прийняття рішень. Проте, перш ніж перейти до побудови нейронної мережі із СПР, потрібний ще один крок у формалізації завдання - визначення величини вхідних сигналів.

Оскільки в мережі, нейрони представлені в деякому стандартному виді, то величини збудження, синапсичні зв'язки, пороги і функції мають бути однакові, ну або лежати в загальному діапазоні значень для усієї мережі.

Після цього кроку, можна приступити до вибору параметрів мережі.

Візьмемо передатну функцію деякого нейрона:

$$V = \varepsilon \sum_{j=1}^{n} (V_j \omega_{ij} - h_i),$$

де $V_i = \begin{cases} 1, V \geq 1 \\ V, иначе \end{cases}$; $\varepsilon(x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases}$, Vj - величина сигналу, що поступає на j- й вхід нейрона.

У моделі логічної нейронної мережі розглядаються 2 типи нейронів, що реалізовують логічні функції. Це нейрон- кон’юнктор (див. рис 1.3) і нейрон-диз’юнктор (див. рис. 1.4)

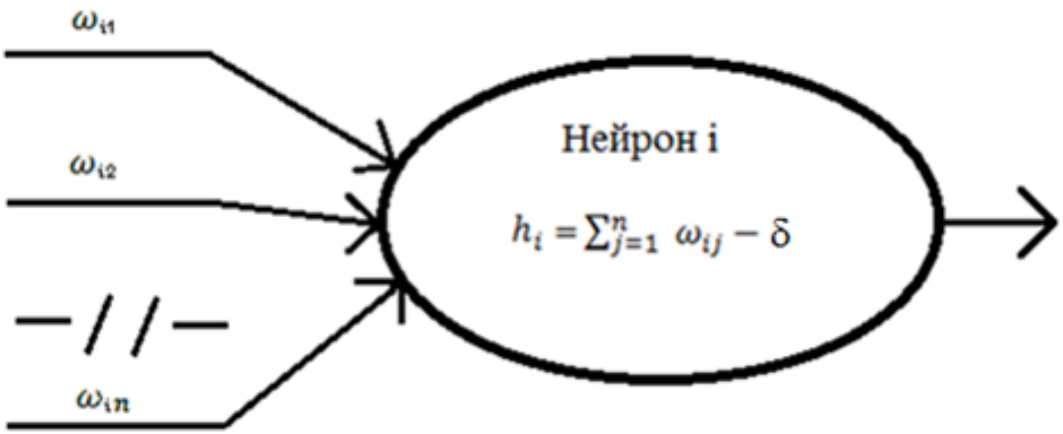


Рисунок 1.3 – Модель нейрона- кон’юнктора

Для реалізації моделі нейрона-кон’юнктора використовується високе значення порогу. Значення «δ» - поправка, необхідна для того, щоб для подолання порогу, сигнал поступав по усіх входах.

Модель нейрона-диз’юнктора виконується при низькому значенні порогу, але при високому значенні вагів. Ця умова забезпечує набуття

вихідного значення при хоч би одному вхідному сигналі, але у такому разі потрібна додаткова умова, що обмежує вихідне значення, тобто Vi ≤ 1. Під час процесу навчання диз'юнктора, передбачається отримання точних вхідних і вихідних даних, а отже логічна функція « АБО» перетворюється на те, що «виключає АБО», оскільки в даному випадку передбачається збудження всього на одному вході.

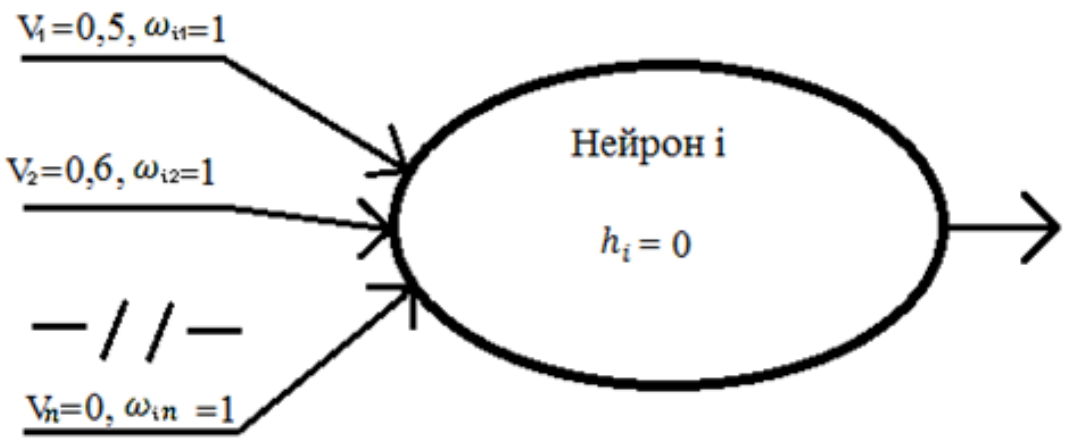


Рисунок 1.4 – Модель нейрона- диз'юнктора

Головною відмінністю логічних нейронних мереж від мереж інших видів є те, що не існує одного шаблону для реалізації усього класу логічних нейронних мереж. Кожна така мережа будується під завдання, шляхом об'єднання в мережі кон'юнкторів і диз'юнкторів. Тільки шляхом аналізу конкретного завдання може бути створена логічна нейронна мережа. Таким чином метою цієї роботи буде розробка конструктора, що дозволяє створити логічну нейронну мережу.

Проаналізувавши вхідну множину, необхідно виділити усі використовувані змінні, що подаються на вхід, після чого потрібно побудувати структуру, а потім навчити її. Перша частина навчання відбувається методом трасування, коли по еталону, що пред'являється мережі, вибудовується шлях від вхідних значень до вихідного. При пред'явленні декількох еталонів нейронна мережа будує шляхи послідовно, вибираючи ще не задіяні нейрони і вводячи необхідні зв'язки. Цей алгоритм

повторюється до тих пір, поки не буде досягнута повна визначеність для усіх вхідних еталонів, що пред'являються мережі. Для алгоритму трасування нейронної мережі використовується поняття матриці слідування - матриці, в якій рядки і стовпці - нейрони, а значення, що стоять на їх перетині відповідають за наявність або відсутність зв'язку. Алгоритм дозволяє збудувати зв'язки, ланцюжки виходу з вхідного шару до вихідного через приховані шари. Таким чином, ми отримуємо готову структуру логічної нейронної мережі. Наступним кроком навчання є приведення нейронної мережі після трасування, тобто корекція вагів і порогів для досягнення необхідних результатів. Також, для отримання точніших результатів, можна використати коефіцієнт приведення, який розраховуватиметься для кожного нейрона. Коефіцієнт «K» обчислюватиметься за наступною формулою:

$$Kj = \frac{U_j}{V_j},$$

де в якості значення U_j буде вибрано, наприклад, максимальне значення збудження нейрона, або що деяка, що задовольняє завданню оцінка.

У результаті, отримана логічна нейронна мережа здатна на основі формалізованих вхідних даних, видати значення шляхом логічних операцій. Таким чином, логічна нейронна мережа, по суті, є автоматизованою системою логічного виведення. Сферою застосування таких мереж є системи управління, банківська сфера, різні системи аналізу, а також система автоматичного доказу теорем, що ґрунтується на методі логічного виведення.

1.3 Логічне виведення

1.3.1 Принцип резолюції Робінсона

Математична логіка, як наука, містить в собі безліч підрозділів. Один з таких підрозділів, що називається теорією доказів, розглядає можливість

представлення доказів, як формальних математичних об'єктів, шляхом їх аналізу математичними методами. Теорія доказів є синтаксичним методом, і отримала свій розвиток завдяки роботам багатьох учених, таких як Г. Фреге, Дж. Пеано. Проте вважається, що сучасніша теорія доказів розпочинається з робіт Д. Гільберта, К. Геделя і Яна Лукашевича. У результаті теорія доказів була зведена до методів дедукції і природного виведення. Так в 1930 році в докторській дисертації Жака Ербрана уперше з'явилося поняття правила резолюцій - основного механізму для логічного виведення.

Формулювання правила було наступним:

Маючи логічну формулу A, з якої виводиться логічна формула B, то доказова операція імплікації (логічного слідування) A=>B.

Це правило дозволяло аналітичним шляхом встановити існування виведень і доказів, не використовую їх побудови.

Подальшу розробку цього правила, використовувану по сьогоднішній день, провів Джон Алан Робінсон. У 1965 році Робінсон опублікував роботу «машинно-орієнтована логіка, ґрунтована на принципі резолюції», в якій метод резолюції розглядався детальніше, а також був опис методів побудови логічних систем на основі цього методу. Надалі своєму розвитку, роботи Робінсона стали основою для створення логічних систем, зокрема мови логічного програмування PROLOG.

Суть методу резолюції полягала в наступному: маючи 2 вислови, що називаються резольвованими, необхідно отримати з них новий вираз, який міститиме усі літерали первинних висловів, за винятком взаємно зворотних літералів. Тобто, маючи в якості початкових даних вирази типу A1 = P v B'1 і A2 =!P v B'2, необхідно отримати резольвенту B'1 v B'2 . Знак «!» означає логічне заперечення. Правило виведення цього виразу називається правилом резолюцій. Це правило застосовується для доказу теорем. Відбувається це по наступному алгоритму:

а) Формулювання:

Нехай деяка формула H - гіпотеза теореми, є логічним наслідком деякої безлічі формул F1,.,Fk. Чи іншими словами, «якщо F1,.,Fk - істинні, то істинна H».

б) Застосування методу:

1) Створимо нову безліч формул {F1,.,Fk, !H}

2) Приводимо кожну формулу до КНФ, прибираємо знаки кон'юнкції, отримуємо безліч диз’юнкторів S.

3) Шукається виведення порожнього диз’юнкта, попарним застосуванням правила резолюцій, тобто шляхом виключення взаємно зворотних літералів.

в) Результат методу :

Якщо після застосування методу до безлічі S, був отриманий порожній диз’юнкт, то формула H є логічним слідством з безлічі формул F1,.,Fk. Інакше, якщо порожній диз’юнкт не був отриманий, то H не є слідством з формул F1,.,Fk.

В цілому, метод резолюції є методом доказу «від протилежного», тобто спочатку узявши заперечення теореми, у кінці застосування методу ми отримуємо порожній диз’юнкт, тобто протиріччя, а отже, первинне заперечення було помилковим, звідки слідує істинність теореми.

На основі методу резолюцій базується величезна безліч логічних систем і засобів логічного виведення. У логічному програмуванні цей метод застосовується в сукупності з одним з методів міркування - прямим або зворотним, правилами «modus ponens» і «modus tollens».

Основним недоліком методу резолюції є породження великої кількості нових резольвент, які частенько виявляються зайвими, такими, що не призводять до шуканого результату.

1.3.2 Паралельний метод резолюції

Незважаючи на універсальність, значущість і застосовність методу резолюції, сформульованого Робінсоном, він був недостатньо ефективний в

плані тимчасових витрат. Повний перебір усієї безлічі диз’юнктів і породжуваних додатково резольвент, призводили до того, що навіть на максимально потужних ЕОМ того часу, виконання алгоритму відбувалося занадто довго.

З кінця 1970-х - початку 1980-х років, ідеологія у сфері комп'ютерної індустрії спрямувалася в нове русло - створення потужного суперкомп'ютера з функціями штучного інтелекту. Цей проект отримав найбільший пріоритет в Японії, оскільки тоді була складена урядова програма, тобто ці розробки були стратегією розвитку технологій для усієї країни. Оскільки суперкомп'ютер повинен був мати штучний розум, в нього необхідно було закласти основи роботи з логікою, методики обробки даних і логічного виведення. Основою для логічного виведення став метод резолюції, проте технічні можливості дещо випереджали сам метод, тобто потрібна була модифікація методу для досягнення максимальної ефективності. Головне завдання було - зробити паралельні алгоритми, що підходять під структуру комп'ютера з паралельними процесорами.

Для реалізації паралельної системи логічного виведення, Ехуд Шапиро в другій половині 1980-х років розробив новий вид мови логічного програмування PROLOG, названий «Flat Concurrent PROLOG», в якому замість послідовної обробки логічної інформації, використовувався паралельний підхід. Розпаралелювання алгоритмів логічного виведення дозволило добитися істотного скорочення тимчасових витрат. Якщо колишні, послідовні версії, для функції часу від кількості даних видавали значення, пропорційні O(log n), то нововведені паралельні методи дозволили добитися залежності виду O(n/(log n) 2), що означало істотний прорив в цій технології.

Як мова програмування, паралельна реалізація Прологу, була мовою високого рівня, що забезпечує паралельну роботу з оброблюваними даними. У основі логічного виведення лежав модернізований, розпаралелюючий метод резолюції, зроблений Ехудом Шапиро. Ідея методу була проста - замість послідовного перебору усіх резольвованих диз’юнктів і резольвент,

потрібна була паралельна обробка відразу усієї великої кількості. Завдяки динамічному паралелізму процесів, породження нових резольвент не було проблемою для цієї реалізації методу.

Алгоритм у своїй суті не відрізнявся від послідовного. Різниця була в паралельному виборі і обробці диз'юнктів. Розглянемо сам паралельний алгоритм:

а) Формулювання залишилося таким же, як і при послідовному випадку, тобто «якщо формули F1,.,Fk - істинні, то істинна гіпотеза H».

б) Застосування методу:

1) Також відбувалося об'єднання безлічі формул {F1,.,Fk, !H}

2) Приводимо кожну формулу до КНФ, прибираємо знаки кон'юнкції, отримуємо безліч диз'юнктів S.

3) Шукається виведення порожнього диз'юнкта, попарним застосуванням правила резолюцій. Кожна пара обробляється паралельно таким чином: для усіх індексів і і j, де i, jє[1, n], і ≠j, створювалися пари(Fi,Fj), до яких застосовувалося правило резолюції.

4) Після виконання одного паралельного кроку, з'являлися нові резольвенти і додавалися до безлічі S, після чого для них також вибудовувалися пари, і застосовувалося правило резолюції.

в) Результат методу :

Результат застосування алгоритму залишився таким же, як і раніше, тобто залежно від отримання порожнього диз'юнкта, судилося про істинність або хибність поставленої теореми.

1.4 GPU загального призначення

GPGPU (**англ. *General — purpose graphics processing units*** «GPU загального призначення») — техніка використання апаратної платформи Nvidia для загальних обчислень, які зазвичай проводить центральний процесор, тобто це набір апаратних і програмних технологій, що дозволяють

використати графічні процесори, для прискорення не графічних програм [1]. Graphics Processing Units (GPUs) є високоефективними багатоядерними процесорами, здатними до складних обчислень і дуже високої пропускної спроможності даних.

GPGPU є результатом розвитку шейдерних програм і спеціалізованих на них мов програмування високого рівня, таких як, Cg, GLSL і HLSL.

Первинне призначення графічних процесорів спрямоване на рішення вузького круга завдань, що полягає в обробці графічних даних. Виходячи з цього, архітектура центрального і графічного процесорів істотно розрізняються (будова CPU і GPU див. рис. 1.4). Наприклад, основою відеочіпів NVIDIA є вісімдесяти ядерний мультипроцесор, що має в розпорядженні декілька тисяч регістрів.

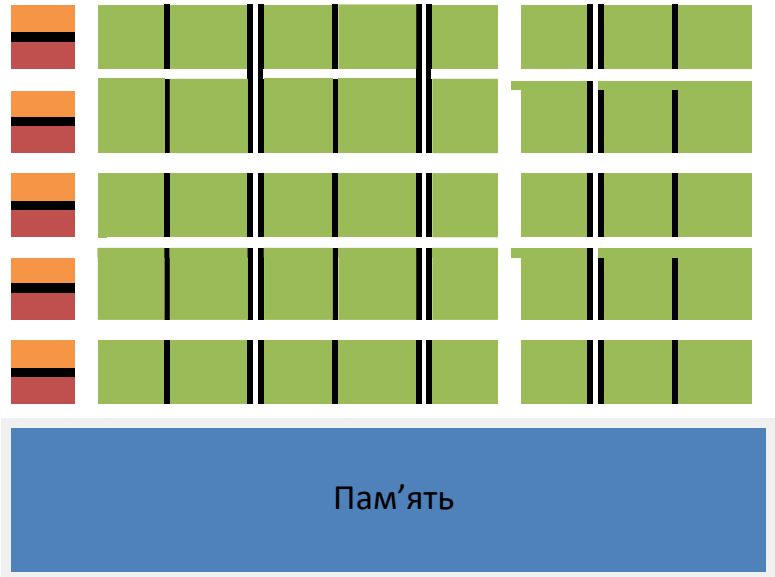


Рисунок 1.5 – Будова GPU

Графічний процесор підтримує SIMD (Single Instruction stream Multiple Data stream) метод обчислень, що означає, що ядра GPU виконують один і той же набір інструкцій для кожного екземпляра даних. Більшість графічних алгоритмів використовують саме такий підхід, як найбільш ефективний засіб для завдань графічної візуалізації. Такий модуль, що перетворює потік

вхідних даних у вихідні, називається kernel –ядро (див. рис. 1.5.) [2].

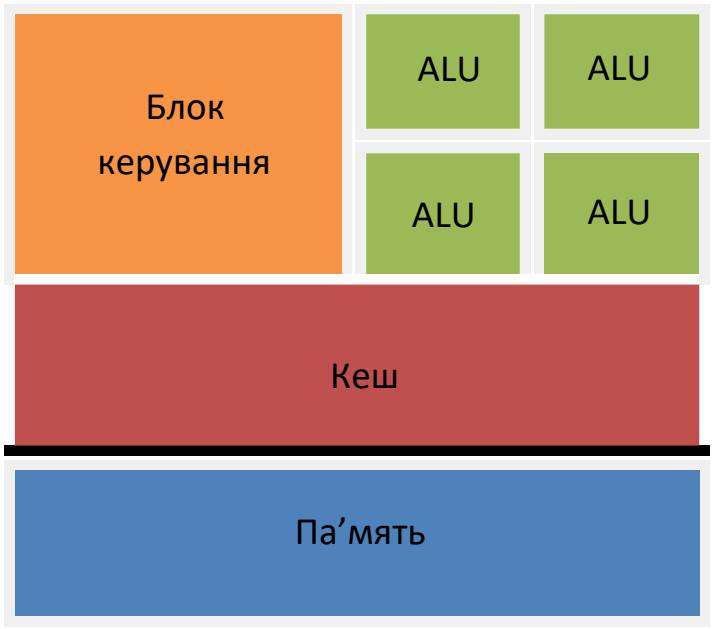


Рисунок 1.6 – Будова CPU

Для узагальнення основних відмінностей між архітектурою центрального і графічного процесорів варто відмітити, що основним завданням CPU є послідовне виконання одного потоку інструкцій з максимальною продуктивністю, тоді як конструкція GPU припускає виконання найбільшої кількості паралельних потоків одночасно.

З метою досягнення максимальної продуктивності центрального процесора розробники підвищують число завдань, що виконуються за один такт. Проте, потік виконуваних CPU команд незмінно є послідовним, що виключає всяку можливість при поточній архітектурі збільшити швидкість виконання алгоритмів пропорційним збільшенням кількості ядер.

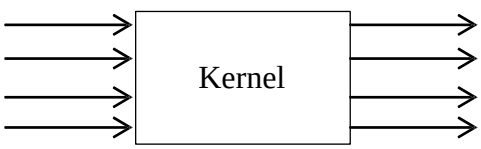


Рисунок 1.7 - Робота SIMD- архітектури

Графічний процесор, будучи допоміжним обчислювальним пристроєм, займається обробкою графічних примітивів, таким чином, що на кожному етапі графічного конвеєра дані один від одного не залежать і можуть оброблятися паралельно. Внаслідок подібної організації обчислень, графічний процесор використовує безліч виконавчих блоків, які на відміну від послідовного потоку інструкцій для центрального процесора, легко завантажити.

Принцип організації доступу до пам'яті у центрального і графічного процесорів теж істотно розрізняється. На відміну від CPU, де структура орієнтована на алгоритми з випадковим доступом до пам'яті, GPU здійснює доступ послідовно, що означає для кожного моменту часу якщо було звернення до елементи пам'яті, далі будуть задіяні дані що йдуть услід. Закономірно, що запис даних здійснюється за тим же принципом. Окрім іншого, колосальний об'єм даних характеризує більшість завдань, що вирішуються на графічному процесорі. Проблема затримки доступу до даних на CPU вирішується засобами технології кешування і пророцтва галуження коду. Рішення тієї ж задачі засобами GPU виглядає абсолютно інакше - якщо один з паралельно виконуваних процесів призупинив виконання в очікуванні доступу до пам'яті, відеочіп перемикається на інший процес, що вже має усі необхідні для подальшого виконання дані. Пропускна спроможність пам'яті апаратної платформи Nvidia, до речі, має істотно велику пропускну спроможність, ніж оперативна. Механізм кешування для зменшення затримки доступу до пам'яті використовується і в GPU. Проте якщо в центральному процесорі кеш займає переконливу долю чіпа, то GPU відводить під потреби кеша всього 128-256 кілобайт, що у свою чергу робиться швидше для підвищення пропускної можливості.

Багатопоточність в графічних процесорах також реалізована і на апаратному рівні. На CPU же її задіяти не доцільно, оскільки кожне перемикання між потоками неминуче веде до значних тимчасових затримок тривалістю в декілька сотень тактів. На додаток, ядро CPU здатне виконувати

всього декілька одночасних потоків (1-2). GPU, у свою чергу, здатний миттєво перемикатися між потоками, а кожному ядру під силу обробка до 1024 потоків.

CPU, будучи універсальним обчислювальним пристроєм, ефективно справляється з цілим спектром різних завдань, тоді як призначення графічних процесорів набагато більше вузьконаправлене. У завданнях з множинними галуженнями і переходами графічний процесор не такий ефективний як центральний.

Тому слід пам'ятати, що технологія GPGPU актуальна при обробці великих об'ємів даних. На ранній стадії розвитку цієї технології, продуктивність GPU перевершувала в 10 разів CPU. Виробники побачили велику перспективу в GPGPU, тому нестримно стали розвиватися в цьому напрямі (див. рис. 1.7.)

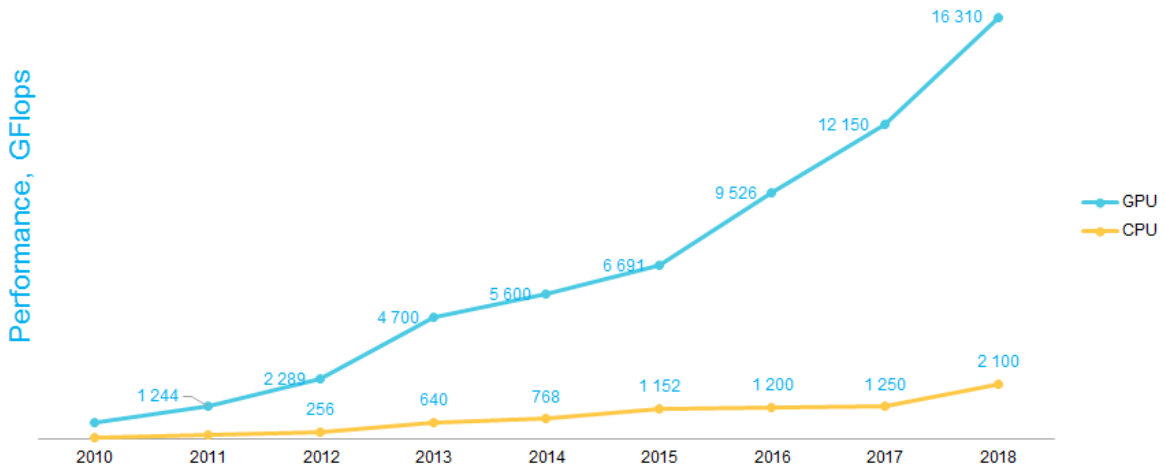


Рисунок 1.7 - Динаміка зростання продуктивності GPU і CPU

На сьогодні графічні прискорювачі працюють в 8 разу швидше. Жоден призначений для користувача CPU не може порівнятися з GPU, враховуючи, що продуктивність процесорів Intel не виходить за 3000 GFlops.

Тому використання технології GPGPU як ніколи актуально для

розробників. У зв'язку з цим, виробники почали активно розвиватися в області неграфічних обчислень. Таким чином, результатом роботи компанії nVidia, показники продуктивності якої найбільш високі, стала NVIDIA CUDA.

Кожен з виробників графічних прискорювачів представив свої засоби: NVIDIA - CUDA, AMD/ATI - AMD APP(раніше AMD Stream). Так само не так давно була представлена відкрита мова обчислень - OpenCL-фреймворк для написання комп'ютерних програм, пов'язаних з паралельними обчисленнями на різних графічних (GPU) і центральних процесорах (CPU) [3]. Важливе те, що OpenCL є повністю відкритим стандартом, його використання не обкладається ліцензійними відрахуваннями. с розробляється і підтримується некомерційним консорціумом Khronos Group, в який входять багато великих компаній, включаючи Apple, AMD, Intel, nVidia, Sun Microsystems, Sony Computer Entertainment і інші.

Але OpenCL має ряд істотних недоліків для виконання поставленої мети. По-перше, для використання необхідно зв'язатися з розробниками. По-друге, OpenCL має ряд обмежень, наприклад, відсутність підтримки покажчиків на функції, рекурсії, бітових полів та ін. На сьогодні OpenCL за оцінками фахівців nVidia є недопрацьованим. Але виробники активно розвиваються в загальній стандартизації GPGPU. Поза сумнівом, OpenCL полегшить програмістам роботу, і дасть новий поштовх розвитку обчислень на GPU.

1.5 NVIDIA CUDA

CUDA (англ. Compute Unified Device Architecture) — програмно-апаратна архітектура, що дозволяє проводити обчислення з використанням графічних процесорів NVIDIA [2], що підтримують технологію GPGPU.

У відмінності від попередніх поколінь GPU, в яких обчислювальні

ресурси підрозділялися на вершинні і піксельні шейдери, в архітектуру CUDA включені уніфікований шейдерний конвеєр, що дозволяє програмі, що виконує обчислення загального призначення, задіяти будь-який арифметично-логічні пристрої, що входить в мікросхему. Виконуючою будовою GPU дозволений довільний доступ до пам'яті для читання і запису, а так само доступ до програмно-керуючому кешу, що дістав назву пам'ять, що розділялася. Усі ці засоби додані в архітектуру CUDA з метою створити GPU, який відмінно справляється з обчисленнями загального призначення, а не тільки з традиційними завданнями комп'ютерної графіки [2]

Щоб охопити максимальну кількість розробників, NVIDIA узяла стандартну мову CI і доповнила її декількома новими ключовими словами, що дозволяють задіяти спеціальні засоби, властиві архітектурі CUDA. Через декілька місяців після випуску GeForce 8800GTX відкрила доступ до компілятора нової мови CUDAC. Він став першою мовою, розробленою компанією по виробництву GPU з метою спростити програмування GPU для обчислень загального призначення.

Окрім створення мови для програмування GPU, NVIDIA пропонує спеціалізований драйвер, що дозволяє використати можливості масивно-паралельних обчислень в архітектурі CUDA. Тепер користувачам немає нужди вивчати програмні інтерфейси OpenGL або DirectX або представляти свої завдання у вигляді завдань комп'ютерної графіки.

1.6 AMD ATI Stream Technology

ATI Stream Technology (раніше як ATI FireStream і AMD StreamProcessor) — це набір апаратних і програмних технологій, які дозволяють використати графічні процесори AMD, спільно з центральним процесором, для прискорення багатьох застосувань (не лише графічних) [4]

1.7 Переваги CUDA

Основними плюсами CUDA є її безкоштовність(SDK для усіх основних платформ вільно викачується з developer.nvidia.com), простота (програмування ведеться на "розширеному С") і гнучкість. Технологія NVIDIACUDA - це єдине середовище розробки на мові програмування С, яка дозволяє розробникам створювати програмне забезпечення для вирішення складних обчислювальних завдань за менший час, завдяки багатоядерній обчислювальній потужності графічних процесорів.

Хоча фахівці сперечаються, вибираючи між CUDA і OpenCl. Але поки що основною проблемою реалізації OpenCL від NVidia є низька продуктивність в порівнянні з CUDA, але з кожним новим випуском драйверів продуктивність OpenCL під управлінням CUDA все ближче підбирається до продуктивності CUDA застосунків.

Вивчивши ринок, можна сказати, що NVIDIACUDA найбільш поширена. За цією технологією є більша кількість учбових матеріалів, чим для інших аналогічних.

NVIDIA веде активну підтримку розробників, також на офіційному сайті виділений спеціальний розділ CUDAZONE для спілкування програмістів, обміну матеріалами, перекладів статей і описів технології, можливості поставити питання творцям CUDA.

1.8 Оцінка наукового, технічного та економічного рівня НДДКР

Проаналізуємо рівень науково-дослідної роботи яка пов’язана з дослідженням автономного пристрою на Nvidia для задач штучного інтелекту. Виходячи з відповідних вимог НТП, доцільно орієнтуватися на час проведення НДДКР 2 роки і менше (+2), при чому технічні показники результатів плануються на на рівні кращих світових зразків (0); наявність

можливості отримання авторських свідоцтв на винахід - часткові можливості (0); а строк окупності витрат 2 роки і менше (+3).

В таблиці 1.1 наведено критерії та бальна оцінка для визначення наукового та технічного рівня науково-дослідної роботи

Таблиця 1.1 – Критерії та бальна оцінка для визначення наукового, технічного та економічного рівня науково-дослідної роботи.

Критерії оцінки	Шкала критеріїв	Індекс оцінки
Час, необхідний для проведення НДР	2 роки і менше 3 роки 4 роки 5-6 років 7 років і більше	+2 +1 0 -1 -2
Технічні показники результатів розробки	Вище рівня кращих світових зразків На рівні кращих світових зразків Нижче рівня кращих світових зразків	+2 0 -2
Можливості отримання авторських свідоцтв на винахід	Впевненість в отриманні авторських свідоцтв Часткові можливості Можливості немає	+2 0 -1
Строк окупності витрат	2 роки і менше 3-4 роки 5 років 6-7 років 8 років і більше	+3 +2 0 -1 -2

Проаналізувавши дані таблиць 1.1 та 1.2, та підрахувавши загальну суму балів (+2+0+0+3=+5), робимо висновок, що дана науково-дослідна робота з

дослідження автономного пристрою на Nvidia для задач штучного інтелекту є досить перспективною.

В таблиці 1.2 наведено можливі результати оцінки теми НДДКР.

Таблиця 1.2 – Можливі результати оцінки теми НДДКР

Сума індексів	Оцінка теми
Позитивна(+)	Розробка є досить перспективною
Задовільна(0)	Розробка перспективна
Негативна(-)	Розробка не перспективна

1.9 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу результатів НДДКР. В результаті оцінювання можна зробити висновок щодо напрямів (особливостей) організації подальшого впровадження результатів з врахуванням встановленого рейтингу.
Рекомендується здійснювати оцінювання комерційного потенціалу розробки за 12-ма критеріями, наведеними в таблиці 1.3. **[10]**

Таблиця 1.3 - Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздат-ність продукту в реальних умовах

Бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					

Бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років

Бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання комерційного потенціалу розробки зведемо до таблиці 1.4.

Таблиця 1.4 - Результати оцінювання комерційного потенціалу розробки

Критерії	експерт		
	1	2	3
	Бали, виставлені експертами:		
1. Технічна здійсненність концепції	3	3	3
2. Ринкові переваги (наявність аналогів)	3	3	3
3. Ринкові переваги (ціна продукту)	2	2	2
4. Ринкові переваги (технічні властивості)	4	5	4
5. Ринкові переваги (експлуатаційні витрати)	2	2	3
6. Ринкові перспективи (розмір ринку)	1	1	1

Критерії	експерт		
	1	2	3
	Бали, виставлені експертами:		
7. Ринкові перспективи (конкуренція)	2	2	3
8. Практична здійсненність (наявність фахівців)	3	3	2
9. Практична здійсненність (наявність фінансів)	3	3	3
10. Практична здійсненність (необхідність нових матеріалів)	2	2	2
11. Практична здійсненність (термін реалізації)	3	4	3
12. Практична здійсненність (розробка документів)	3	3	3
Сума балів	31	33	32
Середньоарифметична сума балів СБ	**32,0**		

За даними таблиці 1.4 зробимо висновок щодо рівня комерційного потенціалу дослідження. При цьому доцільно користуватися рекомендаціями, наведеними в таблиці 1.5. **[10]**

Таблиця 1.5 - Рівні комерційного потенціалу розробки

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

Згідно проведених досліджень рівень комерційного потенціалу розробки становить 32,0 бала, що, згідно таблиці 1.5, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

1.10 Розрахунок узагальненого коефіцієнта якості для нового рішення

В процесі дослідження необхідно розглянути основні технічні показники, пристрою, що може бути спроектований в результаті проведення дослідження автономного пристрою на Nvidia для задач штучного інтелекту. Ці показники по-різному впливають на загальну якість проектної розробки.

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення розрахуємо за формулою [11]:

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i , \qquad (1.1)$$

де k – кількість найбільш важливих технічних показників, які впливають на якість нового технічного рішення;

α_i – коефіцієнт, який враховує питому вагу *і-го* технічного показника в загальній якості розробки. Коефіцієнт α_i визначається експертним шляхом і при цьому має виконуватись умова $\sum_{i=1}^{k} \alpha_i = 1$;

β_i – відносне значення *і-го* технічного показника якості нової розробки.

Відносні значення β_i для різних випадків розраховують за такими формулами:

- для показників, зростання яких вказує на підвищення в лінійній залежності якості нової розробки:

$$\beta_i = \frac{I_{нi}}{I_{ai}} , \qquad (1.2)$$

де $I_{ні}$ та $I_{на}$ – чисельні значення конкретного i-го технічного показника якості відповідно для нової розробки та аналога;

- для показників, зростання яких вказує на погіршення в лінійній залежності якості нової розробки:

$$\beta_i = \frac{I_{ai}}{I_{ні}}; \tag{1.3}$$

Використовуючи наведені залежності можемо проаналізувати та порівняти техніко-економічні характеристики аналогу та майбутньої розробки на основі отриманих наявних та проектних показників, а результати порівняння зведемо до таблиці 1.6.

Таблиця 1.6 – Порівняння основних параметрів пристрою що проектується та аналога.

Показники (параметри)	Одиниця вимірю-вання	Аналог	Проектований пристрій	Відношення параметрів нової розробки до аналога	Питома вага показника
Графічний процесор, ядра NVIDIA CUDA	шт	256	128	0,5	0,25
Процесор, ядра ARM® Cortex	шт	2	4	2	0,15
Кодування відео 1x 4K @ (HEVC)	шт	30	30	1	0,2
Декодування відео 1x 4K @ (HEVC)	шт	30	60	2	0,15
Підключення	Гігабіт	1	10	10	0,25

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення складе:

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i = 0,5*0,25+2*0,15+1*0,2+2*0,15+10*0,25 = 3,43.$$

Отже за технічними параметрами, згідно узагальненого коефіцієнту якості розробки, проектований компонент переважає існуючі аналоги приблизно в 3,43 рази.

1.11 Визначення рівня конкурентоспроможності розробки

Проектування виробу розпочинається з прогнозу його конкурентоспроможності. В процесі попереднього оцінювання доцільності детального проектування нової розробки здійснимо оцінювання рівня конкурентоспроможності, яке проведемо визначенням сукупності параметрів, що підлягають оцінюванню.

Загальні технічні та економічні характеристики пристрою представлено в таблиці 1.6.

Таблиця 1.6 – Основні техніко-економічні показники аналога та пристрою, що проектується

Показники (параметри)	Одиниця вимірю-вання	Аналог	Проектований пристрій	Відношення параметрів нової розробки до аналога	Питома вага показника
Графічний процесор, ядра NVIDIA CUDA	шт	256	128	0,5	0,25
Процесор, ядра ARM® Cortex	шт	2	4	2	0,15

Показники (параметри)	Одиниця вимірю-вання	Аналог	Проектований пристрій	Відношення параметрів нової розробки до аналога	Питома вага показника
Кодування відео 1x 4K @ (HEVC)	шт	30	30	1	0,2
Декодування відео 1x 4K @ (HEVC)	шт	30	60	2	0,15
Підключення	Гігабіт	1	10	10	0,25
Ціна пристрою	грн	4600	2800	0,61	1

Одиничний параметричний індекс розраховується за формулою [12]:

$$q_i = \frac{P_i}{P_{базі}}. \qquad (1.4)$$

де q_i – одиничний параметричний індекс, розрахований за i-м параметром;

P_i – значення i-го параметра виробу;

$P_{базі}$ – аналогічний параметр базового виробу-аналога, з яким проводиться порівняння.

Нормативні параметри оцінюються показником, який отримує одне з двох значень: 1 – пристрій відповідає нормам і стандартам; 0 – не відповідає.

Груповий показник конкурентоспроможності за нормативними параметрами розраховується як добуток частинних показників за кожним параметром за формулою [12]:

$$I_{НП} = \prod_{i=1}^{n} q_i \ , \qquad (1.5)$$

де $I_{нп}$ – загальний показник конкурентоспроможності за нормативними параметрами;

q_i – одиничний (частинний) показник за i-м нормативним параметром;

n – кількість нормативних параметрів, які підлягають оцінюванню.

За нормативними параметрами розроблюваний пристрій відповідає вимогам ДСТУ, тому $I_{нп} = 1$.

Значення групового параметричного індексу за технічними параметрами визначається з урахуванням вагомості (частки) кожного параметра [12]:

$$I_{ТП} = \sum_{i=1}^{n} q_i \cdot \alpha_i \ , \qquad (1.6)$$

де $I_{тп}$ – груповий параметричний індекс за технічними показниками (порівняно з виробом-аналогом);

q_i – одиничний параметричний показник i-го параметра;

α_i – вагомість i-го параметричного показника, $\sum_{i=1}^{n} \alpha_i = 1$;

n – кількість технічних параметрів, за якими оцінюється конкурентоспроможність.

Проведемо аналіз параметрів згідно даних таблиці 1.6.

$$I_{тп} = 0,5*0,25+2*0,15+1*0,2+2*0,15+10*0,25 = 3,43.$$

Груповий параметричний індекс за економічними параметрами розраховується за формулою [12]:

$$I_{ЕП} = \sum_{i=1}^{m} q_i \cdot \beta_i \ , \qquad (1.7)$$

де $I_{ЕП}$ – груповий параметричний індекс за економічними показниками;

q_i – економічний параметр i-го виду;

β_i – частка i-го економічного параметра, $\sum_{i=1}^{m} \beta_i = 1$;

m – кількість економічних параметрів, за якими здійснюється оцінювання.

Проведемо аналіз параметрів згідно даних таблиці .

$$I_{ЕП} = 0,61 * 1 = 0,61.$$

На основі групових параметричних індексів за нормативними, технічними та економічними показниками розрахуємо інтегральний показник конкурентоспроможності за формулою [12]:

$$K_{ІНТ} = I_{НП} \cdot \frac{I_{ТП}}{I_{ЕП}}, \qquad (1.8)$$

$$K_{ІНТ} = 1 * 3,43 / 0,61 = 5,61.$$

Інтегральний показник конкурентоспроможності $К_{ІНТ} > 1$, отже проектований пристрій переважає аналог за своїми техніко-економічними показниками.

1.12 Висновки до першого розділу

Згідно проведених досліджень рівень комерційного потенціалу розробки становить 32,0 балів, що свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки

вище середнього). При оцінюванні за технічними параметрами, згідно узагальненого коефіцієнту якості розробки, удосконалений пристрій переважає існуючі аналоги приблизно в 3,43 рази. Отже можна зробити висновок про доцільність проведення НДДКР з дослідження та розробки автономного пристрою на Nvidia для задач штучного інтелекту.

Таблиця 1.7 – Основні техніко-економічні показники аналога і нової розробки

Показники	Одиниця виміру	Аналог	Засіб, що проектується	Відношення параметрів
Графічний процесор, ядра NVIDIA CUDA	шт	256	128	0,5
Процесор, ядра ARM® Cortex	шт	2	4	2
Кодування відео 1x 4K @ (HEVC)	шт	30	30	1
Декодування відео 1x 4K @ (HEVC)	шт	30	60	2
Підключення	Гігабіт	1	10	10
Ціна пристрою	грн	4600	2800	0,61

Нейронні мережі наведені в додатку Б 08-36.МКР.003.00.000 ПЛ1.

2 АРХІТЕКТУРА СИСТЕМИ, ЩО РОЗРОБЛЯЄТЬСЯ

2.1 Застосування технології CUDA для паралельної реалізації логічної нейронної мережі

Технології обробки інформації постійно розвиваються, для досягнення більшої ефективності на новій архітектурі. Було запропоновано використати процесор графічної карти для паралелізування алгоритму обчислень, шляхом передачі оброблюваних даних з центрального процесора на процесор апаратної платформи Nvidia. через 3 роки, одна з передових компаній, що виробляють графічні карти утілило нову архітектуру побудови процесора графічної карти, що дозволяє задіяти її в якості співпроцесора. Технологія була названа CUDA, що було абревіатурою від «Compute Unified Device Architecture».

Технологія CUDA є не лише новою архітектурою графічної карти. У це поняття також входить набір програм і надбудов, що дозволяють створювати свої застосування на різних мовах програмування, використовуючи паралелізм графічної карти. Мовами, що підтримують цю технологію являються Fortran і C/C++. Для того, щоб почати роботу з цією технологією, необхідно викачати з офіційного сайту компанії Nvidia відповідний пакет програм, названий CUDA Toolkit, і встановити його.

У основі інтерфейсу створення застосувань CUDA лежить мова C, з деякими невеликими змінами, а також безліч бібліотек, що підключаються, для використовуваного середовища програмування. Для реалізації застосувань, використовуючих CUDA, створений компілятор на мові « Сі», названий nvcc, що дозволяє обробити код програми і розподілити його залежно від використовуваної технології.

Архітектура CUDA використовує модель пам'яті «Grid→Block→Thread», що дозволяє незалежний паралельний доступ до пам'яті апаратної платформи. Ця структура зображена на рис. 2.1.

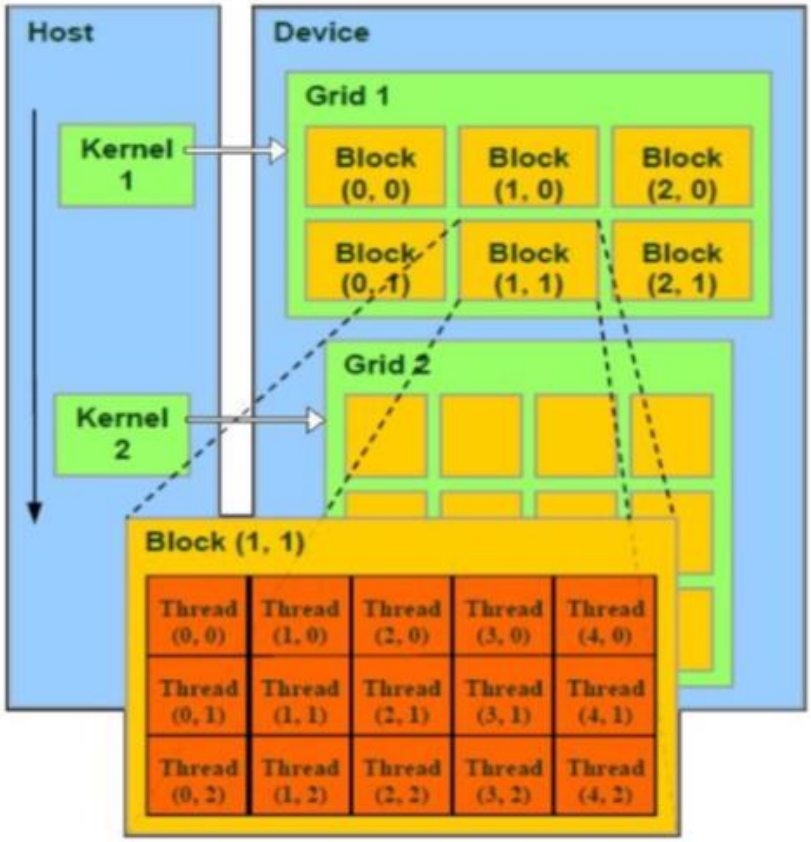


Рисунок 2.1 – Структура пам'яті в апаратній платформі Nvidia

На цій ілюстрації є присутніми наступні поняття:

1) Host – центральний процесор

2) Kernel – ядро ЦП

3) Device – апаратна платформа Nvidia

4) Grid – розмічена сітка, масив з блоків пам'яті.

5) Block – блок пам'яті, масив, що об'єднує деяку кількість ниток

6) Thread – нитка, покажчик на конкретне місце в пам'яті.

Кожному блоку відповідає одне ядро апаратної платформи Nvidia, що дозволяє добитися високого паралелізму при використанні обчислень на графічній карті. У сучасних апаратних платформах Nvidia Tesla кількість таких ядер налічує близько п'яти тисяч, що означає можливість використання до 5 тисяч паралельних операцій. Завдяки бар'єрній пам'яті, що синхронізується, не існує проблем з використанням однієї ділянки різними

процесами, а отже дозволяє уникнути безлічі помилок. В цілому, систему пам'яті в CUDA можна зображувати, як на рисунку 2.2

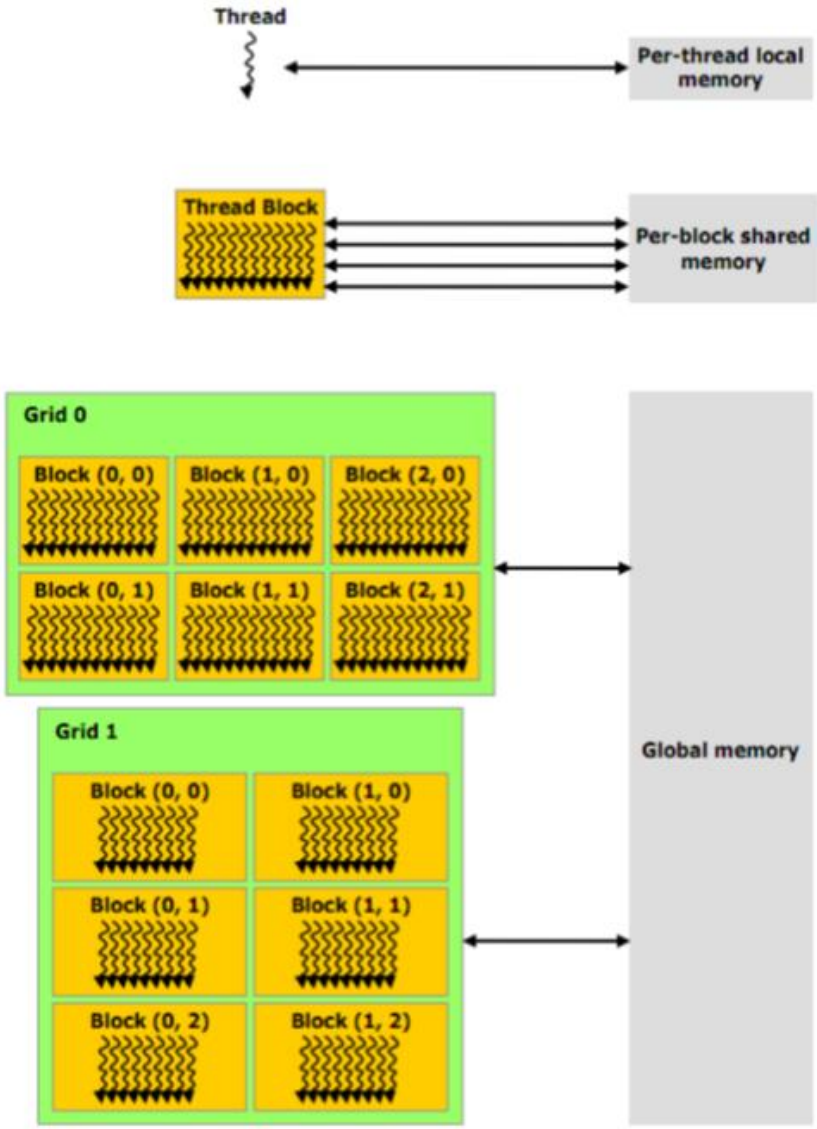


Рисунок 2.2 – Організація пам'яті в CUDA

Оскільки у більшості сфер не потрібно застосовувати складні, витратні з точки зору продуктивності ЕОМ, операції, то вигідніше провести виконання таких простих процесів не на центральному процесорі, тим самим звільнивши його, і надавши можливість обробки складніших операцій. Таким чином, вигідно використати апаратні платформи Nvidia для подібних простих обчислень, особливо при необхідності реалізації схеми SIMD, для одночасної обробки великої кількості однотипних даних.

Окрім створення різних аналітичних систем, технологія розпаралелювання обчислень на ядра апаратної платформи Nvidia використовується для отримання надпотужних суперкомп'ютерів. Подібні розробки в Японії і Китаї дозволили добитися продуктивності в десятки гігафлопів.

Нейронні мережі, що використовують архітектуру CUDA, дозволяють проводити процес навчання і обробки інформації в десятки разів швидше, ніж при послідовних операціях на центральному процесорі. Наприклад, для систем аналізу зображення, замість послідовної обробки кожного пікселя, можливий аналіз усього зображення, або його великих фрагментів.

Для реалізації логічної нейронної мережі також можливе застосування паралельної архітектури CUDA для навчання, і для подальшої роботи. Якщо розглядати реалізацію логічної нейронної мережі, як засоби логічного виведення і автоматичного доказу теорем, то для застосування паралельного методу резолюції Шапіро, потрібна паралельна структура, яка дозволить одночасно обробляти усю вхідну безліч диз’юнктів. Таким чином технологія CUDA за всіма характеристиками підходить для цієї реалізації.

2.2 Архітектура NVIDIA CUDA

Основне призначення — дати програмісту можливість використати GPU в якості співпроцесора для завдань, що вимагають паралельних обчислень, абстрагуючись при цьому від термінології і не використовуючи бібліотеки специфічні для обробки 3D — графіки. Проте, не все так просто, тому що платформи спочатку орієнтовані і багато років розвивалися як пристрої для обробки графіки. І саме особливості архітектури GPU викликають найбільші складнощі при першому знайомстві з платформою.

До складу платформи, окрім самого процесора входить, так званий драйвер CUDA, CUDA Toolkit і CUDA SDK. CUDA Toolkit включає

бібліотеки, необхідні для роботи з платформою, і компілятор nvcc, що транслює початковий код програм в проміжний асемблерний код, а так само додаткові бібліотеки. Драйвер CUDA в ранніх версіях включав тільки компілятор, що здійснює перетворення проміжного асемблера, генерованого компілятором nvcc в мікрокод, що виконується на GPU. Зараз драйвер CUDA і драйвер пристрою об'єднані в один пакет. CUDA SDK містить набір початкових кодів простих програм, що ілюструють методи і можливості роботи з платформою CUDA.

Особливо варто звернути увагу на той факт, що CUDA підтримується лише процесорами відеоприскорювачів GeForce восьмого покоління і старше, а так само Quadro і Tesla.

Перерахуємо основні характеристики CUDA :

– Уніфіковане програмно-апаратне рішення для паралельних обчислень.

– Великий набір підтримуваних графічних плат (від мобільних до мультичіпових).

– Як мова програмування використовується розширений варіант мови С.

– Підтримує взаємодію з графічними API OpenGL і DirectX

– Є підтримка 32 - і 64-бітових операційних систем: Windows, Linux і MacOSX.

– Можливість розробки на низькому рівні.

– CUDA забезпечує доступ до швидкої пам'яті, що розділяється, яка може бути використана для міжпоточної взаємодії.

Незважаючи на той факт, що при програмуванні шейдерів і CUDA - програма може бути використана одна і та ж апаратна частина, ці підходи істотно відрізняються один від одного. Слабким місцем шейдерного підходу є те, що цей метод не дозволяє повною мірою завантажити GPU. Украй складно скласти алгоритм так, щоб завантажити вершинні шейдери. Використання графічного API так само змушує програміста використати

текстури для зберігання даних (іншої альтернативи просто немає), що веде до додаткових затримок, пов'язаних з необхідністю їх створення і підготовки.

CUDA- програми пишуться на діалекті мови С, реалізованого так, щоб на нім можна було писати як код, що виконується на GPU, так і код, що виконується на CPU.

Оскільки фізична архітектура GPU є комерційною таємницею nVidia, для розуміння роботи з платформою на рисунку 2.3 розглядається логічна архітектура:

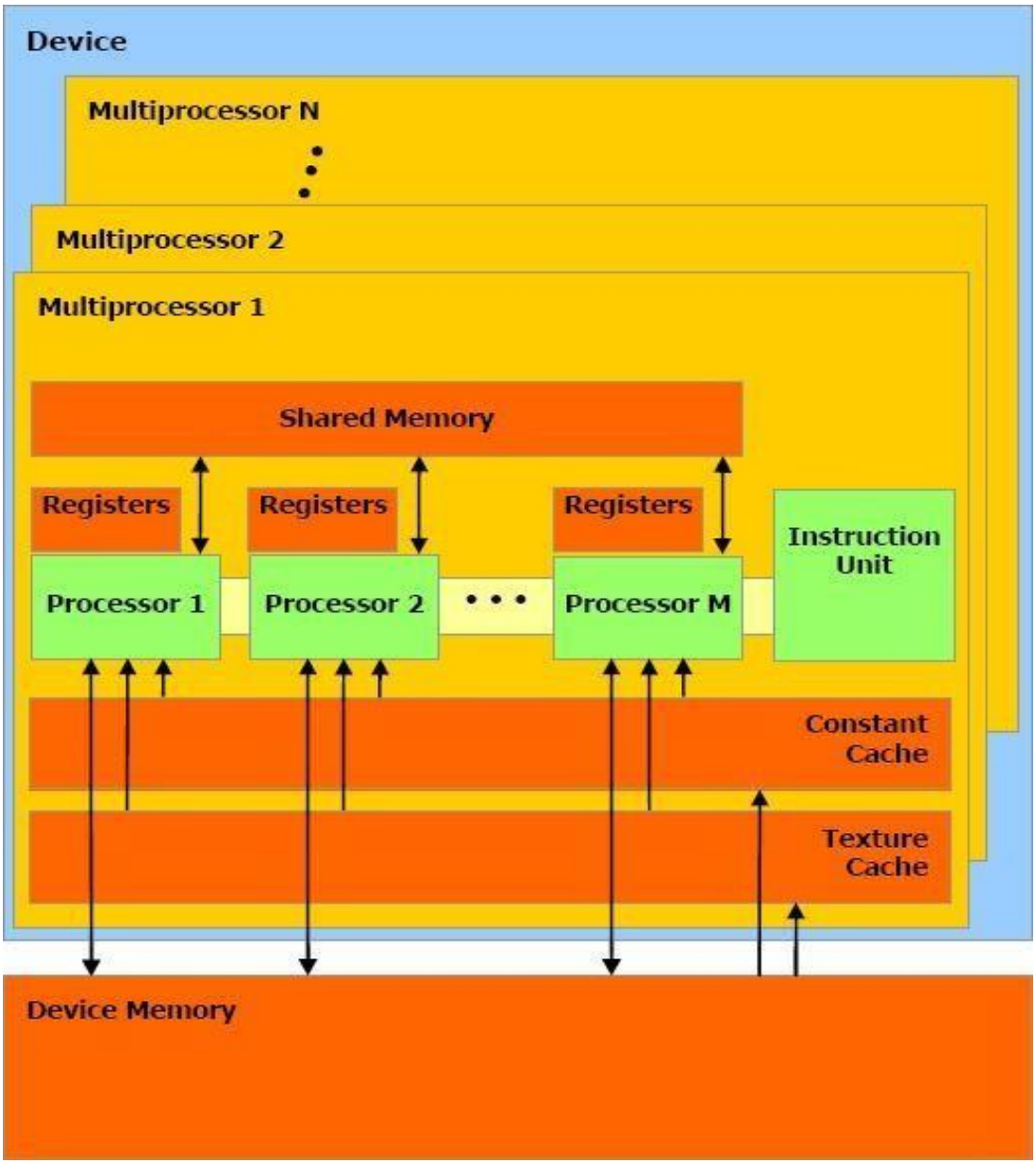


Рисунок 2.3 – Архітектура CUDA

2.3 Модель пам'яті CUDA

Мабуть, однією з найбільш важливих для розробників особливостей CUDA є вільний доступ до пам'яті(підтримка scatter - і gathe - операцій) з можливістю побайтової адресації.

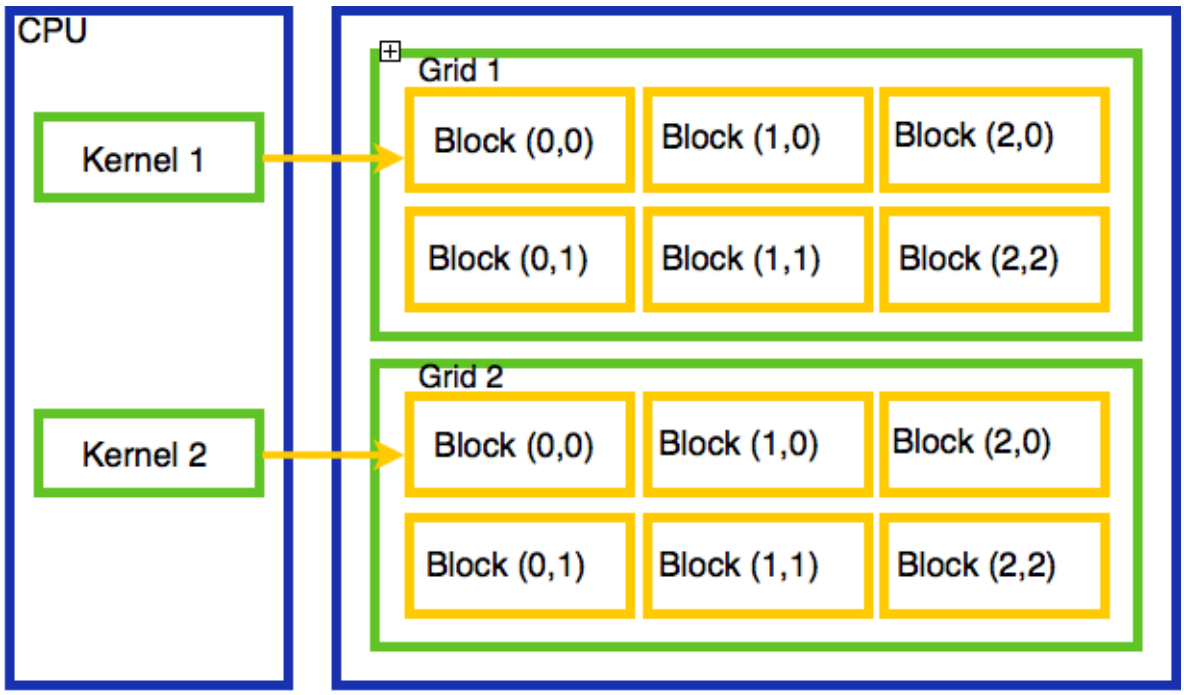


Рисунок 2.4 – Модель мультипроцесора NVIDIA

Кожен з потоків, виконуваний на GPU має доступ до наступних типів пам'яті :

Глобальна пам'ять - основний тип пам'яті, що має найбільший об'єм і доступний для усіх мультипроцесорів на відеочіпі. Розмір глобальної пам'яті безпосередньо залежить від моделі і варіюється від 256 мегабайт до 4 гігабайт на Tesla. Має високу пропускну спроможність (більше 100 гігабайт/с на останніх рішеннях від NVIDIA), але робота з цим типом пам'яті зв'язана зі значними часовими затримками (декілька сотень тактів). Глобальна пам'ять не кешується, підтримує лінійну адресацію і звичайні покажчики.

Локальна пам'ять - тип пам'яті, в якій за умовчанням розміщуються усі змінні оголошені усередині CUDA- програми. Також як і глобальна пам'ять,

вона дуже повільна і не підтримує кешування.

Пам'ять, що розділяється - це 16-кілобайтний блок пам'яті, доступний на запис і читання з усіх потокових процесорів в мультипроцесорі. За швидкістю цей тип пам'яті порівнянний з регістрами. Основне призначення пам'яті, що розділяється - забезпечення взаємодії між потоками. Пам'ять, що розділяється, також може бути використана в ролі програмованого кеша, за допомогою якого досягається зниження затримок при роботі з глобальною пам'яттю.

Константна пам'ять – область пам'яті розмірів в 64 кілобайти використовувана для зберігання константних значень програми. Цей тип пам'яті кешується (по 8 кілобайт на мультипроцесор). І за відсутності необхідних даних в кеші читання з константної пам'яті здійснюється із затримками в декілька сотень тактів.

Таблиця 2.1 - Технічні характеристики різних видів пам'яті

Назва	Розташування	Кешованість	Читання/Запис	Доступність	Тип пам'яті
Регістрова пам'ять	Чіп	Ні	Читання і Запис	Ядро	Dedicated HW
Локальна пам'ять	За межами чіпа	Ні	Читання і Запис	Ядро	DRAM
Загальна пам'ять	Чіп	Ні	Читання і Запис	Усі ядра в групі	Dedicated HW
Глобальна пам'ять	За межами чіпа	Ні	Читання і Запис	GPU і CPU	DRAM
Константна пам'ять	За межами чіпа	Так	Тільки читання	GPU і CPU	DRAM
Текстурна Пам'ять	За межами чіпа	Так	Тільки читання	GPU і CPU	DRAM

Текстурна пам'ять – блок пам'яті, доступний тільки на читання усіма мультипроцесорами. Вибірка з цього типу пам'яті відбувається за допомогою текстурних блоків відеочіпа. За рахунок цього можливе виконання лінійної інтерполяції без яких-небудь додаткових витрат. Цей тип пам'яті кешується [5].

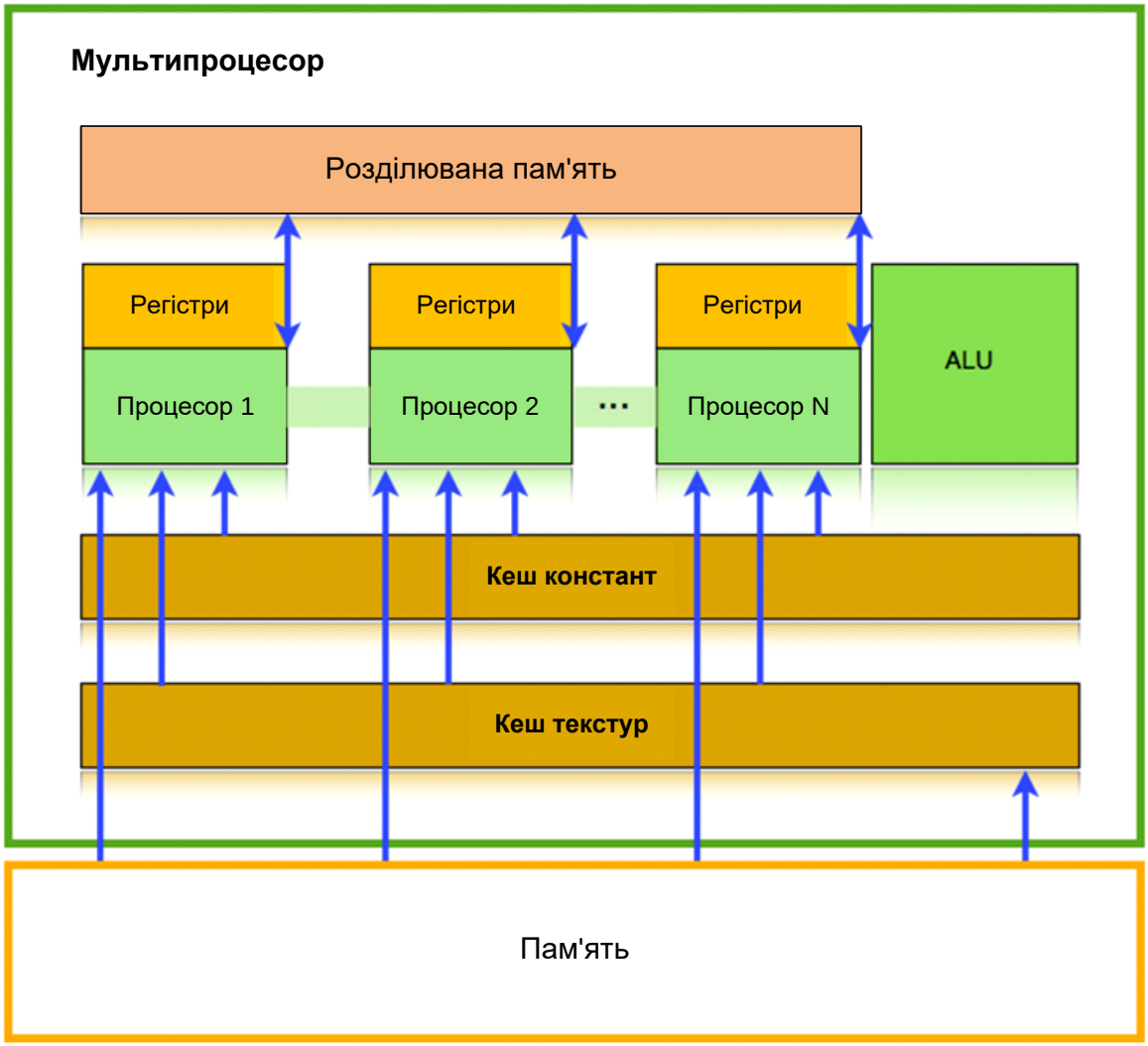


Рисунок 2.5 – Програмна модель пам'яті мультипроцесора

На рисунку 2.3 зображена програмна модель пам'яті. Насправді глобальна, локальна, текстурна і константна пам'ять представлені на чіпі у вигляді локальної пам'яті. Центральному процесору доступна лише глобальна, константна і текстурна пам'ять.

При проектуванні алгоритму для CUDA украй важливо пам'ятати про усі види пам'яті і їх особливості. Так, приміром, треба мінімізувати кількість звернень до глобальної і локальної пам'яті, оскільки вони повільні і не підтримують кешування.

2.4 Мультипроцесори

Відеочіпи від NVIDIA складаються з декількох кластерів текстурних блоків (Texture Processing Cluster, TPC). Кожен кластер у свою чергу складається з блоку текстурних вибірок і декількох мультипроцесорів. Мультипроцесор складається з 8 обчислювальних пристроїв і двох суперфункціональних блоків. Усі інструкції в GPU виконуються за принципом SIMD, тобто одна інструкція застосовується до усіх потоків в wrap (група з 32-х потоків). Цей спосіб виконання так і назвали SIMT (Single Instruction Multiple Threads) - одна інструкція і багато потоків [4].

На кожного з мультипроцесорів виділяється від 8192 до 16384 регістрів(залежно від моделі), які є загальними для усіх потоків виконуваних на нім. Також на кожному процесорі є 16 кілобайт швидкої пам'яті, що розділяється, яка доступна на запис і читання з будь-якого потоку усередині одного блоку. Мультипроцесори також мають доступ до відеопам'яті, але необхідно бути украй обережним, оскільки доступ до неї зв'язаний зі значними тимчасовими затримками. Для прискорення доступу і зниження кількості звернень до відеопам'яті усі мультипроцесори оснащуються невеликим (8 кілобайт) кешем для констант і текстур.

Мультипроцесори GPU можуть виконувати до восьми блоків і до 24-х wrap, кожен з яких включає 32 потоки. Іншими словами, на мультипроцесор доводиться до 768 потоків.

Знання архітектури GPU украй важливе при складанні алгоритму, оскільки воно дозволяє оптимізувати його під доступні ресурси.

2.5 Програмування CUDA

Як згадувалося раніше, графічний конвеєр, використовуваний для візуалізації зображення, є набором з безлічі стадій обробки. При використанні традиційних API програміст незалежно від складності алгоритму завжди зобов'язаний конфігурувати усі частини графічного конвеєра. Цей факт істотно утрудняє використання GPU для вирішення завдань загального призначення, оскільки навіть просте складання двох матриць вимагає виконання ряду команд по підготовці і відмальовці зображень позаекранному буфері. У результаті на декілька рядків шейдерної програми доводяться сотні рядків додаткового коду. При рішенні завдань з невеликою розмірністю ці додаткові витрати здатні звести нанівець увесь виграш від використання GPU.

Модель програмування, використовувана в CUDA відрізняється від традиційних API тим, що повністю приховує графічний конвеєр від програміста, дозволяючи йому тим писати програми в звичніших для нього «термінах» на розширеній варіації мови C.

Крім того, CUDA надає програмістові зручнішу модель роботи з пам'яттю. Більше немає необхідності зберігати дані в 128-бітових текстурах, оскільки CUDA дозволяє читати дані безпосередньо з пам'яті.

До складу NVIDIACUDA входять два API: високого рівня (CUDA Runtime API) і низького (CUDA Driver API) дивися рисунок 2.6. При необхідності задіяти низькорівневі функції графічного процесора програміст завжди може відмовитися від Runtime API на користь Driver API. Варто відмітити, що використання обох API в одній програмі не є можливим [6].

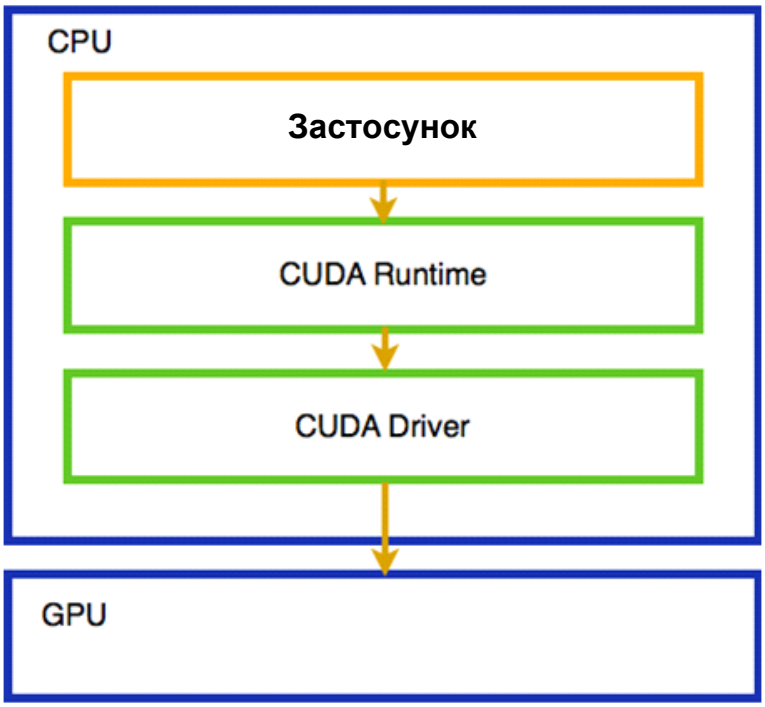


Рисунок 2.6 – Схема CUDA API

Першим кроком при перенесенні існуючого алгоритму на CUDA неодмінно являється його аналіз, мета якого полягає в пошуку «вузького місця», що потребує розпаралелювання. Як правило, в алгоритмі для CPU такі ділянки поміщені в цикл або рекурсію.

Повне перенесення алгоритму на GPU не є можливим, оскільки графічний процесор не має доступу ні до пам'яті комп'ютера, ні до облаштувань введення/виведення(за винятком буфера кадру, який може бути відображений у вигляді картинки на екрані комп'ютера). При виконанні програми CPU як і раніше відповідає за підготовку і постобробку даних, сама ж трудомістка робота лягає на GPU. Набір інструкцій, що виконується на графічному процесорі, називається ядром (kernel). Ядро, по суті, є розвитком концепції шейдерів.

За формування і компіляцію ядер відповідає CPU. Відеочіп просто приймає вже скомпільоване ядро і створює його копії для кожного елементу даних. Кожне з ядер виконується у своєму власному потоці.

Потоки в GPU можуть виконуватися лише групами по 32 екземпляри

(wrap). При цьому загальне число потоків необхідне для вирішення завдання може перевершувати максимально допустиме для поточного пристрою. Тому кожен такт апаратне забезпечення вибирає, який з wrap буде виконаний. Але якби в CPU подібне перемикання зайняло б сотні тактів, то GPU робить це майже миттєво.

На відміну від шейдерів, де усі дані представлені у вигляді чотирьох компонентних векторів, дані в ядрі скалярних. Таке представлення природніше для більшості неграфічних завдань.

Модель програмування CUDA припускає угрупування потоків блоки одно- двох- або тривимірні матриці. Взаємодія між ними здійснюється за допомогою пам'яті, що розділяється. Також існують точки синхронізації, що дозволяють привести дані в усіх потоках в актуальний стан.

Кожне з ядер виконується над сіткою (grid) блоків. У кожен момент часу на GPU може виконуватися лише одна сітка. Подібне угрупування дозволяє досягти максимальної масштабованості. Якщо у GPU недостатньо ресурсів для запуску усіх блоків - вони виконуватимуться послідовно, один за одним. Це дозволяє розробникові не замислюватися про потужність пристрою, на якому буде запущено застосування.

У кожному застосуванні, написаному на NVIDIACUDA незалежно від його призначення можна виділити ряд загальних кроків :

Підготовка пам'яті. Оскільки GPU не має доступу до оперативної пам'яті програмістові необхідно заздалегідь потурбуватися про те, що усі ресурси, необхідні для виконання ядра застосування знаходяться в пам'яті. Для цих цілей використовуються три основні функції з CUDASDK: cudaMalloc, cudaMemcpy і cudaFree. Ці функції мають те ж призначення, що і стандартні malloc, memcpy і free, але, зрозуміло, усі операції проводяться над відеопам'яттю. Так само варто відмітити, що функція cudaMemcpy має додатковий параметр, що означає напрям копіювання інформації (з CPU в GPU або навпаки).

Конфігурація сітки (grid) і блоків (blocks). Процес конфігурації украй

простий і полягає в завданні розмірів сітки і блоків. Основним завданням програміста на цьому кроці є знаходження оптимального балансу між розміром і кількістю блоків. Збільшенням кількості потоків у блоці можна понизити кількість звернень до глобальної пам'яті за рахунок збільшення інтенсивності обміну даними між потоками через швидку пам'ять, що розділяється. З іншого боку, кількість регістрів що виділяються на блок фіксовано і якщо кількість потоків виявиться сильно великою, то GPU почне розміщувати дані в повільній локальній пам'яті, що істотно збільшить час виконання ядра. NVIDIA рекомендує програмістам використати блоки по 128 або 256 потоків. У більшості завдань така кількість потоків у блоці дозволяє досягти оптимальних затримок і кількості регістрів.

Запуск ядра. Ядро викликається як звичайна функція в мові С. Єдина істотна відмінність полягає в тому, що при виклику ядра необхідно передати раніше певну розмірність сітки і блоку.

Отримання результатів і звільнення пам'яті. Після виконання ядра необхідно скопіювати результати виконання програми назад, в оперативну пам'ять за допомогою функції cudaMemcpy з вказівкою зворотного напряму копіювання (з GPU в CPU). Точно також, як і у будь-якій С- програмі для запобігання витокам пам'яті необхідно звільнити усі виділені ресурси.

Організація пам'яті в CUDA наведена в додатку В 08-36.МКР.003.00.000 ПЛ2.

Архітектура апаратної платформи Nvidia наведена в додатку Г 08-36.МКР.003.00.000 ПЛ3.

3. ПРОГРАМНА ТА АПАРАТНА РЕАЛІЗАЦІЯ

3.1 Основні поняття і сфера застосування обчислення схем адвекції

Термін адвекція означає перенесення чого-небудь з однієї області в іншу. Метеорологи найбільше зацікавлені в адвекції таких змінних як температура, вологість і вихор. Оцінка адвекції на картах погоди залежить від двох показників: 1) сила вітру і 2) кут вітру відносно ліній рівної величини (ізолінії) змінної адвекції. Найсильніша адвекція спостерігається, коли вітри орієнтовані перпендикулярно (під 90 градусів) відносно ізоліній. Розрізняють два типи адвекції : позитивна і негативна адвекція. Кінцевий результат позитивної адвекції полягає у збільшенні значень змінною у напрямі перенесення.

У зв'язку із зростаючим інтересом до чисельного моделювання погодних явищ ряд авторів пропонує ознайомитися з їх методами обчислення атмосферних процесів. Основним завданням при розрахунку атмосферних явищ перенесення є чисельне рішення рівняння перенесення. Рівняння перенесення — рівняння в приватних похідних, що описує перенесення скалярної величини, що зберігається, в просторі.

Рівняння перенесення має вигляд:

$$\frac{\partial \psi}{\partial t} + \nabla \bullet F = 0 \tag{3.1}$$

де ∇• — оператор дивергенції, а F — вектор потоку скалярної величини.

Він дорівнює добутку величини на швидкість: ψu. Часто передбачається, що поле швидкостей соленоїд, тобто ∇u = 0. В цьому випадку рівняння набирає вигляду:

$$\frac{\partial \psi}{\partial t} + u * \nabla \psi = 0 \tag{3.2}$$

У одновимірній постановці має вигляд:

$$\frac{d\psi}{dt} + u\frac{d\psi}{dx} = 0 \tag{3.3}$$

Якість загального рішення задачі адвекції сильно залежить від того, яким чином апроксимовані члени, що описують адвективне перенесення. Помилки апроксимації, що проявляються у вигляді паразитної дисперсії і в'язкості апроксимації, призводять не лише до кількісного, але і до якісного спотворення отримуваного рішення. Зважаючи на велику практичну значущість, проблема пошуку нових напрямів кардинального поліпшення дисипативних і дисперсійних властивостей чисельних схем для рівняння перенесення є надзвичайно актуальною, а відповідне завдання може бути віднесене до розряду фундаментальних завдань обчислювальної гідродинаміки. Саме важливістю застосувань пояснюється велика увага дослідників до рішення подібного роду рівнянь. Останніми роками інтенсивне вивчення проблем механіки суцільного середовища, прогнозу погоди, поширення що забруднює речовин, динаміки океану привело до розробки досить універсальних і ефективних методів знаходження чисельного рішення рівняння адвекції [7, 8]. Разом з цим триває і пошук принципово нових підходів до рішення цієї задачі. Стійкість цих схем перевіряється за критерієм Куранта- Фридрихса-Леві.

У одновимірному випадку умова має вигляд:

$$u\frac{\Delta t}{\Delta x} < C \tag{3.4}$$

де u — швидкість перенесення, Δt — часовий крок, Δx - просторовий крок, а константа C залежить від рівняння, але не залежить від Δt і Δx.

Спроби використати технологію CUDA для розрахунків складних фізичних процесів, а саме коагуляції, робилися, наприклад, роботі Раба Н.О. «Розробка і реалізація алгоритму розрахунку коагуляції в моделі хмар зі

змішаною фазою з використанням технології CUDA» [10]. У моїй роботі розглядається алгоритм Андреаса Ботта, що використовує чисельно-різницевий метод «вгору по потоку» з модифікаціями, дозволяє досягти стійкості навіть при значному градієнті значень.

3.2 Опис алгоритму розрахунку схем адвекції

Андреас Ботт у своїй статті [9] описує алгоритм розрахунку позитивно певної версії інтегральних потокових форм Трембека, ґрунтований на методі «перших різниць по потоку». Цей алгоритм може бути застосований не лише для інтегральних потокових схем, але і розширений для застосування в інших відомих схемах, які можуть бути описані потоковою формою.

Наступне рівняння описує перенесення кількості речовини ψ (x, y, z, t) в полі потоку :

$$\frac{d\psi}{dt} = -\nabla * (v\psi) \tag{3.5}$$

Для простоти розглянемо лише одновимірний випадок:

$$\frac{d\psi}{dt} = -\frac{du\psi}{dx} \tag{3.6}$$

Якщо припустити, що крок сітки Δx і крок дискретизації часу Δt константні, то кінцеві різниці виглядатимуть таким чином:

$$\psi_j^{n+1} = \psi_j^n - \frac{\Delta t}{\Delta x}[F_{j+1/2}^n - F_{j-1/2}^n] \tag{3.7}$$

де ψ_j^n значення ψ в комірці сітки j після n кроків часу і $F^n_{j+1/2}$ потоки $F^n_{j\ -\ 1/2}$ через лівий і правий кордон комірку відповідно.

Використання методу «вгору по потоку» дає ψ-потік через праву границю

$$F^n_{j+1/2} = \frac{\Delta x}{\Delta t}[c_j^+ \psi^n_j - c_j^- \psi^n_{j+1}] \tag{3.8}$$

де $c_j^{\pm} = \pm \frac{c^n_{j+\frac{1}{2}} \pm \left| c^n_{j+\frac{1}{2}} \right|}{2}$, $c^n_{j+\frac{1}{2}}$ число Куранта, визначуване як

$$c^n_{j+\frac{1}{2}} = u^n_{j+\frac{1}{2}} \frac{\Delta t}{\Delta x} \tag{3.9}$$

Для спрощення запису далі, там де це можливо індекс n буде опущений. Чисельна стабільність схеми гарантується критерієм Куранта- Фридріха-Леві

$$c_j^+ + c_{j-1}^- \leq 1 \tag{3.10}$$

в усьому полі потоку в кожен крок часу.

Важливо, щоб початкові дані для методу «висхідного потоку» були позитивно визначені і консервативні. На жаль, це тільки перший порядок за часом і простору, через що метод дає сильне чисельне загасання. Це робить таку схему досить даремною, особливості у випадках великої різниці в значеннях ψ.

Сильне числове загасання виходить в схемі «вгору по потоку» за рахунок низького рівня представленості ψ на кроці алгоритму з константними значеннями в комірках. Кроули і Трімбек використали метод поліноміальної підгонки для отримання кращого опису ψ-полів. Згідно їх концепції передбачається, що в комірці j розподіл ψ описується поліномом порядку l.

$$\psi_{ij}(x') = \sum_{k=0}^{l} a_{j,k} x'^{k} \qquad (3.11)$$

де $x' = (x - x_j)/\Delta x$ и $-1/2 \le x' \le 1/2$. Коефіцієнт $a_{j,k}$ - функція від $(l + 1)$ ψ-значення і може бути отримано інтерполяцією ψ-кривої по сусідніх елементах таблиці. Значення $a_{j,\,k}$ для l = 0,.,4 представлені в таблиці 3.1.

Коли будуються поліноми парної міри потрібно значення в лівій і правій сусідніх комірках, для поліномів непарної міри потрібна ще додаткова точка x_(i) (i = j± (l+1) /2). Тому в таблиці 1 представлені для l = 1 і 3 2 значення для i=j -(l+1) /2(1a,3a) і для i = j -(l+1) /2(1b,3b). Як і очікується асиметричність поліномів непарних мір впливає на чисельні результати, особливо для l = 1.

Таблиця 3.1 - Значення коефіцієнтів aj, k у виразі (3.11)

l	$a_{j,0}$	$a_{j,\,1}$	$a_{j,2}$	$a_{j,3}$	$a_{j,4}$
0	ψ_j	–	–	–	–
1a	ψ_j	$\psi_{j+1} - \psi_j$	–	–	–
1b	ψ_j	$\psi_j - \psi_{j-1}$	–	–	–
2	ψ_j	$1/2\left(\psi_j - \psi_{j-1}\right)$	$1/2\left(\psi_{j+1} - 2\psi_j - \psi_{j-1}\right)$	–	–
3a	ψ_j	$1/6(-\psi_{j+2} + 6\psi_{j+1} - -3\psi_j - 2\psi_{j-2})$	$1/2\left(\psi_{j+1} - 2\psi_j - \psi_{j-1}\right)$	$1/6(\psi_{j+2} - 3\psi_{j+1} - -3\psi_j - \psi_{j-2})$	–
3b	ψ_j	$1/6(2\psi_{j+1} - 6\psi_{j-1} + +3\psi_j - \psi_{j-2})$	$1/2\left(\psi_{j+1} - 2\psi_j - \psi_{j-1}\right)$	$1/6(\psi_{j+2} - 3\psi_{j+1} - -3\psi_j - \psi_{j-2})$	–
4	ψ_j	$1/12(-\psi_{j+2} + 8\psi_{j+1} - -8\psi_{j-1} + \psi_{j-2})$	$1/24(-\psi_{j+2} + 16\psi_{j+1} - -30\psi_j + 16\psi_{j-1} - \psi_{j-2}$	$1/12(-\psi_{j+2} - 2\psi_{j+1} + +2\psi_{j-1} - \psi_{j-2})$	$1/24(\psi_{j+2} - 4\psi_{j+1} + +6\psi_j - 4\psi_{j-1} + \psi_{j-2}$

Визначимо і вичислимо інтеграли:

$$I_l^+\left(c_{j+\frac{1}{2}}\right) = \int_{\frac{1}{2}-c_j^+}^{1/2} \psi_{j,l}(x')\,dx' = \sum_{k=0}^{l} \frac{a_{j,k}}{(k+1)2^{k+1}}\left[1-(1-2c_j^+)^{k+1}\right] \quad (3.12)$$

$$I_l^-\left(c_{j+\frac{1}{2}}\right) = \int_{-\frac{1}{2}}^{-\frac{1}{2}+c_j^-} \psi_{j+1,l}(x')\,dx' =$$

$$= \sum_{k=0}^{l} \frac{a_{j+1,k}}{(k+1)2^{k+1}}(-1)^k\left[1-(1-2c_j^+)^{k+1}\right] \quad (3.13)$$

отримаємо ψ-поток через правий кордон комірки j

$$F_{j+1/2} = \frac{\Delta x}{\Delta t}\left[I_l^+\left(c_{j+\frac{1}{2}}\right) - I_l^-\left(c_{j+\frac{1}{2}}\right)\right] \quad (3.14)$$

У такій формі схема консервативна і підходить під потокову форму Тримбека. Схема позитивно визначена, достатня умова для цього

$$0 \le I_l^+\left(c_{j+\frac{1}{2}}\right) + I_l^-\left(c_{j-\frac{1}{2}}\right) \le \psi_j \quad (3.15)$$

Отже, кінцевий потік завжди позитивний і обмежений ψ_j . При l = 0 ця умова виконується, але для l > 0 за виконанням цієї умови треба стежити. Наприклад, якщо l≥2 і $\left|c_{j+1/2}\right| = 1$ аналіз стабільності Неймана дає коефіцієнт відношення лінійної і фазової швидкості менше одиниці, що показує що рішення цього рівняння адвекції неможливе. Це відбувається внаслідок того, що для $\left|c_{j+1/2}\right| = 1$ аналітичне рішення можливо тільки якщо комірки площини, що покривається $\psi_{j,l}\,(x')$ задаються формулою $\psi_j\,\Delta x$. Ця умова завжди вірна для *l = 0, 1,* проте для поліномів вище 2-го порядку це, в загальному випадку,

не вірно. Тепер помножимо $\psi_{j,\ l}(x')$ на ваговий коефіцієнт $\psi_j/\ \ I_{i,j}$ де $I_{l,j} = I_l^{\pm}\ (c_{j\pm1/2} = \pm1)$ i

$$I_{i,j} = \int_{-1/2}^{1/2} \psi_{j,l}(x')dx' = \sum_{k=0}^{l} \frac{a_{j,k}}{(k+1)2^{k+1}}[(-1)^k + 1] \tag{3.16}$$

Тепер уся площина, покрита зваженими поліномами, в кожному комірку визначається формулою $\psi\ \Delta x$ і адвективна схема даватиме точні результати при $\left|c_{j+1/2}\right| = 1$, якщо інтеграл в правій частині виразу (3.12) теж домножити на відповідні вагові коефіцієнти. Отримуємо наступну формулу потоку:

$$F_{j+1/2} = \frac{\Delta x}{\Delta t}\left[\frac{I_l^+\left(c_{j+\frac{1}{2}}\right)}{I_{l,j}}\psi_j - \frac{I_l^-\left(c_{j+\frac{1}{2}}\right)}{I_{l,j+1}}\psi_{j+1}\right] \tag{3.17}$$

Нарешті, позитивна визначеність алгоритму досягається якщо адвективні потоки обмежені згори і знизу відповідно до (3.12) і (3.13). Для позитивних швидкостей при $I_l^+(c_{j+1/2}) < 0\ F_{j+1/2}$ встановлюється в 0. Крім того потоки мають бути обмежені ψj Δx / Δt якщо $I_l^+(c_{j+1/2}) > I_{l,j}$. Відповідні обмеження мають бути введені і для негативних швидкостей. Після об'єднання усіх необхідних обмежень, адвективна схема може бути записана таким чином:

$$F_{j+1/2} = \frac{\Delta x}{\Delta t}\left[\frac{i_{l,j+1/2}^+}{i_{l,j}}\psi_j - \frac{i_{l,j+1/2}^-}{i_{l,j+1}}\psi_{j+1}\right] \tag{3.17}$$

де

$$i^{+}_{l,j+1/2} = \max\left(0, I^{+}_{l}\left(c_{j+\frac{1}{2}}\right)\right)$$

$$i^{-}_{l,j+1/2} = \max\left(0, I^{-}_{l}\left(c_{j+\frac{1}{2}}\right)\right)$$

$$i_{l,j} = \max\left(I_{l,j}, i^{+}_{l,j+1/2} + i^{i}_{l,j+1/2} + \epsilon\right)$$

Знак $\in$ введений щоб уникнути ділення на нуль при $i_{l,j} = 0$.

За описаною вище схемою можна реалізувати алгоритм розрахунку схем адвекції. Розглянемо одновимірний випадок перенесення субстанцій з постійною швидкістю u > 0 уздовж траєкторій. В якості початкових даних ми маємо початковий розподіл ψ (x) при t=0, відповідно можемо заповнити масив, розмір якого залежить від кроку сітки початковими значеннями.

Вхідні дані: початкові концентрації аерозольних часток у момент часу t_0 (зчитуються з файлу), число розбиття варіювалося від 100 000 до 20 000 000, $t_{зад}$ - інтервал часу, для якого розраховувалося перенесення.

Вихідні дані: концентрації аерозольних часток, у момент часу $t_{зад}$ Для масиву вхідних даних по формулах з таблиці 3.1 рахуємо $a_{j,k}$ і записуємо в масив результатів. Залежність часу виконання від об'єму даних представлена в таблиці 3.2. По формулі (3.14) обчислюємо потоки через правий кордон і записуємо в масив вихідних даних. Перераховуємо значення в комірках. Вхідні дані(початкова концентрація в комірках сітки) прочитуються з файлу. Лістинги коду в застосуванні. Копіювання даних між потоками, що виконуються, приведене на малюнку 3.1 Оскільки для обчислення потоку через правий кордон комірку потрібно значення розподілу в 2-х комірках справа і зліва від і- й. Далі значення, розраховане і- м потоком записується в і- ю комірок масиву значень розрахованих потоків через правий кордон. І нарешті, нове значення концентрації в і- й комірку розраховується і- м потоком і записується в результуючий масив значень. Цей масив і передається в якості результату в CPU.

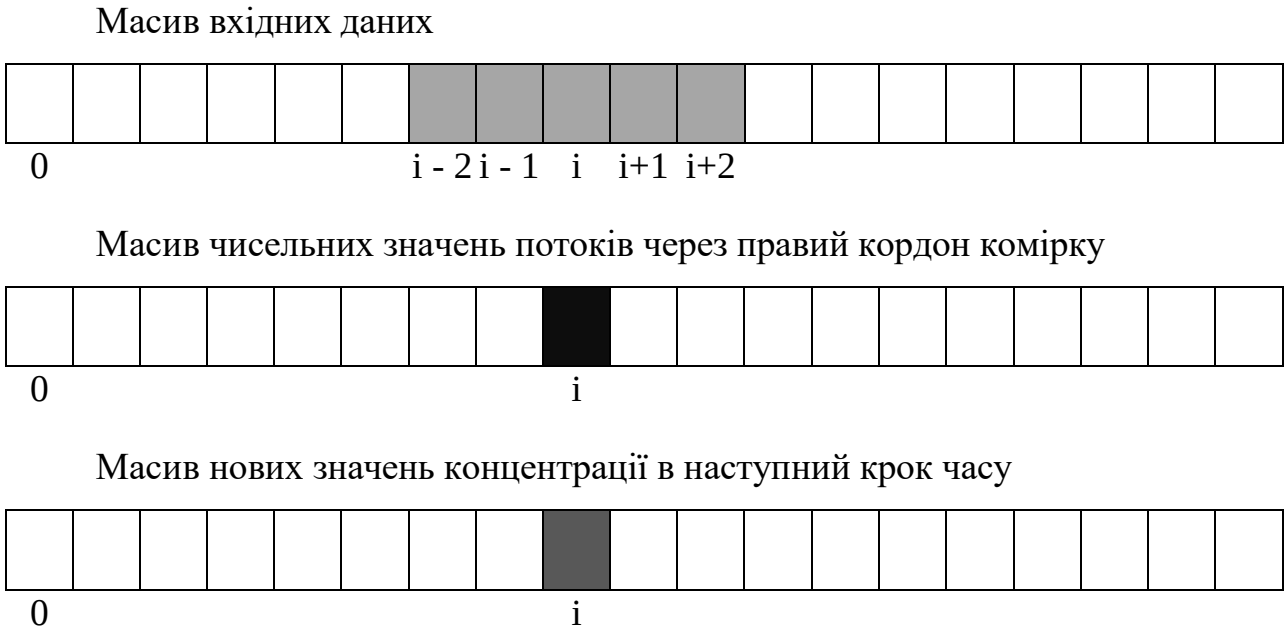


Рисунок 3.1 - Передача даних між потоками

3.3 Реалізація алгоритму

Розглянуті в другому розділі алгоритми створення мережі і обробки вхідних даних можуть бути реалізовані за допомогою паралельної архітектури CUDA. Як було сказано, найбільш витратним процесом є передача безлічі значень для вхідних змінних, і обчислення функцій їх, що використовують. Розберемо, як саме технологія CUDA дозволить розпаралелювати цей процес.

Перш ніж почати виконання обчислень, при використанні CUDA, необхідно виділити пам'ять під усі використовувані значення. Це робиться функцією «cudaMalloc». Для використання цієї функції необхідно створити покажчик на початок ділянки, в яку будуть поміщені використовувані дані, а також розмір цих даних у байтах. У реалізації це виглядатиме таким чином:

Int *dev _ ptr = 0;

cudaMalloc((void**)&dev _ ptr, sizeof(int) *N), де N - кількість використовуваних значень.

Після виділення пам'яті, у вказане місце необхідно помістити самі значення, для чого використовується функція

«cudaMemcpy(dev _ ptr, input, N * sizeof(int), cudaMemcpyHostToDevice);«, де

dev _ ptr - наш покажчик

input - передаване значення

N * sizeof(int), - розмір у байтах

cudaMemcpyHostToDevice - напрям передачі даних, HostToDevice означає передачу з центрального процесора (Host) на процесор графічної карти (Device).

Якщо кількість вхідних значень перевищує кількість ядер на графічній карті, потрібна реалізація циклу, що відповідає за часткову передачу значень.

Після того, як дані були передані в пам'ять графічної карти, необхідно запустити «device» - функцію, що виконує задані нами операції на процесорі відеокарти. Для цього використовується запис наступного типу :

«addKernel<<<grid, threads>>> (dev _ ptr, dev _ array);«,

де

addKernel - ім'я функції

grid - кількість використовуваних блоків пам'яті, ціле число

threads - кількість ниток в кожному блоці, ціле число

dev _ array - масив, в якому записана структура мережі і усі введені спочатку функції.

Запис виду <<<A, B>>> (dev _ ptr, dev _ array) є вказівкою компілятору виконати функцію на графічній карті, а не на центральному процесорі. У дужках вказуються передавані параметри.

Сама функція, що виконується на графічній карті має наступний прототип :

__global__ void addKernel(int *dev _ ptr, int *dev _ array);

Далі, кожне передане значення необхідно пронумерувати. Робиться це шляхом вказівки до яких блоку і нитці відноситься це значення. Введемо наступне позначення:

Int i _ x = blockIdx.x + threadIdx.x;

Оскільки використовувані нами масиви є двовимірними, потрібне ще одне позначення:

Int i _ y = blockIdx.y + threadIdx.y;

Так ми отримуємо унікальні номери для кожного з використовуваних значень а отже можемо паралельно вичислити усі функції.

3.4 Паралельна версія алгоритму

З попереднього розділу стає зрозуміло, що обчислення, необхідні для розрахунку схеми досить громіздкі і вимагають великих обчислювальних потужностей. Як правило, для цього завдання потрібно розрахунок для великого об'єму вхідних даних, що займає досить багато часу. Для прискорення роботи алгоритму прийнято рішення виконувати алгоритм в декілька потоків. Зробити це вдалося таким чином : увесь масив даних зберігається в глобальній пам'яті, до якої мають доступ усі потоки, потім кожен потік розраховує значення потоку через правий кордон осередку. Які осередки в якому потоці обробляються вирішується залежно від кількості потоків.

3.5 Апаратна реалізація

Система Nvidia Jetson Nano може використовуватись підприємствами, стартапами і дослідниками, які раніше не могли собі дозволити дорогі рішення. За рахунок зниження вартості платформи для розробників платформа Jetson значно розширює свою аудиторію, а прискорювач ШІ фактично стає набагато більш доступним.

Дана плата дозволяє користуватися сучасними рішеннями в області ШІ, стимулюючи нову хвилю інновацій від виробників, винахідників, розробників і студентів, які можуть створювати проекти зі ШІ, які раніше були неможливі, і виводити існуючі проекти на новий рівень, такі як мобільні роботи і дрони, цифрові помічники, автоматизовані прилади та багато іншого. Комплект поставляється з підтримкою повноцінного десктопного Linux, що є сумісним з багатьма популярними периферійними пристроями та аксесуарами [21].

3.5.1 NVIDIA Jetson AGX Xavier

З набором інструментів для розробника NVIDIA Jetson Xavier є можливість легкого створення і розгортання готових робототехнічних програм на базі штучного інтелекту для виробництва, транспортних сервісів, роздрібної торгівлі, розумних міст та інших областей. Завдяки підтримці SDK NVIDIA JetPack і DeepStream, а також програмних бібліотек CUDA, cuDNN і TensorRT він надає всі необхідні інструменти для швидкого початку роботи. Так як платформа працює на базі нового процесора NVIDIA Xavier, це дозволяє отримати у 20 разів більш продуктивне і в 10 разів менше енергоспоживання попередника Jetson TX2.

Jetson AGX Xavier 8 Гб - це більш енергоефективний і недорогий варіант Jetson AGX Xavier, який забезпечує повну сумісність програмно-апаратного забезпечення з Jetson AGX Xavier. Платформа споживає максимум 20 Вт, при цьому забезпечуючи до 20 тера-операцій в секунду в задачах ШІ [22].

Зовнішній вигляд модуля наведений на рис. 3 [22, 25].

Рисунок 3.2 – Зовнішній вигляд модуля NVIDIA Jetson AGX Xavier

Модуль NVIDIA Jetson Xavier, що входить до складу платформи для розробників володіє технічними характеристиками, що наведені у таблиці 2 [2, 6].

Таблиця 3.2 – Специфікація модуля NVIDIA Jetson Xavier

Графічний процесор	384-Core Volta GPU with 48 Tensor cores
Прискорювач глибокого навчання	(2x) NVDLA Engines
Процесор	6-core ARM v8.2 64-bit CPU, 6MB L2 + 4MB L3
Пам'ять	8GB 256-bit LPDDR4x - 85GB/s
Роз'єми дисплея	Three multi-mode DP 1.2a/e DP 1.4/HDMI 2.0 a/b
Флеш-пам'ять	32GB eMMC 5.1
Прискорювач комп'ютерного зору	7-Way VLIW Vision Processor
Кодування відео	6x 4K @ 30 (HEVC)
Декодування відео	4x 4K @ 30 (HEVC)
Камера	16 lanes MIPI CSI-2 8 lanes SLVS-EC
UPHY	3x USB 3.1, 4x USB 2.0 1 x8 or 1 x4 or 1 x2 or 2 x1 PCIe (Gen3)
Інші роз'єми	UART, SPI, CAN, I2C, I2S, DMIC, GPIOs
Підключення	10/100/1000 RGMII

Зовнішній вигляд платформи розробника NVIDIA Jetson Xavier Developer Kit в цілому наведений на рис. 3.3 [2]

Рисунок 3.3 – Платформа розробника NVIDIA Jetson Xavier Developer Kit в цілому

3.5.2 NVIDIA Jetson TX2

Набір інструментів для розробників NVIDIA Jetson TX2 забезпечує простий і швидкий спосіб розробки програмно-апаратних рішень для суперкомп'ютера-на-модулі Jetson TX2. Він надає апаратні можливості і інтерфейси модуля і підтримується NVIDIA Jetpack - повноцінним SDK, який містить BSP, бібліотеки глибокого навчання, комп'ютерного зору, обчислень на GPU, обробки мультимедіа та інших завдань.

Jetson TX2 - це суперкомп'ютер-на-модулі з рівнем енергоспоживання 7,5 Вт, який забезпечує можливості ШІ в кінцевих пристроях. Він створений на базі графічних процесорів NVIDIA Pascal, оснащений 8 Гб пам'яті і має пропускну здатність 59,7 Гбіт/с. Суперкомп'ютер містить цілу лінійку інтерфейсів, які дозволяють використовувати модуль для безлічі пристроїв і форм-факторів.

Зовнішній вигляд модуля наведений на рис. 3.4 [22, 27].

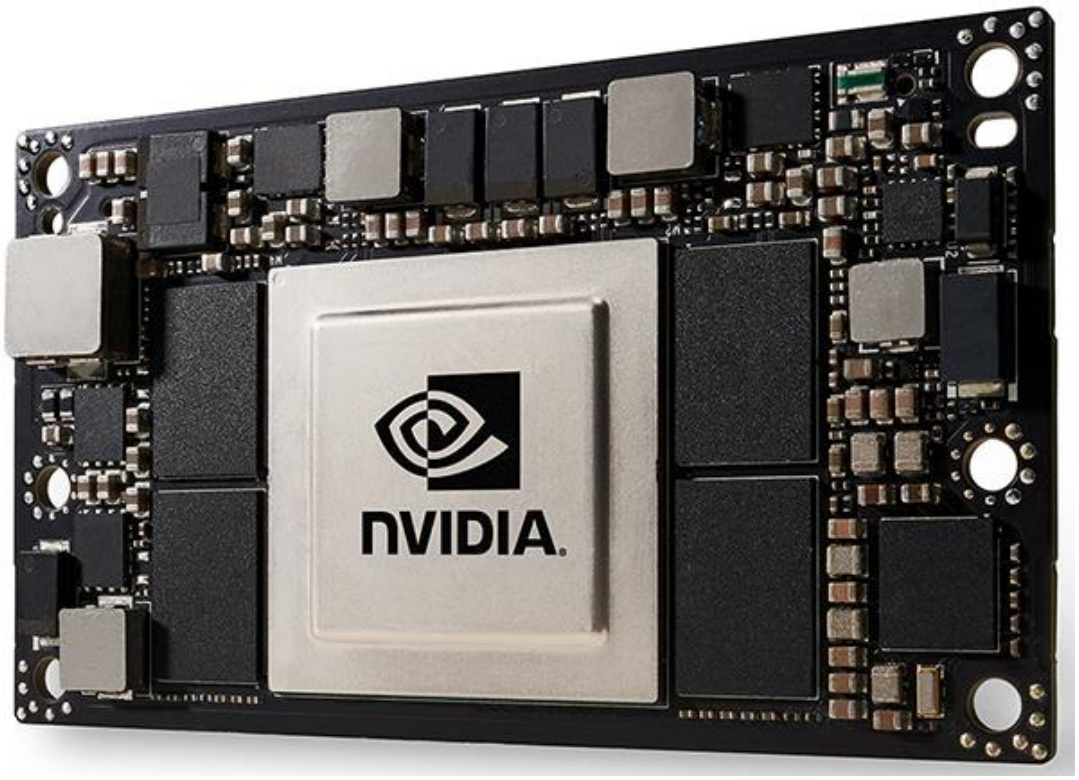


Рисунок 3.4 – Зовнішній вигляд модуля NVIDIA Jetson TX2

Модуль NVIDIA Jetson TX2, що входить до складу платформи для розробників володіє технічними характеристиками, що наведені у таблиці 3 [22].

Таблиця 3.3 – Специфікація модуля NVIDIA Jetson TX2

Графічний процесор	Архітектура NVIDIA ™ Maxwell ™ з 256 ядрами NVIDIA CUDA®
Процесор	Двоядерний процесор Denver 2 64-біт і двоядерний ARM A57
Пам'ять	8 Гб LPDDR4, 128-біт
Флеш-пам'ять	32 Гб eMMC 5.1
Кодування відео	2 потоки в дозволі 4K з частотою 30 Гц (HEVC)
Кодування відео	2 потоки в дозволі 4K з частотою 30 Гц, підтримка 12-

	біт
Підключення	Вбудоване підключення Wi-Fi
	Gigabit Ethernet
Камера	12 каналів MIPI CSI-2, D-PHY 1.2 (30 Гбіт / с)
Роз'єми дисплея	HDMI 2.0 / eDP 1.4 / 2 роз'єми DSI / 2 роз'єми DP 1.2
UPHY	Gen 2 | 1x4 + 1x1 або 2x1 + 1x2, USB 3.0 + USB 2.0
Розмір	87 мм x 50 мм
підключення	Роз'єм 400-pin з теплообмінником (TTP)

Зовнішній вигляд платформи розробника NVIDIA Jetson TX2 Developer Kit в цілому наведений на рис. 3.5 [22]

Рисунок 3.5 – Платформа розробника NVIDIA Jetson TX2 Developer Kit в цілому

3.5.3 NVIDIA Jetson Nano

Jetson Nano має високу продуктивність та можливості для швидкого виконання найактуальніших завдань штучного інтелекту і дозволяє забезпечити сучасні можливості ШІ в кожному рішенні. Jetson Nano забезпечує можливості ШІ для вбудованих і IoT-додатків, в тому числі для відеореєстраторів початкового рівня, домашніх роботів та інтелектуальних шлюзів з можливостями аналітики.

Набір інструментів для розробників NVIDIA Jetson Nano - це компактний і потужний комп'ютер, який дозволяє паралельно запускати кілька нейронних мереж в додатках для класифікації зображень, розпізнавання об'єктів, сегментації та обробки мови. Зручна у використанні платформа дозволяє виконувати всі ці завдання і споживає при цьому всього 5 Вт.

Зовнішній вигляд модуля наведений на рис. 3.6 [22, 23].

Рисунок 3.6 – Зовнішній вигляд модуля NVIDIA Jetson Nano

Модуль NVIDIA Jetson Nano, що входить до складу платформи для розробників має технічні характеристики, що наведені у таблиці 3.4 [22, 24].

Таблиця 3.4 – Специфікація модуля NVIDIA Jetson Nano

Графічний процесор	NVIDIA Maxwell ™ Архітектура ® з 128 ядрами NVIDIA CUDA
Процесор	Чотириядерний процесор ARM® Cortex®-A57® Cortex®-MPCore
Пам'ять	4 Гб LPDDR4, 64-біт
Флеш-пам'ять	16 Гб eMMC 5.1 Flash
Кодування відео	1x 4K @ 30 (HEVC)
Декодування відео	1x 4K @ 60 (HEVC)
Камера	12 каналів (3x4 або 4x2) MIPI CSI-2 DPHY 1.1 (1,5 Гбіт / с)
Підключення	Gigabit Ethernet
Роз'єми дисплея	Роз'єм HDMI 2.0 або DP1.2 | eDP 1.4 | 2 роз'єми DSI (1x2), що підключаються одночасно
UPHY	1 роз'єм 1/2/4 PCIE, 1 роз'єм USB 3.0, 3 роз'єми USB 2.0
Введення / Виведення даних	3x UART, 2x SPI, 2x I2S, 4x I2C, GPIOs
Розмір	69,6 мм x 45 мм
Підключення	Роз'єм 260-pin

Зовнішній вигляд платформи розробника NVIDIA Jetson Nano Developer Kit в цілому наведений на рис. 3.9 [12]

Рисунок 3.7 – Зовнішній вигляд модуля NVIDIA Jetson Nano Developer Kit

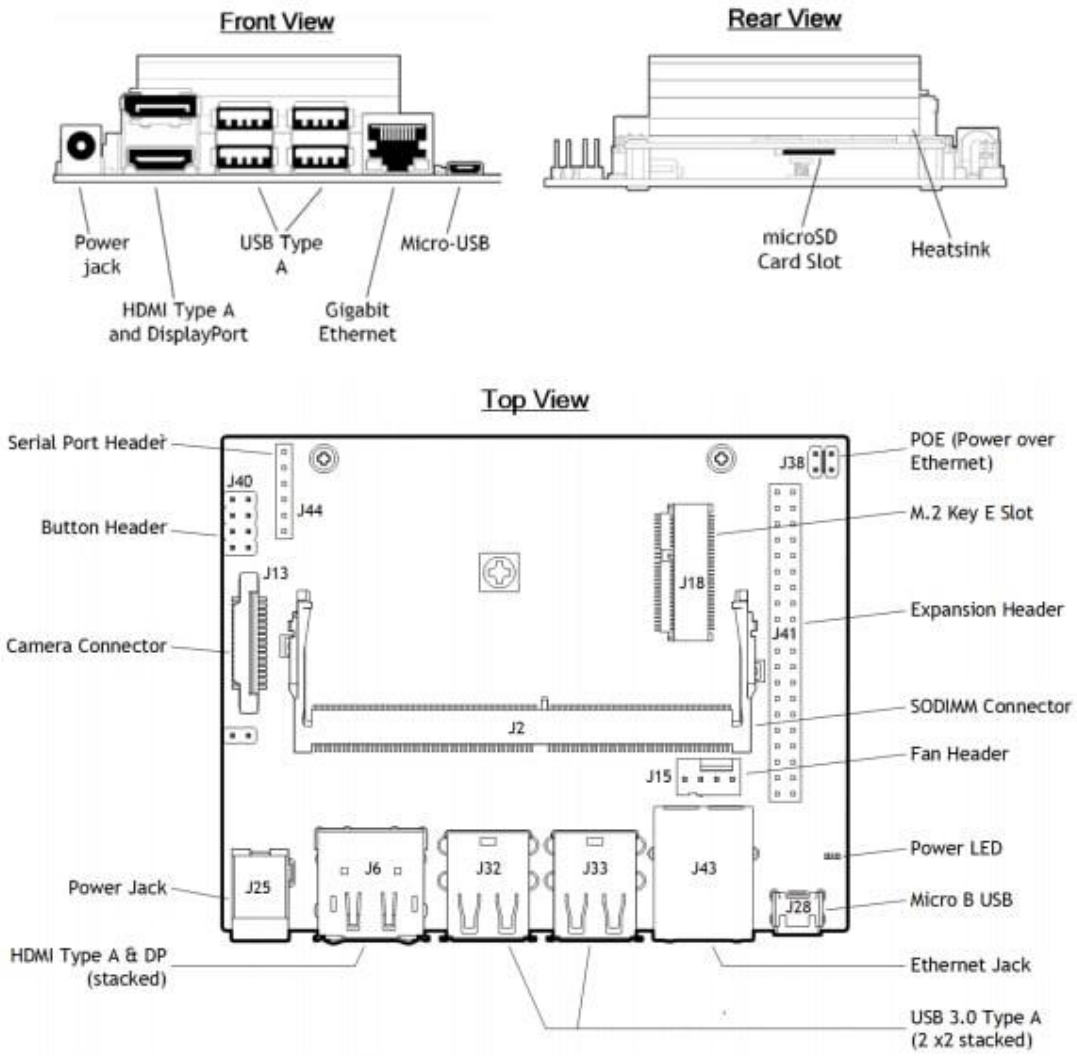


Рисунок 3.8 – Структурна схема NVIDIA Jetson Nano

3.6 Висновки до третього розділу

В даній роботі було розглянуто автономні рішення від Nvidia для задач штучного інтелекту. Було розглянуто такі платформи як NVIDIA Jetson Nano, Jetson AGX Xavier та Jetson TX2, порівняння даних платформ показало, що оптимальним рішенням по співвідношенню вартість/обчислювальна потужність є Nvidia Jetson Nano, а найбільш продуктивним і дороговартісним автономним рішенням є Jetson AGX Xavier [28, 29].

Опис алгоритму розрахунку схем адвекції наведені в додатку Д 08-36.МКР.003.00.000 ПЛ4.

Апаратна реалізація наведена в додатку Е 08-36.МКР.003.00.000 ПЛ5.

4 РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ

4.1 Розпаралелювання алгоритму за допомогою бібліотеки OpenMP

OpenMP реалізує паралельні обчислення за допомогою багатопоточності, в якій «головний» (master) потік створює набір підпорядкованих (slave) потоків і завдання розподіляється між ними. Передбачається, що потоки виконуються паралельно на машині з декількома процесорами (кількість процесорів не обов'язково має бути більша або дорівнює кількості потоків).

За рахунок ідеї "інкрементального розпаралелювання" OpenMP ідеально підходить для розробників, що бажають швидко розпаралелювати свої обчислювальні програми з великими паралельними циклами. Розробник не створює нову паралельну програму, а просто послідовно додає в текст послідовної програми OpenMP - директиви. При цьому, OpenMP - досить гнучкий механізм, що надає розробникові великі можливості контролю над поведінкою паралельного застосування. Передбачається, що OpenMP - програма на однопроцесорній платформі може бути використана в якості послідовної програми, тобто немає необхідності підтримувати послідовну і паралельну версії. Директиви OpenMP просто ігноруються послідовним компілятором, а для виклику процедур OpenMP можуть бути підставлені заглушки (stubs), текст яких приведений в специфікаціях. Одним з достоїнств OpenMP його розробники рахують підтримку так званих "orphan" (відірваних) директив, тобто директиви синхронізації і розподілу роботи можуть не входити безпосередньо в лексичний контекст паралельної області.

Завдання, що виконуються потоками паралельно, також як і дані, потрібні для виконання цих завдань, описуються за допомогою спеціальних директив препроцесора відповідної мови — прагм. Наприклад, ділянка коду на мові Fortran, яка повинна виконуватися декількома потоками, кожен з яких має свою копію змінної N, упереджається наступною директивою: !$OMP

PARALLEL PRIVATE(N)

Кількість створюваних потоків може регулюватися як самою програмою за допомогою виклику бібліотечних процедур, так і ззовні, за допомогою змінних оточення.

Ключовими елементами OpenMP є

- конструкції для створення потоків (директива parallel)
- конструкції розподілу роботи між потоками (директиви DO/for і section)
- конструкції для управління роботою з даними (вирази shared і private)
- конструкції для синхронізації потоків (директиви critical, atomic і barrier)
- процедури бібліотеки підтримки часу виконання (наприклад, omp _ get _ thread _ num)
- змінне оточення (наприклад, OMP _ NUM _ THREADS).

При використанні OpenMP обчислення проводяться засобами CPU. Отже максимальне число одночасно використовуваних ядер залежить від характеристик CPU. Зазвичай найефективніше використати число потоків рівне числу ядер (фізичних або фізичних і віртуальних при використанні технології Intel Hyper - Threading). Менше число потоків спричиняє за собою простою ядер, більше - черги на виконання і відповідні накладні витрати.

У реалізації явно представлений цикл по усіх елементах і немає обчислень для визначення поточного індексу оброблюваного елементу, для усіх осередків таблиця потік розраховується однаково. Так само з OpenMP програмістові немає необхідності думати про зберігання інформації на окремому пристрої. Усі дані зберігаються в одному місці. І передачу даних між потоками бібліотека бере на себе.

Реалізація алгоритму розрахунку схеми адвекції з використанням OpenMP.

Таблиця 4.1 - Залежність часу виконання від кількості потоків і об'єму даних

К-ть потоків/ розмір масиву даних	2	4	6	8	10
100 000	1,917	1,388	1,205	1,139	1,138
1 000 000	19,122	14,863	11,53	10,837	11,171
10 000 000	199,49	140,12	102,85	109,39	111,93
20 000 000	407,62	279,64	241,27	229,7	216,93

Оскільки для обчислень використовувався двоядерний процесор, та кількість потоків вибралася кратне двом, оскільки при такій реалізації можна добитися найбільшої продуктивності. Залежність часу виконання від кількості потоків і об'єму даних представлена в таблиці 3.1. Час вимірюється в секундах.

Графік залежності часу виконання від об'єму вхідних даних представлений на рис. 4.1

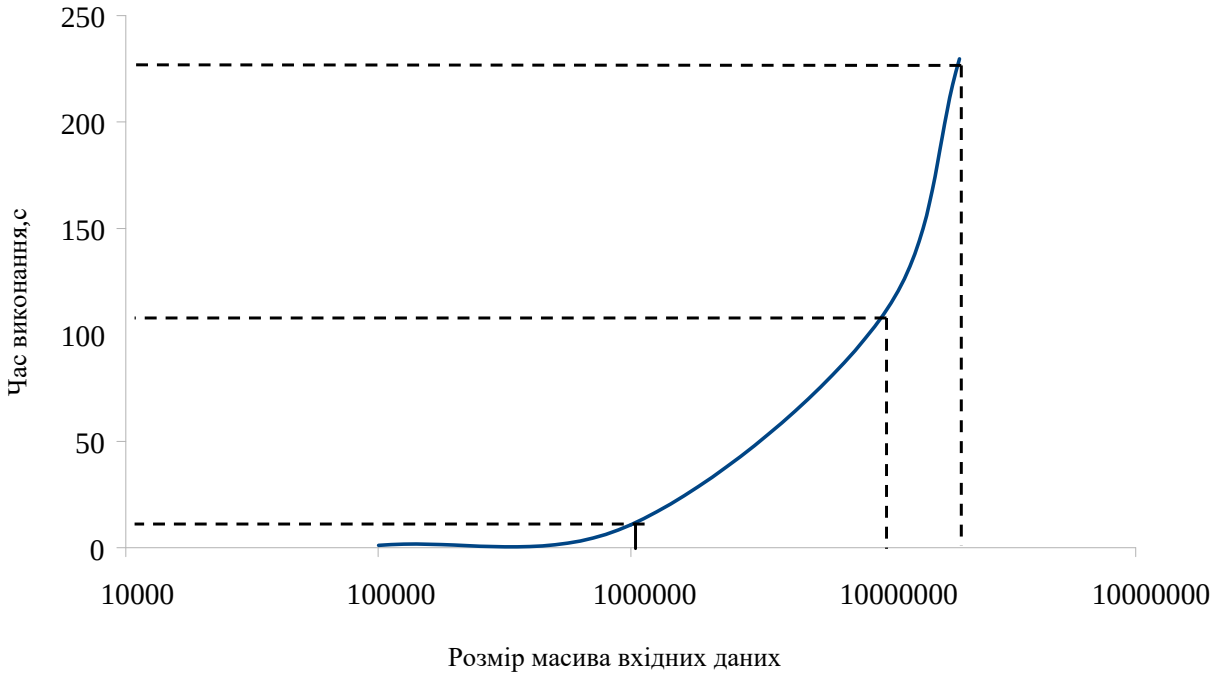


Рисунок 4.1 - Графік залежності часу виконання від об'єму вхідних даних

Таким шляхом ми змогли прискорити алгоритм приблизно в 1,6 разу. Навіть при розпаралелюванні алгоритму на виконання одного кроку циклу алгоритму потрібно значний час. Причиною цьому являється те, що на фізичних двох ядрах розділення більш ніж на 2 потоки швидше логічне, оскільки потоки все одно чекають своєї черги для виконання на ядрі. Графік залежності прискорення від об'єму вхідних даних представленні на рисунку 4.2.

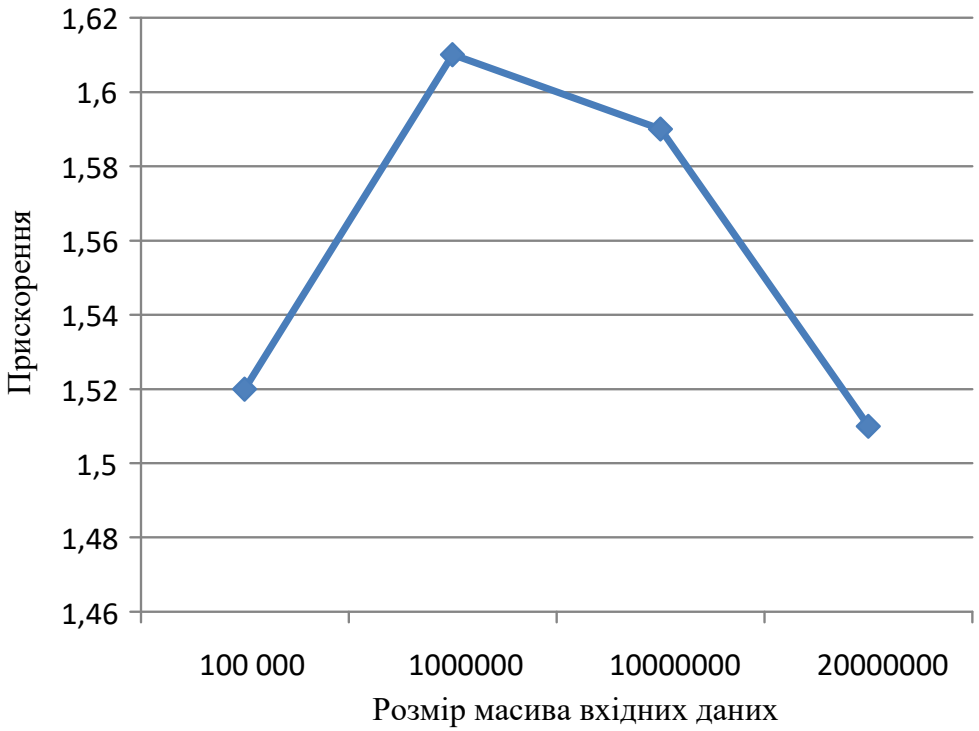


Рисунок 4.2 - Графік залежності прискорення від об'єму вхідних даних

4.2 Розпаралелювання алгоритму за допомогою технології CUDA

При обчисленні одного кроку алгоритму за допомогою GPU ми можемо задіяти кількість ядер, рівну кількості осередків сітки у кращому разі. Для цього у відеопам'ять копіюємо максимальну кількість цих осередків, яку можливо обробити.

```
float* allocAndCopyDataToDevice(list<float> &data)

float *devicePtr = nullptr;

float *tmpArray = new float[data.size()](); size _ t dataSize = data.size();

cudaError _ t error;

copy(data.begin(), data.end(), tmpArray);

error = cudaMalloc((void**)&devicePtr, dataSize*sizeof(float));

cout << "cudaMallocError: " << cudaGetErrorString(error)  << endl; cout <<
cudaGetErrorString(cudaGetLastError())  << endl;

error = cudaMemcpy(devicePtr, tmpArray, dataSize*sizeof(float),
cudaMemcpyHostToDevice); cout << "cudaMemcpyError: " <<
cudaGetErrorString(error)  << endl;

cout << cudaGetErrorString(cudaGetLastError())  << endl;

cout << "devDataPtr: " << devicePtr << " data size: " << dataSize << endl;
delete[] tmpArray;

return devicePtr;

}
```

У кожне виконуване ядро передаємо осередок з номером ядра і по 2 сусідніх ліворуч і справа, по формулах з таблиці 3.1 рахуємо aj, k і записуємо в масив результатів. Обчислюємо потоки через праву границю і записуємо в масив вихідних даних.

```
__global    void kernel(float* deviceOutputData, float* inputData, float *flow)
```

```
{

int index = blockIdx.x * blockDim.x + threadIdx.x;

float Iplus = max((float) 0, calcIplus(&inputData[index])); float Iminus =
max((float) 0, calcIminus(&inputData[index]));

float Iplus 1 = max((float) 0, calcIplus(&inputData[index+1])); float Iminus 0 =
max((float) 0, calcIminus(&inputData[index - 1]));

float i _ j = max(calcI(&inputData[index]), Iplus + Iminus 0 + 0.0001f);

float i j 1 = max(calcI(&inputData[index+1]), Iplus 1 + Iminus + 0.0001f);

flow[index + 1] = delta _ x/delta _ t * (Iplus*inputData[index]/i _ j -
Iminus*inputData[index+1]/i j 1);
```

Перераховуємо значення в осередках і записуємо їх в результуючий масив. Повторюємо цей цикл дій потрібна кількість разів. Копіюємо масив з відеопам'яті в ЦПУ.

Таким чином алгоритм вдалося значно прискорити

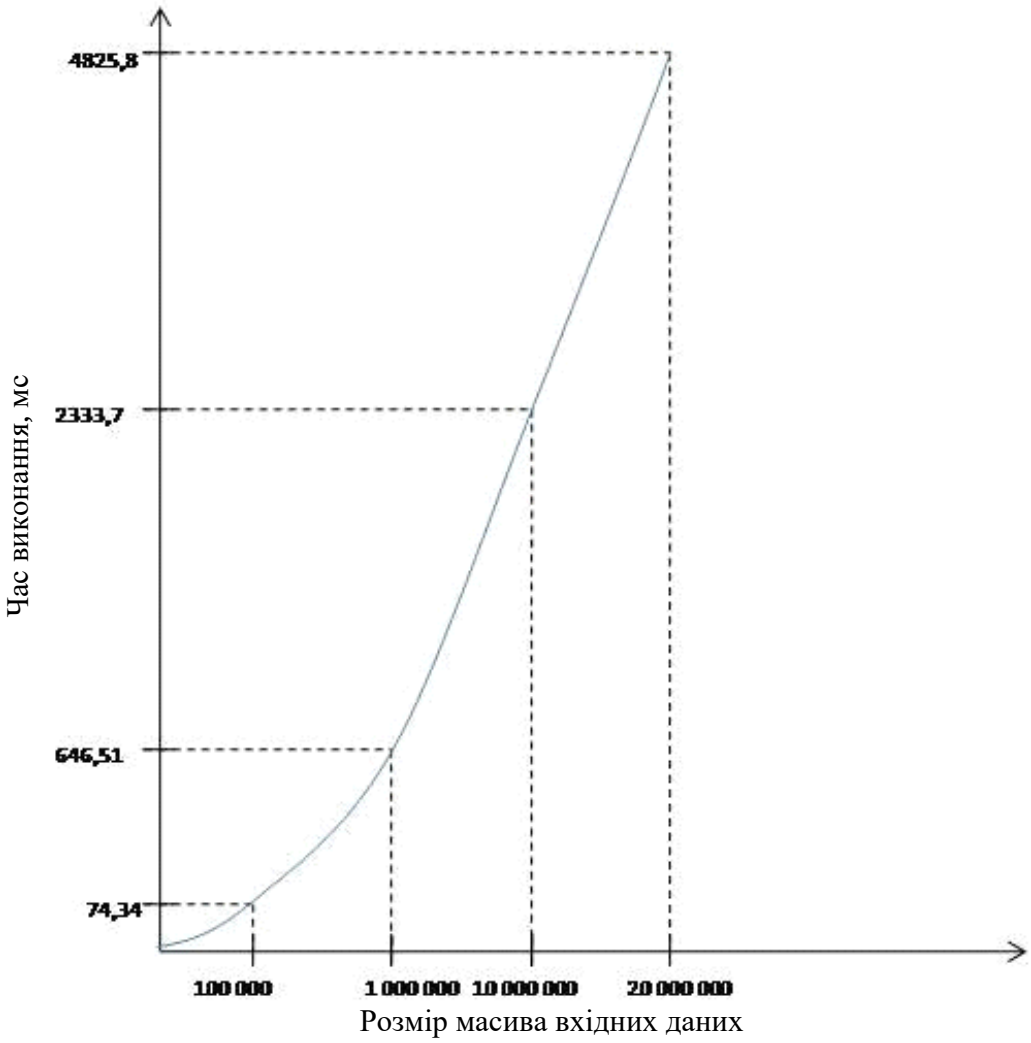


Рисунок 4.3 – Графік залежності часу виконання від розміру вхідних даних

Як видно з графіків, в середньому вдалося зменшити час виконання одного циклу алгоритму від 27 до 96 разів залежно від об'єму даних. Графік залежності прискорення від об'єму вхідних даних представлений на рис. 4.4

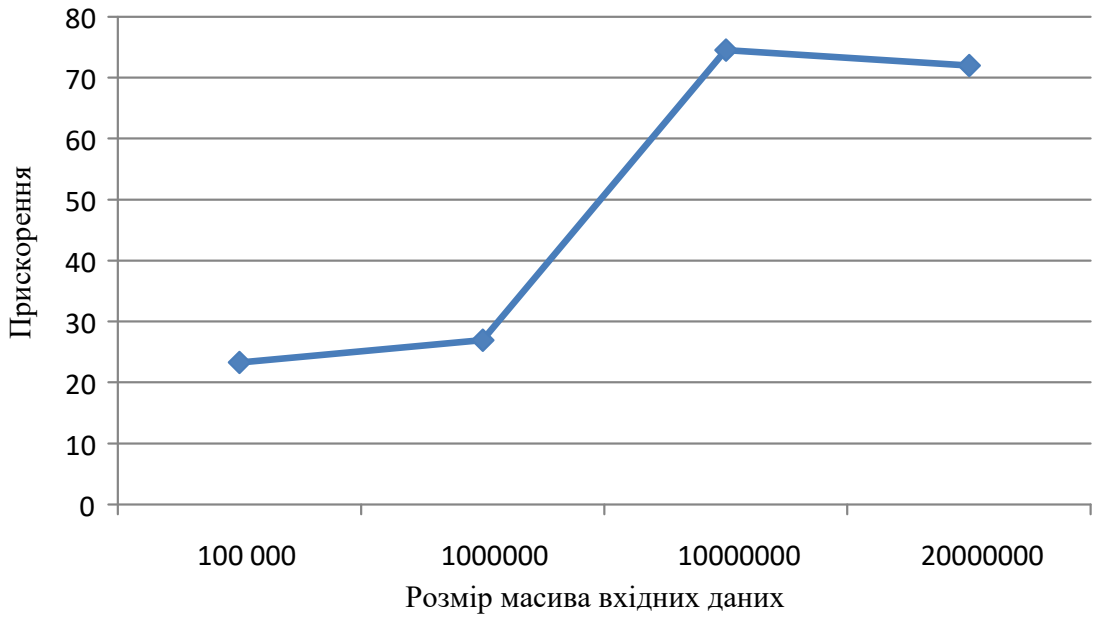


Рисунок 4.4 – Графік залежності прискорення від об'єму вхідних даних

Як видно з графіку на невеликих об'ємах даних вдалося добитися меншого прискорення, оскільки в порівнянні з часом, що витрачається на обчислення, час на передачу даних досить великий.

4.3 Прискорення алгоритму за допомогою приміщення даних в константну відеопам'ять

Велика частина часу при виконанні алгоритму витрачається на передачу даних з CPU в GPU і при копіюванні даних усередині GPU. Алгоритм можна значно прискорити, якщо початкові дані записувати в константну пам'ять, доступ до якої також є у усіх ядер, але здійснюється значно швидше. Таким чином вдалося прискорити виконання одного циклу алгоритму в середньому в 1,3 разу в порівнянні з попереднім варіантом алгоритму. Залежність часу виконання від об'єму даних представлена в таблиці 4.2.

Таблиця 4.2 – Залежність часу виконання від об'єму даних

Об'єм вхідних даних(ед.)	Час виконання (ms)
100 000	62,82
1 000 000	505,1
10 000 000	1809,0
20 000 000	3712,15

Графік залежності часу виконання від розміру вхідних даних представлений на рис. 4.5. Графік прискорення алгоритму представленні на рисунку 4.6.

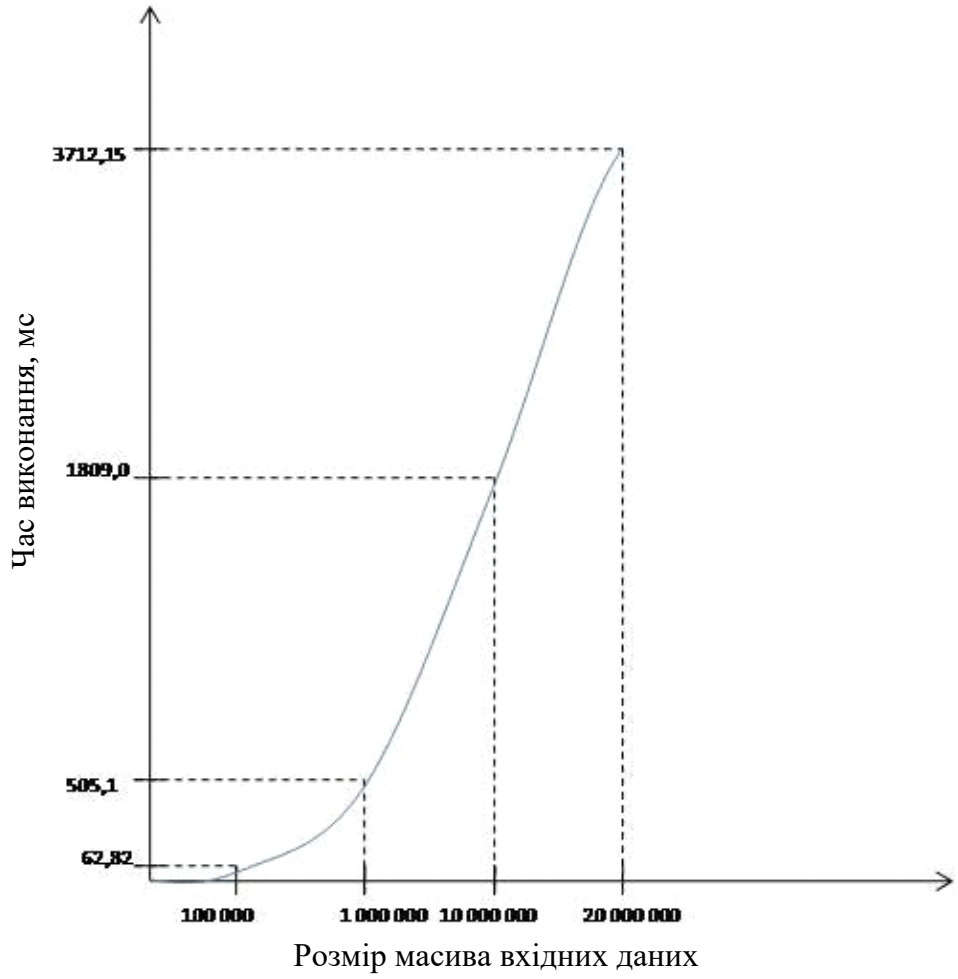


Рисунок 4.5 - Графік залежності часу виконання від розміру вхідних даних

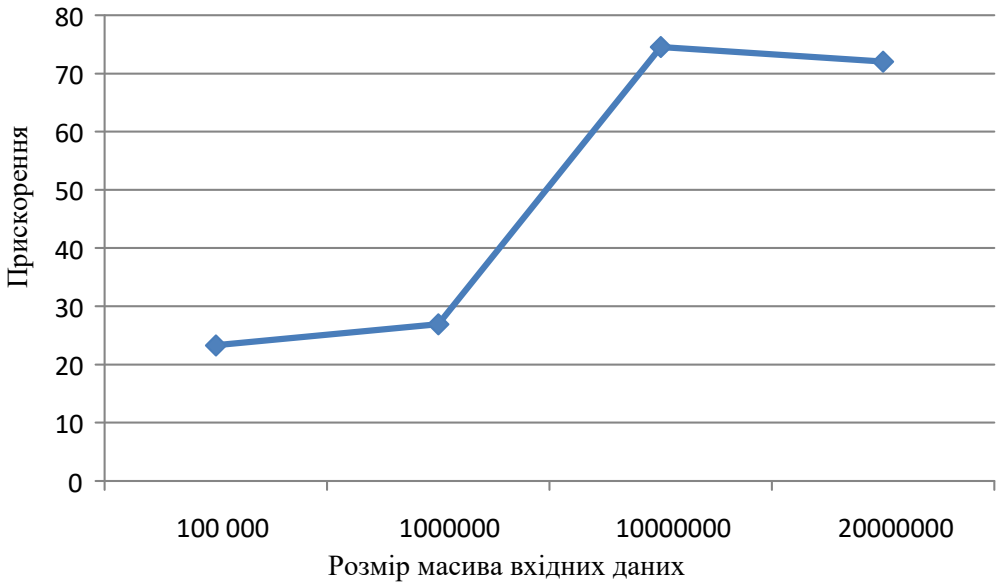


Рисунок 4.6 - Графік прискорення алгоритму

Приміщенням початкових даних в константну пам'ять вдалося добитися ще помітнішого прискорення алгоритму, але при малих об'ємах даних воно також залишається значно меншим, це пов'язано з тим, що копіювання з константної пам'яті займає все одно досить тривалий час.

З попередніх результатів можна зробити висновок, що обчислення на апаратній платформі Nvidia дозволяють добитися колосального прискорення, яке не вдається отримати на центральному процесорі. Порівняння графіків прискорення представлене на рисунку 4.7

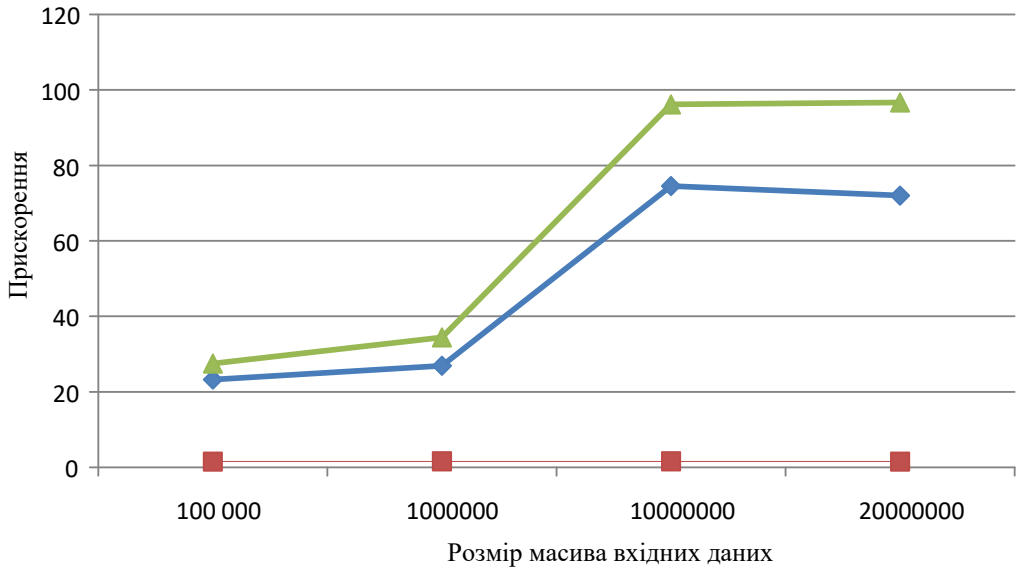


Рисунок 4.7 – Прискорення усіх 3х алгоритмів

4.4 Висновки до четвертого розділу

Сама нижня лінія показує прискорення за допомогою OpenMP, середня з використанням CUDA і найвища з використанням CUDA і оптимізацією зберігання даних.

Результати моделювання наведені в додатку Ж 08-36.МКР.003.00.000 ПЛ6.

5 ЕКОНОМІЧНА ЧАСТИНА

В техніко-економічному обґрунтуванні представленому в першому розділі даної магістерської кваліфікаційної роботи було приблизно обґрунтовано доцільність проведення НДДКР. Тому в даному розділі будуть проведені більш детальні розрахунки витрат на проведення НДДКР стосовно дослідження автономного пристрою на Nvidia для задач штучного інтелекту.

Для економічного розрахунку проведення НДДКР потрібно скласти кошторис витрат, який передбачає розрахунок визначених основних статей витрат.

Таблиця 5.1 – Основна заробітна плата дослідників та розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
1. Керівник проекту	11300,00	513,64	44	22600,00
2. Ст. науковий співробітник	10600,00	481,82	12	5781,82
3. Інженер-метролог	9650,00	438,64	15	6579,55
4. Аналітик	9650,00	438,64	10	4386,36
5. Інженер-радіотехнік 1 кат.	9000,00	409,09	38	15545,45
Разом				54893,18

Основна заробітна плата дослідників та розробників, яка розраховується за формулою [**10**]:

$$З_о = \frac{M}{T_р} \cdot t \quad , \tag{5.1}$$

де M – місячний посадовий оклад конкретного розробника (дослідника), грн.;

$T_р$ – число робочих днів в місяці, 22 дн;

t – число днів роботи розробника (дослідника).

Проведені розрахунки зводимо до таблиці.

Витрати на основну заробітну плату працівників (Зр), що здійснюють підготовку робочих місць необхідних для дослідження автономного пристрою на Nvidia для задач штучного інтелекту, підготовку та формування баз даних, підготовку та монтаж обладнання, макетів, виготовлення дослідних зразків тощо, розраховуються на основі норм часу, які необхідні для виконання даної роботи, за формулою **[10]**:

$$З_р = \sum_{1}^{n} t_i \cdot С_i \cdot К_с \quad , \qquad (5.2)$$

де t_i - норма часу (трудомісткість) на виконання конкретної роботи, годин;

n - число робіт по видах та розрядах;

$К_с$ - коефіцієнт співвідношень, який установлений в даний час Генеральною тарифною угодою між Урядом України і профспілками, $К_с$ = 1,75;

Ci - погодинна тарифна ставка робітника відповідного розряду, який виконує відповідну роботу, грн./год.

$С_i$ визначається за формулою **[10]**:

$$С_i = \frac{М_м \cdot К_i}{Т_р \cdot Т_{зм}}, \qquad (5.3)$$

де, $М_н$ – прожитковий мінімум працездатної особи, грн., $М_н$ = 2197,00 грн.;

$К_i$ - тарифний коефіцієнт робітника відповідного розряду;

$Т_р$ - число робочих днів в місяці, $Т_р$ = 22 дн;

$Т_{зм}$ - тривалість зміни, $Т_{зм}$ = 8 годин.

Проведені розрахунки винесемо до таблиці.

Таблиця 5.2 – Витрати на основну заробітну плату працівників

Найменування робіт	Трудомісткість, нормо-годин	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн.	Величина оплати, грн.
1. Встановлення офісного обладнання	8,0	2	1,1	24,03	192,24
2. Інсталяція програмного забезпечення	8,0	4	1,35	29,49	235,93
3. Компіляція аналогово-цифрових модулів	10,0	5	1,7	37,14	371,37
4. Відлагодження програмно-інтерфейсних модулів	5,0	5	1,7	37,14	185,68
5. Формування радіоелектронної моделі апаратного комплексу	16,0	2	1,1	24,03	384,48
6. Виготовлення макету	9,5	3	1,35	29,49	280,16
Разом					1649,86

Додаткова заробітна плата розробників, дослідників та працівників, які приймали участь в дослідженнях та розробці НДДКР розраховується як 11% від основної заробітної плати розробників та працівників:

$$З_д = З_о \text{ *·} 11 / 100\% \quad (5.4)$$

$$З_д = (54893,18 + 1649,86) \text{ *·} 11 / 100\ \% = 6219,73 \text{ (грн.)}$$

Нарахування на заробітну плату дослідників та працівників.

Згідно діючого законодавства нарахування на заробітну плату складають 22% від суми основної та додаткової заробітної плати:

$$Н_з = (З_о + З_д) \cdot * 22\% / 100\% \quad (5.5)$$

$Н_з$ = (54893,18+ 1649,86+ 6219,73) * 22% / 100% = 13807,81 (грн.)

Витрати на матеріали на даному етапі проведення НДДКР пов'язані з використанням моделей елементів та моделювання роботи і досліджень за допомогою комп'ютерної техніки та створення експериментальних блоків і компонентів, тому дані витрати формуються на основі як офісних витратних матеріалів так і обмеженого переліку матеріалів.

Витрати на матеріали, що були використані при проведенні досліджень, розраховуються по кожному виду матеріалів за формулою **[10]**:

$$M = \sum_{1}^{n} H_i \cdot Ц_i \cdot К_i \quad (5.6)$$

де, - $Н_i$ - витрати матеріалу *і*-го найменування, кг;

$Ц_i$ - вартість матеріалу *і*-го найменування, грн./кг.;

$К_i$ - коефіцієнт транспортних витрат, $К_i$ = 1,1;

n - кількість видів матеріалів,

Проведені розрахунки зводимо до таблиці.

З врахуванням транспортних витрат вартість матеріалів складе

М = 1417,92 * 1,1 = 1559,71 грн.

Таблиця 5.3 – Витрати на основні матеріали

Найменування матеріалу, марка, тип, сорт	Одиниця виміру	Ціна за одиницю, грн.	Витрачено	Вартість витраченого матеріалу, грн.
Папір канцелярський	уп.	92,00	1	92,00
Компакт-диски	шт.	10,10	5	50,50
Канцелярські товари	компл.	136,00	4	544,00
Офісне начиння	комплект	194,00	2	388,00
Тонер для принтера	кг	5998,00	0,02	119,96
Флюс ФКСН	кг	145,00	0,2	29,00
Припій ПОС-61	кг	690,00	0,2	138,00
Спирт	кг	140,00	0,06	8,40
Дріт монтажний	кг	214,50	0,09	19,31
Бензосуміш	кг	35,00	0,08	2,80
Лак	кг	160,00	0,065	10,40
СклотекстолітСФ-2Н-35	кг	86,00	0,14	12,04
Смола поліамідна 68С	кг	58,60	0,06	3,52
Всього				1417,92

Витрати на комплектуючі (основне обладнання, емулятори, моделі, комплектуючі макетів), що були використані при дослідженні автономного пристрою на Nvidia для задач штучного інтелекту, розраховуються за формулою:

$$Н = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i \ , \qquad (5.7)$$

де: *Нi* - кількість комплектуючих *i*-го виду, шт.;

Цi - покупна ціна комплектуючих *i*-го виду, грн.;

$К_i$ - коефіцієнт транспортних витрат, $К_i = 1,10$;

n - кількість видів матеріалів.

Проведені розрахунки зводимо до таблиці.

Таблиця 5.4 – Витрати на комплектуючі для формування компонентів для НДДКР

Найменування комплектуючих	Кількість, шт.	Ціна за штуку, грн.	Сума, грн.
Резистори	0	0,00	0,00
3590S-2-101-100R	2	38,80	77,60
R 0805	8	0,20	1,60
Конденсатори	0	0,00	0,00
ECAP-KM-330mkf - 16v	1	1,50	1,50
TAJP475K016RNJ	3	5,71	17,13
CAP 0805	12	0,32	3,84
Котушки	0	0,00	0,00
SMD 3D16 4.7UH	2	1,60	3,20
Діоди	0	0,00	0,00
BAT54AW	4	0,40	1,60
HS1D	2	0,82	1,64
SS14	1	1,80	1,80
P4SMAJ5.0C	1	6,54	6,54
Кварцовий резонатор	0	0,00	0,00
HC-49S 24	1	2,50	2,50
Транзистори	0	0,00	0,00
IRFP4568	2	8,30	16,60
Мікросхеми	0	0,00	0,00
EP4CE6E22C8N	1	145,00	145,00
CM6800T	2	35,00	70,00
MX25L3206E	1	12,80	12,80
25L3206E SOP8	1	19,50	19,50
Роз’єми	0	0,00	0,00
РП 13-35	1	17,00	17,00
Корпус	1	128,00	128,00
Всього			527,85

Витрати на комплектуючі з урахуванням транспортних витрат складають:

$$H = 527,85 *\cdot 1,10 = 580,64 \text{ (грн.)}$$

Амортизація обладнання для проведення досліджень

Таблиця 5.5 - Величина амортизаційних відрахувань

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, міс.	Величина амортиза-ційних відрахувань, грн
Комп'ютеризована система проекту-вання з графічним виводом інформації	26634,00	5	2	887,80
Вимірювальний комплекс	10255,00	5	2	341,83
Генератор сигналу	6378,00	5	2	212,60
Осцилограф	7262,00	5	2	242,07
Частотомір	6453,00	5	2	215,10
Лабораторія	295000,00	25	2	1966,67
Всього				3866,07

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню можуть бути

розраховані з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (5.8)$$

де $Ц_б$ – балансова вартість обладнання, приміщень тощо, які використовувались для розробки нового технічного рішення, грн.;

$t_{вик}$ – термін використання обладнання, приміщень під час розробки, місяців;

$T_в$ – строк корисного використання обладнання, приміщень тощо, років.

Проведені розрахунки необхідно звести до таблиці.

Витрати на силову електроенергію на проведення досліджень розраховують за формулою **[10]**:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \quad (5.9)$$

де, $В$ – вартість 1 кВт-години електроенергії, $В$ = 2,91 грн./кВт –година;

$П$ – встановлена потужність обладнання, кВт.;

$Ф$ – фактична кількість годин роботи обладнання, годин. ;

$К_п$ – коефіцієнт використання потужності.

Всі проведені розрахунки зведемо до таблиці.

Інші витрати охоплюють: загальновиробничі витрати, адміністративні витрати, витрати на відрядження, матеріали, окремі непередбачені витрати, зв’язок, витрати на інтернет-послуги тощо.

Таблиця 5.6 – Витрати на електроенергію при проведенні досліджень

Найменування обладнання	Кількість годин роботи обладнання, год.	Встановлена потужність, кВт	Коефіцієнт використання потужності	Величина оплати
Комп'ютеризована система проекту-вання з графічним виводом інформації	288	0,52	1	435,80
Вимірювальний комплекс	105	0,32	1	97,78
Генератор сигналу	46	0,15	1	20,08
Осцилограф	46	0,15	1	20,08
Частотомір	46	0,16	1	21,42
Всього				595,15

Інші витрати доцільно приймати як 200...300% від суми основної заробітної плати дослідників та робітників. Величина інших витрат складе:

$$I = (54893,18 + 1649,86) * 200\% / 100\% = 113086,08 \text{ (грн.)}$$

Загальні витрати на проведення науково-дослідної роботи.

Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$$B = 54893,18+1649,86+6219,73+13807,81+1417,92+580,64+3866,07+595,15+ +113086,08=196116,43 \text{ (грн.)}$$

Загальна (повна) вартість всієї НДДКР визначається за формулою:

$$B_{заг} = \frac{B}{\alpha}, \tag{5.10}$$

де α - частка витрат, які безпосередньо здійснює виконавець даної НДДКР, у відносних одиницях.

$$B_{заг} = \frac{B}{\alpha} = \frac{196116,43}{0,95} = 206438,35, \text{ грн.}$$

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

$$ЗВ = \frac{B_{заг}}{\beta}, \tag{5.11}$$

де β - коефіцієнт, який характеризує етап (стадію) виконання даної НДДКР (від 0,1… до 0,9).

$$ЗВ = \frac{B_{заг}}{\beta} = \frac{206438,35}{0,9} = 229375,95, \text{ грн.}$$

5.1 Прогнозування комерційних ефектів від реалізації результатів розробки

В умовах ринку узагальнюючим позитивним результатом, що його отримує підприємство (організація) від впровадження результатів тієї чи іншої розробки, є збільшення чистого прибутку підприємства (організації). Зростання чистого прибутку ми можемо оцінити у теперішній вартості грошей.

Саме зростання чистого прибутку забезпечить підприємству (організації) надходження додаткових коштів, які дозволять покращити

фінансові результати діяльності та виплатити кредити (якщо вони потрібні для впровадження результатів розробки).

При проведенні даної розробки не можливо прямо оцінити зростання чистого прибутку підприємства від впровадження результатів наукової розробки. У цьому випадку збільшення чистого прибутку підприємства для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum (\Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\upsilon}{100}) \qquad (5.12)$$

де $\Delta Ц_o$ - покращення основного оціночного показника від впровадження результатів розробки у даному році. Зазвичай таким показником може бути ціна одиниці нової розробки;

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

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

$Ц_o$ - основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

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

λ - коефіцієнт, який враховує сплату податку на додану вартість. У 2020 р. ставка податку на додану вартість дорівнює 20%, а коефіцієнт λ = 0,8333.

ρ - коефіцієнт, який враховує рентабельність продукту. Рекомендується приймати р = 0,2...0,3;

υ - ставка податку на прибуток. У 2020 році υ =18%.

В результаті впровадження результатів наукової розробки покращується якість нашої розробки, що дозволяє підвищити ціну її реалізації на 175 грн.

Кількість одиниць реалізованої продукції також збільшиться: протягом першого року - на 100 шт., протягом другого року - ще на 300 шт., протягом третього року - ще на 400 шт., а протягом четвертого року – на 200 шт. Орієнтовно: реалізація аналогічного пристрою до впровадження результатів наукової розробки складала 3000 шт., а її ціна - 2800 грн.

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

Збільшення чистого прибутку підприємства протягом першого року складе:

$$\Delta П_1 = \left[175 \cdot 3000 + (2800 + 175) \cdot 100\right] \cdot 0,8333 \cdot 0,25 \cdot (1 - \frac{18}{100}) = 140505,00 \text{ грн.}$$

Збільшення чистого прибутку підприємства протягом другого року (відносно базового року, тобто року до впровадження результатів наукової розробки) складе:

$$\Delta П_2 = \left[175 \cdot 3000 + (2800 + 175) \cdot (100 + 300)\right] \cdot 0,8333 \cdot 0,25 \cdot (1 - \frac{18}{100}) = 292967,00$$ грн.

Збільшення чистого прибутку підприємства протягом третього року (відносно базового року, тобто року до впровадження результатів наукової розробки) складе:

$$\Delta П_3 = \left[175 \cdot 3000 + (2800 + 175) \cdot (100 + 300 + 400)\right] \cdot 0,8333 \cdot 0,25 \cdot (1 - \frac{18}{100}) = 496251,00$$ грн.

Збільшення чистого прибутку підприємства протягом четвертого року (відносно базового року, тобто року до впровадження результатів наукової розробки) складе:

$$\Delta П_4 = \left[175 \cdot 3000 + (2800 + 175) \cdot (100 + 300 + 400) + 200\right] \cdot 0,8333 \cdot 0,25 \cdot \cdot (1 - \frac{18}{100}) = 597893,00$$ грн.

5.2 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Основними показниками, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахунок ефективності вкладених інвестицій передбачає проведення таких робіт:

1. Розраховують теперішню вартість інвестицій, що вкладаються в наукову розробку. Такою вартістю ми можемо вважати прогнозовану величину загальних витрат ЗВ=**229375,00** грн. на виконання та впровадження результатів НДДКР.

2. Розраховують очікуване збільшення прибутку, що його отримає підприємство (організація) від впровадження результатів наукової розробки, для кожного із років, починаючи з першого року впровадження.

3. Для спрощення подальших розрахунків будують вісь часу, на яку наносять всі платежі (інвестиції та прибутки), що мають місце під час виконання науково-дослідної роботи та впровадження її результатів.

Платежі показуються у ті терміни, коли вони здійснюються.

Проведемо відповідні розрахунки.

У першому році підприємство отримає збільшення чистого прибутку на **140505,00** грн відносно базового року.

У другому році - збільшення чистого прибутку на **292967,00** грн (відносно базового року).

У третьому році - збільшення чистого прибутку на **496251,00** грн (відносно базового року),

У четвертому - на **597893,00** грн (відносно базового року).

Тоді рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, наведений на рис.

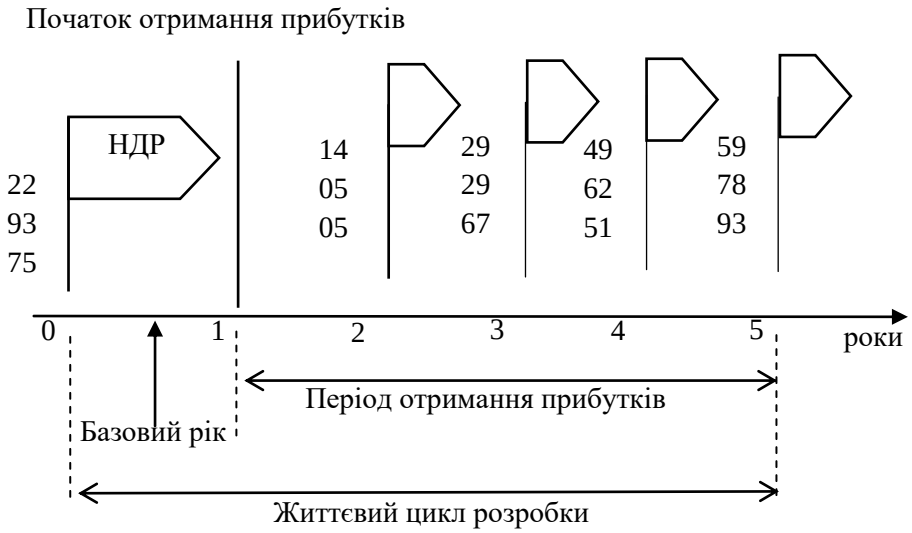


Рисунок 5.1 - Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

4. Розраховують абсолютну ефективність вкладених інвестицій $E_{абс}$. Для цього використаємо формулу [10]:

$$E_{абс} = (ПП - PV), \tag{5.13}$$

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

PV - теперішня вартість інвестицій *PV = ЗВ=229375,00* грн.

У свою чергу, приведена вартість всіх чистих прибутків *ПП* розраховується за формулою:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.14)$$

де *ΔП* - збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

T - період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ - ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,12;

t - період часу (в роках) від моменту отримання чистого прибутку до точки „0".

$$ПП = \frac{140505}{(1+0,2)^2} + \frac{292967}{(1+0,2)^3} + \frac{496251}{(1+0,2)^4} + \frac{597893}{(1+0,2)^5} = \text{ грн.}$$
$$= 112009 + 208528 + 315376 + 339260 = 975174,00$$

Розрахуємо абсолютну ефективність інвестицій, вкладених у реалізацію проекту. Отримаємо:

$$E_{абс} = (975175,00 – 229376,00) = 745799,00 \text{ грн.}$$

Оскільки $E_{абс} > 0$, то вкладання коштів на виконання та впроваджен¬ня результатів НДДКР може бути доцільним.

Результат від проведення наукових досліджень та їх впровадження принесе прибуток, але це також ще не свідчить про те, що інвестор буде зацікавлений у фінансуванні даного проекту (роботи).

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.15}$$

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

PV - теперішня вартість інвестицій $PV = ЗВ$, грн;

T - життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[T]{1 + \frac{E_{абс}}{PV}} - 1 = \sqrt[5]{1 + \frac{745799}{229376}} - 1 = 0,34$$

Розраховану величину $E_в$ порівнюємо з мінімальною ставкою дисконтування, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування визначається за формулою:

$$\tau = d + f, \tag{5.16}$$

де d - середньозважена ставка за депозитними операціями в комерцій¬них банках; в 2020 році в Україні (0,08...0,12);

f - показник, що характеризує ризикованість вкладень (0,05...0,1).

$$\tau = d + f = 0,12 + 0,05 = 0,17.$$

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій.

Термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ можна розрахувати за формулою:

$$T_{ок} = \frac{1}{E_в}.$$

$$T_{ок} = \frac{1}{E_в} = \frac{1}{0,34} = 2,98 \text{ року}$$

Якщо $T_{ок}$ < 3...5-ти років, то фінансування даної наукової розробки в принципі є доцільним.

6 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Мінімізація імовірності виникнення виробничого травматизму та захворювань при забезпеченні регламентованих параметрів умов праці є головним завданням охорони праці.

У даному розділі наводиться розгляд небезпечних, шкідливих та уражаючих для людини та навколишнього середовища чинників, які утворюються при проведенні розробки автономного пристрою на Nvidia для задач штучного інтелекту. Тут висвітлюються, зокрема, технічні рішення з гігієни праці та виробничої санітарії, визначення попереднє КПО для однобічного природного освітлення, технічні рішення з промислової та пожежної безпеки під час проведення розробки, безпека в надзвичайних ситуаціях.

Під час розробки вказаного пристрою на працівників вливають ті чи інші небезпечні та шкідливі виробничі фактори (НШВФ) фізичної та психофізіологічної груп згідно [21].

Фізичні небезпечні і шкідливі виробничі фактори: понижена або підвищена температура повітря робочої зони, підвищений рівень шуму на робочому місці, підвищений рівень статичної електрики, недостатність або відсутність природного освітлення, недостатня освітленість робочої зони, відбита або пряма блискучість, підвищена яскравість світла.

Психофізіологічні НШВФ: нервово-психічні перевантаження: розумове перенапруження, монотонність праці, перенапруження аналізаторів.

6.1 Гігієна праці та виробнича санітарія

6.1.1 Склад повітря робочої зони та мікроклімат

Визначаємо для приміщення, де проводяться роботи з розробки

автономного пристрою на Nvidia для задач штучного інтелекту, категорію важкості робіт за фізичним навантаженням – легка Іа.

Відповідно до [22] допустимі показники мікроклімату у робочій зоні для теплого та холодного періодів року приведені у таблиці 6.1 [22].

Таблиця 6.1 – Допустимі показники мікроклімату в приміщенні

Період року	Категорія робіт	Температура повітря, °С для робочих місць		Відносна вологість повітря, %	Швидкість руху повітря, м/с
		постійних	непостійних		
Холодний	Іа	21-25	18-26	75	≤0,1
Теплий	Іа	22-28	20-30	55 при 28°С	0,1-0,2

Розкид значень температури повітря вздовж висоти робочої зони допускається до 3°С. При опроміненні менше 25% поверхні тіла людини, нормована інтенсивність теплового опромінення сладає 100 Вт/м^2.

Повітря робочої зони не повинно містити шкідливих речовин з концентраціями вище гранично допустимих концентрацій (ГДК), які використовуються при проектуванні виробничих приміщень (будівель), обладнання, технологічних процесів, вентиляцій, з метою контролю за якістю виробничого середовища. ГДК шкідливих речовин, що використовуються в даному виробничому приміщені наведено в таблиці 6.2.

Таблиця 6.2 – Гранично допустимі концентрації шкідливих речовин у повітрі робочої зони

Назва речовини	Параметр, що нормується	Значення	Клас небезпеки
Пил нетоксичний	ГДК, мг/м^3	0,15	4
Іони n^+, n^-	число іонів в 1 см^3 повітря	50000	–

З метою встановлення необхідних за нормативами показників

мікроклімату і чистоти повітря робочої зони запропоновано такі заходи:

1) у приміщенні повинна бути встановлена система кондиціонування для теплого і опалення для холодного періодів року;

2) з метою підвищення вологості повітря потрібно використовувати зволожувачі або розташовувати місткості з водою за типом акваріумів поблизу опалювальних приладів;

3) застосування витяжної вентиляції, яка видаляє забруднення або нагріте повітря з приміщення, а також за допомогою неї контролюється швидкість руху повітря і вологість.

6.1.2 Виробниче освітлення

З метою створення гігієнічних раціональних умов на робочих місцях великі вимоги пред'являються до якісних та кількісних параметрів освітлення.

З точки зору задач зорової роботи в приміщенні, де проводяться роботи з розробки автономного пристрою на Nvidia для задач штучного інтелекту, згідно [23] визначаємо, що вони відносяться до III розряду зорових робіт. Вибираємо контраст об'єкта з фоном – великий, а характеристику фону – середню, яким відповідає підрозряд зорових робіт *г*.

Нормативні значення коефіцієнта природного освітлення (КПО) та мінімальні значення освітленості для штучного освітлення приведені в [22].

Оскільки приміщення розташоване в м. Вінниця (друга група забезпеченості природним світлом), а вікна розташовані за азимутом 135°, то для таких умов КЕО визначатиметься за формулою [23, 24]

$$e_N = e_н m_N \ [\%], \tag{6.1}$$

де $e_н$ – табличне значення КЕО, %;

m_N – коефіцієнт світлового клімату;

N – номер групи забезпеченості природним світлом.

За відомими значеннями отримаємо нормовані значення КПО для бокового та суміщеного освітлення:

$$e_{N.б} = 2 \cdot 0{,}85 = 1{,}7\ (\%);$$

$$e_{N.с} = 1{,}2 \cdot 0{,}85 = 1{,}02\ (\%).$$

Для встановлення нормативних значень параметрів освітлення запропоновано:

1) при недостатньому природному освітлені в світлу пору доби доповнення штучним за допомогою газорозрядних ламп з утворенням системи суміщеного освітлення;

2) застосування загального штучного освітлення у темну пору доби. Попередньо визначимо КПО для однобічного природного освітлення, якщо Розміри приміщення (м): 12 × 6 × 3,3. Освітлення здійснюється симетрично розташованими вікнами, розміри яких (м): 2,2 × 1,8. Висота від підлоги до підвіконня – 0,8 м. Остіклення подвійне, плетіння металеві. Конфронтуючі будинки відсутні.

Природне освітлення забезпечується необхідними архітектурно-будівельними рішеннями – положенням світлових пройм в стінах.

Розрахункове значення коефіцієнта природної освітленості визначається за формулою [23]:

$$e_{П} = \frac{nS_{В}\tau_{З}r_{1}100}{K_{З}\eta_{В}S_{П}K_{БУД}}\ [\%], \qquad (6.2)$$

де n – кількість вікон;

$S_{В}$, $S_{П}$ – площа вікна та підлоги відповідно, м^2;

$\tau_{З}$ – загальний коефіцієнт світлопропускання;

r_1 – коефіцієнт, що враховує підвищення КПО при комбінованому освітлені

завдяки світлу, яке відбивається від поверхонь приміщень;

$K_З$ – коефіцієнт запасу (для виробничих приміщень $K_З$ = 1,3...1,5);

$\eta_В$ – світлова характеристика вікон;

$K_{БУД}$ – коефіцієнт, що враховує затінення вікон будівлями, які розташовані напроти.

Кількість вікон n = 2.

Площу кожного вікна визначимо за формулою

$$S_В = H_В B_В \text{ [м}^2\text{]}, \tag{6.3}$$

де $H_В$, $B_В$ – висота та ширина вікна відповідно, м.

Площа стелі рівна площі підлоги і визначається за формулою

$$S_{стелі} = S_П = L_П B_П \text{ [м}^2\text{]}, \tag{6.4}$$

де $L_П$, $B_П$ – довжина та ширина підлоги відповідно, м.

Визначимо загальний коефіцієнт світлопропускання за формулою:

$$\tau_З = \tau_1 \tau_2 \tau_3 \tau_4 \tau_5, \tag{6.5}$$

де τ_1 – коефіцієнт світлопропускання матеріалу;

τ_2 – коефіцієнт, що враховує втрати світла у віконній рамі;

τ_3 – коефіцієнт, що враховує втрати світла у несучих конструкціях (при боковому освітленні τ_3 = 1; при верхньому – τ_3 = 0,8-0,9);

τ_4 – коефіцієнт, що враховує втрати світла у сонцезахисних пристроях;

τ_5 – коефіцієнт, що враховує втрати світла у захисній сітці, яка встановлюється під ліхтарями (при суміщеному освітленні приймається рівним 0,9; при природному 1).

Для одинарного остіклення вибираємо τ_1 = 0,9. Для дерев’яного виду

віконних рам $\tau_2 = 0{,}75$. Для бокового освітлення приймаємо $\tau_3 = 1$. Оскільки сонцезахисні пристрої не використовуються, то приймаємо $\tau_4 = 1$. Для природного освітлення приймаємо $\tau_5 = 1$.

Після підстановки відомих значень у формули (6.3, …, 6.5) отримаємо

$$S_B = 2{,}2 \cdot 1{,}8 = 4 \text{ (м}^2\text{)};$$

$$S_{стелі} = S_П = 12 \cdot 6 = 72 \text{ (м}^2\text{)};$$

$$\tau_З = 0{,}9 \cdot 0{,}75 \cdot 1 \cdot 1 \cdot 1 = 0{,}68.$$

Приймаємо коефіцієнт запасу $K_З = 1{,}3...1{,}5 = 1{,}4$.

Для визначення коефіцієнту r_1 необхідно знайти середній коефіцієнт відбиття приміщення за формулою:

$$\rho_{СР} = \frac{\rho_{стелі} S_{стелі} + \rho_{стін} S_{стін} + \rho_П S_П}{S_{стелі} + S_{стін} + S_П}, \quad (6.6)$$

де $\rho_{стелі}$, $\rho_{стін}$, $\rho_п$ – коефіцієнти відбиття стелі, стін та підлоги відповідно; $S_{стелі}$, $S_{стін}$, $S_п$ – площа стелі, стін, та підлоги відповідно, м^2. Приймаємо $\rho_{стелі} = 0{,}725$; $\rho_{стін} = 0{,}7$; $\rho_п = 0{,}25$.

Площу стін визначимо за формулою

$$S_{стін} = H_{стіни}(2L_П + 2B_П) \text{ [м}^2\text{]}, \quad (6.7)$$

де

$H_{стіни}$ – висота стіни, м.

Після підстановки відомих значень у формули (6.7, 6.6) отримаємо

$$S_{стін} = 3{,}3 \cdot (2 \cdot 12 + 2 \cdot 6) = 118{,}8\ (м^2);$$

$$\rho_{ср} = \frac{0{,}725 \cdot 72 + 0{,}7 \cdot 118{,}8 + 0{,}25 \cdot 72}{72 + 118{,}8 + 72} = 0{,}5836.$$

Для визначення коефіцієнту r_1 необхідно також визначити співвідношення

$$B_n / h;\ l / B_n;\ L_n / B_n, \qquad (6.8)$$

де h – висота від рівня умовної робочої поверхні до верхнього краю вікна, м;
l – відстань розрахункової точки до зовнішньої стіни, м.

Знайдемо висоту від рівня умовної робочої поверхні до верхнього краю вікна за формулою:

$$h = H_{стіни} - h_р - (H_{стіни} - H_В - h_П) = H_В + h_П - h_р\ [м], \qquad (6.9)$$

де $h_р$ = 0,8 м – висота робочої поверхні.

Розрахункову точку приймаємо на відстані 1 м від стіни, протилежної від вікна

$$l = B_n - 1\ [м]. \qquad (6.10)$$

Після підстановки відомих значень у формули (6.10, 6.9) отримаємо

$$l = 6 - 1 = 5\ (м);$$

$$h = 2{,}2 + 0{,}8 - 0{,}8 = 2{,}2\ (м).$$

Отже, співвідношення, необхідні для визначення коефіцієнту r_1 дорівнюють

$B_n / h = 6 / 2{,}2 = 2{,}73$; $l / B_n = 5 / 6 = 0{,}83$; $L_n / B_n = 12 / 6 = 2$.

За отриманими значеннями і величиною $\rho_{ср}$ вибираємо коефіцієнт r_1 = 7,65. Світлову характеристику вікон вибираємо за значеннями співвідношень L_n / B_n; B_n / h, для яких η_B = 11,4.

Оскільки конфронтуючі будинки відсутні, то $K_{БУД}$ = 1.

Отже, розрахункове значення коефіцієнта природної освітленості становить

$$e_n = \frac{2 \cdot 4 \cdot 0{,}68 \cdot 7{,}65 \cdot 100}{1{,}4 \cdot 11{,}4 \cdot 72 \cdot 1} = 3{,}622\ (\%).$$

Оскільки $e_П$ = 3,622 % > e_N = 1,7 %, то природна освітленість в даному приміщенні є достатньою.

6.1.3 Виробничі віброакустичні коливання

Зважаючи на те, що при використанні пристроїв крім усього іншого устаткування використовується обладнання, робота якого генерує шум та вібрацію, необхідно передбачити захист від шуму та вібрації.

Встановлено, що приміщення, де відбувається робота з розробки автономного пристрою на Nvidia для задач штучного інтелекту може мати робочі місця із шумом та вібрацією, що поширюється від сусідніх промислових приміщень.

Для запобігання травмуванню працюючих під дією шуму та вібрації вони підпадає під нормування. Основним нормативом стосовно виробничого шуму, що діє на території нашої країни, є [25], у відповідності з яким нормовані рівні звукового тиску, рівні звуку та еквівалентні рівні шуму на робочих місцях в промислових приміщеннях не повинні перевищувати

значень, що приведені в таблиці 6.4. Норми виробничих вібрацій наведені в таблиці 6.5 для 3-ї категорії (технологічна) типу "в".

Таблиця 6.4 – Допустимі рівні звукового тиску та еквівалентні рівні звуку

Рівні звукового тиску в дБ в октавних полосах з середньо-геометричними частотами, Гц									Рівні звуку та еквівалентні рівні звуку, дБА
31,5	63	125	250	500	1000	2000	4000	8000	
86	71	61	54	49	45	42	40	38	50

З метою забезпечення допустимих параметрів віброакустичних коливань у приміщенні запропоновано:

1) своєчасне проведення профілактичного ремонту;

2) застосування в конструкціях обладнання акустичних екранів та звуко- та віброізоляційних кожухів.

Таблиця 6.5 – Допустимі рівні віброприскорення [26]

Гранично допустимі рівні віброприскорення, дБ, в октавних полосах з середньо-геометричними частотами, Гц						Коректовані рівні віброприскорення, дБА
2	4	8	16	31,5	63	
36	33	33	39	45	51	33

6.1.4 Виробничі випромінювання

Аналіз умов праці показав, що приміщення, де виконується робота з розробки автономного пристрою на Nvidia для задач штучного інтелекту може містити електромагнітні випромінювання.

Гранично допустимі рівні електромагнітних полів показані у таблиці 6.6.

Таблиця 6.6 – Гранично допустимі рівні електромагнітних полів (безперервне випромінювання, амплітудна чи кутова модуляція)

Номер діапазону	Метричний розподіл діапазонів	Частоти	Довжина хвиль, λ	ГДР, В/м
5	Кілометрові хвилі (низькі частоти, НЧ)	30-300 кГц	10-1 км	25
6	Гептаметрові хвилі (середні частоти, СЧ)	0,3-3 МГц	1-0,1 км	15
7	Декаметрові хвилі (високі частоти, ВЧ)	3-30 МГц	100-10 м	3·lg λ
8	Метрові хвилі (дуже високі частоти, ДВЧ)	30-300 МГц	10-1 м	3

Для забезпечення захисту та досягнення нормативних рівнів випромінювань необхідно використовувати екранування робочого місця і скорочення часу опромінення за рахунок перерв на відпочинок.

Технічні рішення щодо промислової та пожежної безпеки під час проведення розробки автономного пристрою на Nvidia для задач штучного інтелекту

6.2 Безпека щодо організації робочих місць

Конструкція робочого місця, взаємне розташування його елементів і його розміри повинні відповідати антропометричним, фізіологічним та психофізіологічним характеристикам людини, а також характеру роботи [27].

Площа одного робочого місця повинна складати не менше 6,0 м^2, об'єм приміщення – не менше як 20 м^3, висота – не менше 3,2 м [8].

Кольорове оздоблення інтер’єру приміщення повинно відповідати вказівкам з проектування кольорової обробки інтер’єрів приміщень будівель промислових підприємств. Поверхня підлоги повинна бути гладкою, без

вибоїн, не слизькою, мати антистатичні властивості, зручною для вологого прибирання. Забороняється застосовувати під час оздоблення інтер'єру полімери, що виділяють у повітря шкідливі хімічні речовини.

6.3 Електробезпека

Основними причинами ураження електричним струмом в даному приміщенні можуть бути: робота під напругою під час проведення ремонтних робіт, несправність устаткування, випадковий дотик до струмоведучих частин чи металевих частин, які опинилися під напругою. У відповідності до [29] дане приміщення належить до приміщень з підвищеною небезпекою ураження електричним струмом через наявність високої (понад 75 %) вологості. Тому безпека використання електрообладнання має забезпечуватись комплексом заходів, які включають використання ізоляції струмоведучих елементів, захисних блокувань, захисного заземлення та ін [30].

6.4 Пожежна безпека

Згідно [31] приміщення, де проводиться робота з розробки автономного пристрою на Nvidia для задач штучного інтелекту, відноситься до категорії пожежної небезпеки Б. Дане приміщення відноситься до 2-го ступеня вогнестійкості, в якому приміщення знаходяться в будівлі з несучими та огороджувальними конструкціями з природних або штучних кам’яних матеріалів, бетону, залізобетону із застосуванням листових і плитних негорючих матеріалів.

Мінімальні межі вогнестійкості будівельних конструкцій розглядуваного приміщення наведені в таблиці 6.7. В таблиці 6.8 приведено протипожежні норми проектування будівель і споруд.

Таблиця 6.7 – Мінімальні межі вогнестійкості приміщення [11]

Ступінь вогнестійкості будівлі	Стіни				Колони	Східчасті майданчики	Плити та інші несучі конструкції	Елементи покриття	
	Несучі та східчасті клітки	Самонесучі	Зовнішні несучі	Перегородки				Плити, прогони	Балки, ферми
2	REI 120 M0	REI 60 M0	E 15 M0	EI 15 M0	R 120 M0	R 60 M0	REI 45 M0	REI 15 M0	R 30 M0

Примітка. R – втрати несучої здатності; E – втрати цілісності; I – втрати теплоізолювальної спроможності; M – показник здатності будівельної конструкції поширювати вогонь (межа поширення вогню); M0 – межа поширення вогню дорівнює 0 см.

Таблиця 6.8 – Протипожежні норми проектування будівель і споруд [33]

Об’єм приміщення, тис. м^3	Категорія пожежної небезпеки	Ступінь вогнестійкості	Відстань, м, при щільності людського потоку в загальному проході, осіб/м^2			Кількість людей на 1 м ширини евакуаційного виходу	Протипожежні розриви, м, для ступеня їх вогнестійкості			Найбільша кількість поверхів	Максимально допустима площа поверху, м^2, для кількості поверхів		
			до 1	2-3	4-5		I,II	III	IV,V		1	2	3 і більше
до 15	Б	2	40	25	15	45	9	9	12	6	н.о.	–	–

Примітки: н.о. – не обмежується.

Вибираємо, що приміщення, в якому проводиться робота з розробки, має бути обладнане двома вогнегасниками, пожежним щитом, ємністю з піском [12].

6.5 Безпека у надзвичайних ситуаціях.

Дослідження стійкості роботи автономного пристрою на Nvidia для задач штучного інтелекту в умовах дії загрозливих чинників надзвичайних ситуацій

Забезпечення стійкості роботи блоків і систем та пристроїв для задач штучного інтелекту у НС базується на комплексі організаційних, інженерно-технічних заходів і засобів, спрямованих на збереження працездатності в умовах дії загрозливих чинників. Для цього необхідно: прогнозувати та оцінити можливі наслідки; заздалегідь спланувати заходи із запобігання та зменшення вірогідності виникнення НС і скорочення масштабів прояву результатів НС; організація робіт в умовах НС та ліквідація її наслідків.

Розроблюваний автономний пристрій на Nvidia призначено для задач штучного інтелекту, для обробки і передачі інформації також під час надзвичайних ситуацій, у воєнний та надзвичайний стан. Також, такі пристрої являють собою сукупність організаційних і технічних засобів для обробки інформації з метою забезпечення інформаційних потреб користувачів.

Одним з загрозливих чинників для автономного пристрою на Nvidia є електромагнітний імпульс (EMI). Уражаюча дія EMI в приземній області й на землі пов'язана з акумулюванням його енергії довгими металевими предметами, рамними і каркасними конструкціями, антенами, лініями електропередачі та зв'язку, в них виникають сильні наведені струми, які руйнують підключене електронне та інше чутливе устаткування. У районі дії EMI безпосередній контакт людини зі струмопровідними предметами теж є небезпечний.

Безвідмовність автономного пристрою на Nvidia зв’язку– це властивість зберігати працездатність при її використанні в процесі передачі даних. Поряд з цим поява відмов автономного пристрою на Nvidia пов’язана з тим, що в деякі моменти часу роботи може виникнути відмова деяких

елементів пристрої обмеженого доступу в наслідок порушення дієздатності апаратної частини пристрої зв’язку. Дані порушення можуть виникнути при нестійкому живленні, що виникає в наслідок дії електромагнітного імпульсу, іонізуючого випромінювання та інших факторів надзвичайних ситуацій.

Як наслідок, автономний пристрій на Nvidia для задач штучного інтелекту може бути використаний у якості пристрої збору інформації для швидкого прийняття рішень у НС, тому необхідно дослідити вплив загрозливих чинників на роботу та розробити заходи, які сприятимуть підвищенню стійкості його роботи.

6.5.1 Дослідження стійкості роботи автономного пристрою на Nvidia для задач штучного інтелекту в умовах дії іонізуючих випромінювань

Мережа передачі даних поєднує програмну та апаратну частину засобів зв’язку, тому загрозливі чинники є досить різноманітними. Загрозливі чинники, що впливають на безпеку роботи автономного пристрою на Nvidia здебільшого можна віднести до чинників техногенного та воєнного характеру.

Визначаємо граничні значення дози опромінення $Д_{грі}$, для елементної бази елементів пристрою, при яких виникають незворотні зміни. Отримані дані заносимо в таблицю 6.5.

Проаналізувавши дані таблиці 6.5, визначили, що самим уразливим елементом блоків автономного пристрою на Nvidia з мінімальною дозою $Д_{грі} = 10^4 Р$ є такі як транзистори та діоди. Визначаємо можливу дозу опромінення за формулою:

$$Д_м = \frac{2 \cdot P_1 \left(\sqrt{t_к} - \sqrt{t_п} \right)}{K_{осл}}, \qquad (6.11)$$

де P_1 – максимальне значення рівня радіації (5,47 Р/год);

t_k – час кінця опромінення ($t_k = 43800$ год);

$t_п$ – час початку опромінення (1 год).

$К_{осл}$ – коефіцієнт послаблення радіації ($К_{осл}$=2).

Таблиця 6.5 – Граничні значення експозиційних доз елементів пристрою.

Блоки автономного пристрою на Nvidia	Елементи блоків	$Д_{грі}$, Р	$Д_{гр}$, Р
Блок перетворення сигналів	Мікросхема LM317	10^5	10^4
	Діод 1N4004	10^4	
	Резистор С2-23	10^7	
Блок формування сигналу і програмування	Конденсатор К50-6	10^4	
	Діод 1N4148	10^4	
	Резистор С2-33	10^7	
	Транзистор 2SA1271	10^4	
Блок обробки сигналу	Резистор С2-29В	10^8	
	Транзистор 2SA1271	10^4	

$$Д_М = \frac{2 \cdot 5,47 \cdot (\sqrt{40000} - \sqrt{1})}{2} = 1273,6 \text{ (Р).}$$

Оскільки $Д_{грі} > Д_м$, то даний пристрій стійкий до дії радіації. Визначимо допустимий час роботи автономного пристрою на Nvidia в заданих умовах за формулою:

$$t_д = \frac{Д_{гр} \cdot К_{осл} + 2 \cdot Р_1 \cdot \sqrt{1}}{2 \cdot Р_1}, \qquad (6.12)$$

$$t_д = \left(\frac{10^4 \cdot 2 + 2 \cdot 5,47\sqrt{1}}{2 \cdot 5,47} \right)^2 = 23127 \text{ (год).}$$

Отже, можлива доза опромінення елементної бази $Д_м$=1273,6 Р, а допустима - 10^4 Р. Отже, автономний пристрій на Nvidia для задач штучного інтелекту буде стійкою в умовах дії іонізуючого випромінювання. Допустимий час роботи автономного пристрою на Nvidia в заданих умовах становить 23127 год., при рівні радіації 5,47 Р/год.

6.5.2 Дослідження стійкості роботи автономного пристрою на Nvidia для задач штучного інтелекту в умовах дії електромагнітного імпульсу

Початковими даними є:

1) Вертикальна складова напруженості електричного поля: $E_в$=12 кВ/м

2) $U_{жі}$=48 В

1. Визначимо горизонтальну складову напруженості електричного поля $E_г=10^{-3}\cdot E_в=12\cdot 10^3\cdot 10^{-3}$=12 В/м

2. Автономний пристрій на Nvidia для задач штучного інтелекту розподіляється на окремі функціональні дільниці. На кожній дільниці визначається максимальна довжина струмопровідних частин. Максимальна довжина струмопровідної частини:

1) $l_в$ = 0,6(м); $l_г$ = 1,3(м);

2) $l_в$ = 0,6(м); $l_г$ = 0,9(м);

3) $l_в$ = 0,8(м); $l_г$ = 0,7(м).

3. Визначимо напругу наведення у вертикальній (горизонтальній) струмопровідній частині:

1) $U_в = E_г l_в = 12\cdot 0,6 = 7,2$ *(В);* $U_г = E_в l_г = 12\cdot 10^3\cdot 1,3 = 15,6\cdot 10^3$*(В);*

2) $U_в = E_г l_в = 12\cdot 0,6 = 7,2$ *(В);* $U_г = E_в l_г = 12\cdot 10^3\cdot 0,9 = 10,8\cdot 10^3$*(В);*

3) $U_в = E_г l_в = 12\cdot 0,8 = 9,6$ *(В);* $U_г = E_в l_г = 12\cdot 10^3\cdot 0,7 = 8,4\cdot 10^3$*(В).*

4. Визначимо допустиме коливання напруги живлення:

$U_д = U_ж + (U_ж/100)\cdot 5 = 48+2,4=50,4$ (В).

5. Визначимо коефіцієнти безпеки:

1) $К_{бв} = 20 \cdot lg(U_д/U_в) = 20 \cdot lg(50,4 /7,2) = 16,9$ (дБ);

$К_{бг} = 20 \cdot lg(U_д/U_г) = 20 \cdot lg(50,4 /15,6 \cdot 10^3) = -49,8$ (дБ);

2) $К_{бв} = 20 \cdot lg(U_д/U_в) = 20 \cdot lg(50,4 /7,2) = 16,9$ (дБ);

$К_{бг} = 20 \cdot lg(U_д/U_г) = 20 \cdot lg(50,4 /10,8 \cdot 10^3) = -46,6$ (дБ);

3) $К_{бв} = 20 \cdot lg(U_д/U_в) = 20 \cdot lg(50,4 /9,6) = 14,4$ (дБ);

$К_{бг} = 20 \cdot lg(U_д/U_г) = 20 \cdot lg(50,4 /8,4 \cdot 10^3) = -44,4$ (дБ).

Зведемо усе до таблиці 5.6.

Таблиця 6.6 – Коефіцієнти безпеки блоків автономного пристрою на Nvidia

Елементи пристрою	$l_в$,м	$l_г$,м	$U_в$,В	$U_г$,В	$К_{бв}$,дБ	$К_{бг}$,дБ	Результат дії
Блок перетворення сигналів	0,6	1,3	7,2	15600	16,9	-49,8	нестійкий
Блок формування сигналу	0,6	0,9	7,2	10800	16,9	-46,6	нестійкий
Блок обробки сигналу	0,8	0,7	9,6	8400	14,4	-44,4	нестійкий

6. Звідси можна побачити, що апаратура буде нестійка в роботі, тому що і $К_{бв}$ і $К_{бг}$ менше 40 дБ.

Заходи по підвищенню стійкості: екранування апаратури стальним,, або алюмінієвим екраном.

6.6. Розробка заходів по підвищенню стійкості роботи автономного пристрою на Nvidia в умовах надзвичайних ситуацій.

Головне завдання захисних пристроїв від EMI - виключити доступ наведених струмів до чутливих вузлів і елементів обладнання. У кожному конкретному випадку повинні бути знайдені найбільш ефективні і економічно доцільні методи захисту електронної апаратури і великих

розгалужених телекомунікаційних систем. Розглянемо основні методи захисту:

1. Металеві екрани відбивають електромагнітні хвилі і гасять високочастотну енергію. Через систему заземлення струм, наведений EMI, стікає в землю, не завдаючи шкоди електронної апаратури, що знаходиться усередині металевих шаф або коробів.

2. Сполучні кабелі для захисту прокладають в земляних траншеях під цементною або бетонною підлогою будівель або укладають в сталеві короби, які заземляють. Надійність підвищується, якщо кабель розгалужується і підводиться до декільком шаф з розділовими трансформаторами. У цьому випадку ізольовані ділянки автономного пристрою на Nvidia володіють великим опором ізоляції і малої ємністю проводів відносно землі. Також доцільно застосовувати фільтри від високочастотних перешкод.

3. Основні функції захисного розрядника - розімкнути лінію або відвести енергію для запобігання пошкодження в обладнанні. Для захисту апаратури можуть бути рекомендовані плавкі запобіжники і захисні вхідні пристосування, які являють собою різні релейні або електронні пристрої, що реагують на перевищення струму або напруги в ланцюзі.

4. Грозозахисні пристрої.

Забезпечують «стікання» великого розряду в землю без пошкодження ізоляційних елементів ліній.

5. Захист периферійних пристроїв.

Зазначені способи і засоби захисту повинні впроваджуватися в усі види електротехнічної та радіоелектронної апаратури з урахуванням характеру вражаючої дії електромагнітного імпульсу для забезпечення надійності роботи автономного пристрою на Nvidia в умовах НС мирного і воєнного часу.

Розрахунок екранів для захисту автономного пристрою на Nvidia від дії електромагнітного імпульсу. Перехідне гасіння енергії електричного поля екраном для сталі:

$$A=40+ К_{бв} \quad (6.13)$$

Для блоку перетворення сигналу:

$$A_1=40+49,8=99,8 \text{ (дБ)};$$

Для блоку формування сигналу:

$$A_2=40+46,6=96,6 \text{ (дБ)};$$

Для блоку обробки сигналу:

$$A_3=40+44,4=84,4(\text{дБ}).$$

Розрахуємо товщину захисних екранів. Для цього візьмемо найслабкішу ланку до дії EMI – блок перетворення сигналу:

$$t = \frac{A}{5,2 \cdot \sqrt{f}} = \frac{99,8}{5,2 \cdot \sqrt{15000}} = 0,156(см) \quad (6.14)$$

де f – найбільш характерна частота (f=15 кГц),

Отже, при екрануванні всіх елементів пристрої з використанням екрану товщиною 0,156 см автономний пристрій на Nvidia для задач штучного інтелекту буде стійкою в умовах дії електромагнітного імпульсу.

Підвищення стійкості роботи автономного пристрою на Nvidia можна досягти шляхом посилення найбільш слабких елементів і ділянок пристрої, а також завчасним проведенням комплексу інженерно-технічних, технологічних та організаційних заходів, які спрямовані на максимальне зниження дії вражаючих факторів.

6.7 Висновки до шостого розділу

В результаті виконання цього розділу було розглянуто такі питання охорони праці та безпеки в надзвичайних ситуаціях, як технічні рішення з гігієни праці і виробничої санітарії, визначення попереднє КПО для однобічного природного освітлення, технічні рішення з промислової та пожежної безпеки під час проведення розробки автономного пристрою на Nvidia для задач штучного інтелекту, безпека у надзвичайних ситуаціях.

Також в даному розділі було досліджено стійкість роботи автономного пристрою на Nvidia для задач штучного інтелекту в умовах дії загрозливих чинників НС. В умовах дії іонізуючого випромінювання система буде працювати стійко, так як граничне значення експозиційної дози випромінювання $Д_{гр}$ = 100000 Р значно більше ніж можливе значення дози $Д_{м}$ = 1273,6 Р. Отже підвищувати стійкість роботи автономного пристрою на Nvidia до впливу іонізуючого випромінювання непотрібно.

Вплив електромагнітного імпульсу на телекомунікаційну мережу призводить до порушення стійкості її роботи. Застосування екранування підвищило стійкість роботи автономного пристрою на Nvidia для задач штучного інтелекту в умовах дії електромагнітного імпульсу.

ВИСНОВКИ

В процесі виконання роботи були розглянуті існуючі рішення для обчислення неграфічних завдань засобами GPU. Для реалізації поставленого завдання була вибрана технологія NVIDIA CUDA, як найбільш ефективний обчислювальний засіб, що має ряд переваг : доступність, легкість у вивченні, підтримка практично будь-якої сучасної платформи NVIDIA, найбільша продуктивність і інші.

З використанням цієї технології CUDA був реалізований алгоритм розрахунку схем адвекції. Аналіз часу виконання алгоритму на GPU, на CPU і на CPU с використанням паралельних алгоритмів, показав, що застосування CUDA в рази скорочує час обробки цієї чималої розмірності. Практичне тестування продемонструвало, що навіть при використанні облаштувань початкового рівня, результативність таких обчислень перевищує на порядок, статистику за подібними операціями на центральному процесорі, навіть з використанням OpenMP. Але хотілося б відмітити, що не має сенсу використати технологію CUDA для роботи з невеликими об'ємами даних. Для малих об'ємів вхідних даних прискорення практично не спостерігалося.

Підводячи підсумок, хотілося б відмітити, що графічний процесор, маючи колосальні обчислювальні здібності, проте по-колишньому не зміг би повністю замінити діяльність центрального процесора, безумовною перевагою якого є універсальність, зате в його силах істотно розвантажити CPU, узявши на себе навантаження, що представляється найбільш трудомісткими і складними завданнями.

У цій роботі були розглянуті системи аналізу, що називаються нейронними мережами, а зокрема клас логічних нейронних мереж. Використовуючи алгебру логіки як базис, цей тип мереж дозволяє вирішувати різні завдання, у тому числі, ті що складно формалізувати і ті, що не формалізуються. До таких завдань відносяться роботи з висловлюваннями,

предикатами, для яких використовується формалізація даних для отримання значень істинності або хибності.

Оскільки логічні нейронні мережі, будучи одним з видів усієї безлічі нейронних мереж, не мають шаблонів для використання, а кожна така мережа будується під конкретне завдання, було прийнято рішення створити алгоритм, що дозволяє динамічно вибудовувати структуру логічної нейронної мережі. Цей конструктор мережі є універсальним, оскільки мережа вибудовується залежно від введених умов, отже, ми отримуємо автоматично масштабовану мережу під кожне завдання. Використання реалізованого алгоритму є доказом того, що неможливо реалізувати шаблонну логічну нейронну мережу, що підходить під будь-які завдання цього класу.

Реалізована мережа виконує перевірку на логічне слідування, шляхом перебору усієї безлічі значень змінних, що є дуже витратним за часом процесом. Проте паралелізування цього алгоритму дозволяє добитися збільшення швидкодії у декілька разів, що є основною метою цієї роботи.

Розгляд різної паралельної архітектури привів до висновку, що оптимальною технологією для реалізації використовуваних алгоритмів є використання графічної карти для паралельних обчислень, що не лише збільшує швидкодію системи, але і дозволяє звільнити центральний процесор від зайвого навантаження і залишити його для виконання фонових програм або складних операцій, вимогливих до продуктивності системи.

Технологія паралельних обчислень апаратній платформі CUDA, випущена компанією Nvidia, на сьогодні є однією з передових технологій обчислень на графічній карті. Використання паралелізму у багатьох сферах дозволяє домагатися поставлених цілей в рази швидше, ніж при використанні послідовних обчислень на центральному процесорі.

Використання технології CUDA в цій роботі дозволило реалізувати алгоритми роботи логічної нейронної мережі. Завдяки спеціальній структурі пам'яті, при використанні технології CUDA можливе одночасне обчислення до декількох тисяч процесів. Так, перебір усіх вхідних значень змінних може

бути змінений з послідовного на паралельний, завдяки чому кількість операцій буде обмежена тільки кількістю ядер в графічній карті. Таким чином, застосування технології CUDA дозволяє добитися максимальної ефективності роботи від реалізованої програми.

Реалізована мною в процесі цієї роботи логічна нейронна мережа дозволяє провести операцію логічного слідування для безлічі вхідних формул. Динамічна побудова структури відкриває безліч варіантів для використання мережі. Шляхом нескладних модифікацій, програма може бути змінена для реалізації перевірки безлічі істинності логічних функцій. Одним з очевидних прикладних застосувань цієї програми є рішення логічних завдань, наприклад, типових завдань по інформатиці.

В цілому, застосовність цієї програми дуже висока, оскільки за рахунок динамічної масштабованості можлива реалізація систем прийняття рішень, управління і аналізу даних.

Згідно проведених досліджень рівень комерційного потенціалу розробки становить 32,0 балів, що свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього). При оцінюванні за технічними параметрами, згідно узагальненого коефіцієнту якості розробки, удосконалений пристрій переважає існуючі аналоги приблизно в 3,43 рази. Отже можна зробити висновок про доцільність проведення НДДКР з дослідження та розробки автономного пристрою на Nvidia для задач штучного інтелекту.

ПЕРЕЛІК ПОСИЛАНЬ

1. McCulloch W.S., Pitts W. A logical calculus of the immanent in nervous activity// Bulletin of mathematical biophysics. 1943. V5.

2. Nordstrom T. Designing parallel computers for self-organizing maps. Forth Swedish Workshop on Computer System Architecture. 1992.

3. Барский А.Б. Логические нейронные сети. НОУ «ИНТУИТ», 2016.

4. Хайкин С. Нейронные сети: полный курс, 2-е изд., испр. – М.: Издательский дом «Вильямс», 2006. – 1104 с.

5. Чень Ч., Ли Р. Математическая логика и автоматическое доказательство теорем. М.: Наука. Главная редакция физико-математической литературы, 1983.

6. Shapiro E.Y. Concurrent prolog: Collected papers. Cambridge. MIT Press, 1987

7. Боголюбов Д.П., Чанкин А.А., Стемиковская К.В. Реализация алгоритма обучения самоорганизующихся карт Кохонена на графических процессорах // Промышленные АСУ и контроллеры. 2012. № 10. С. 30-35.

8. Официальный сайт компании Nvidia посвящённый технологии CUDA.

9. URL: https://developer.nvidia.com/cuda-zone (дата обращения 20.01.2016)

10. Методичні вказівки до виконання студентами-магістрантами наукового напрямку економічної частини магістерських кваліфікаційних робіт / Уклад. В.О. Козловський – Вінниця: ВНТУ, 2012. – 22 с.

11. Козловський В.О. Техніко-економічні обґрунтування та економічні розрахунки в дипломних проектах та роботах. Навчальний посібник. – Вінниця : ВДТУ, 2003. – 75с.

12. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа – Вінниця :

ВНТУ, 2016. – 113 с.

13. Сандерс Дж., Кэндрот Э. Технология CUDA в примерах: введение в программирование графических процессоров: Пер. с англ. – М.:ДМК Пресс, 2011.

14. Стецко И.П., Кулик В.Д. Комп'ютерне проектування цифрових систем [Електронний ресурс] URL: http://www.rfe.by/media/kafedry/kaf4/publication/stetsko/program-cifr-elektonika/lab.pdf (дата звернення: 17.11.2020).

15. Комолов Д., Золотухо Р. Использование микросхем специальной памяти для обеспечения защити ПЛІС FPGAот копирования//Компоненти и технологии. [Електронний ресурс] URL: http://www.kit-e.ru/articles/plis/2008 12 24.php#00.

16. ГОСТ 19.701-90(ИСО 5807-85) Единая система программной документации.

17. Козловський В. О. Основи підприємництва. Курс лекцій. Часть. 1. / В. О. Козловський – Вінниця : ВНТУ, 2005. – 196 с.

18. Козловський В. О. Основи підприємництва. Курс лекцій. Част. 2 / В. О. Козловський – Вінниця : ВНТУ, 2006. – 184 с.

19. Козловський В. О. Інноваційний менеджмент : Навчальний посібник / В. О. Козловський – Вінниця : ВНТУ, 2007. – 210 с.

20. Козловський В. О., Лесько О. Й. Бізнес-планування: Навчальний посібник / В. О. Козловський, О. Й. Лесько [2-е вид., доп. та переробл.] – Вінниця : УНІВЕРСУМ-Вінниця, ВНТУ, 2008. – 241 с.

21. ГОСТ 12.0.003-74.ССБТ. Опасные и вредные производственные факторы. Классификация.

22. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень.

23. ДБН В.2.5-28-2006. Природне і штучне освітлення.

24. Пособие по расчету и проектированию, естественного, искусственного и совмещенного освещения НИИСФ – М.: Стройиздат. 1985.

– 384 с.

25. ДСН 3.3.6-037-99. Санітарні норми виробничого шуму, ультразвуку та інфразвуку.

26. ДСН 3.3.6.039-99. Державні санітарні норми виробничої та загальної вібрацій.

27. ГОСТ 12.2.032-78. ССБТ. Рабочее место при выполнении работ сидя. Общие эргономические требования.

28. Методичні вказівки до опрацювання розділу "Охорона праці та безпека в надзвичайних ситуаціях" в дипломних проектах і роботах студентів спеціальностей, що пов’язані з функціональною електронікою, автоматизацією та управлінням / Уклад. О. В. Березюк, М. С. Лемешев. – Вінниця : ВНТУ, 2012. – 64 с.

29. ДНАОП 0.00-1.21-98 Правила безпечної експлуатації електроустановок споживачів. – К. : Держнаглядохоронпраці, 1998. – 382 с.

30. ДБН В.2.5-27-2006. Захисні заходи електробезпеки в електроустановках будинків і споруд.

31. ДБН В.1.1.7-2002. Пожежна безпека об’єктів будівництва.

32. НАПБ Б.03.001-2004. Типові норми належності вогнегасників.

33. СНиП 2.09.02-85. Противопожарные нормы проектирования зданий и сооружений.

34. Норми радіаційної безпеки України (НРБУ-97), МОЗ України. – К., 1997.

Додаток А
(обов’язковий)

ВНТУ

ПОГОДЖЕНО

“__” _____ 2020 р.

ЗАТВЕРДЖУЮ
Зав. кафедри РТ ВНТУ,
докт. техн. наук, професор
О.В. Осадчук
“__” _____ 2020 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання магістерської кваліфікаційної роботи
Розробка автономного пристрою на Nvidia
для задач штучного інтелекту

08-36.МКР.003.00.000 ТЗ

Керівник роботи: к.т.н. доцент
_____ Гаврілов Д.В.
(підпис)
“____”_____2020 р.
Виконавець: студент гр. РТ-19м
_____ Дука В.А.
(підпис)
“____”_____2020 р.

Вінниця 2020

1 ПІДСТАВА ДЛЯ ВИКОНАННЯ РОБОТИ

Робота проводиться на підставі наказу ректора по Вінницькому національному технічному університету №_214_ від 25.09.2020 р. та індивідуального завдання на магістерську кваліфікаційну роботу.

Дата початку роботи: 03.09.2020 р.

Дата закінчення: 10.12.2020 р.

2 МЕТА І ПРИЗНАЧЕННЯ МКР

Метою роботи є реалізація логічної нейронної мережі з використанням паралельного методу логічного слідування, виконана на мові C++ із застосуванням CUDA технології.

Об’єктом дослідження є розробка теоретичних засад, методів та засобів для задач штучного інтелекту на основі автономного пристрою на Nvidia.

Предметом дослідження – є автономні пристрої на Nvidia для задач штучного інтелекту.

В магістерській кваліфікаційній роботі для досягнення поставленої мети **розв’язуються такі завдання**:

1. Розглянути моделі нейронних мереж, їх реалізації і основні ідеї.

2.Визначити клас логічних нейронних мереж і виконуваних ними завдань.

3. Розглянути послідовний і паралельний методи резолюції, як засоби логічного виведення.

4. Встановити залежність між логічним слідуванням і логічним виведенням.

5. Проаналізувати отриману інформацію, і на її основі викласти ідеї паралельної реалізації логічної нейронної мережі і методу логічного слідування.

6. Реалізувати логічну нейронну мережу для завдання логічного слідування.

3 ДЖЕРЕЛА РОЗРОБКИ

1. McCulloch W.S., Pitts W. A logical calculus of the immanent in nervous activity// Bulletin of mathematical biophysics. 1943. V5.
2. Nordstrom T. Designing parallel computers for self-organizing maps. Forth Swedish Workshop on Computer System Architecture. 1992.
3. Барский А.Б. Логические нейронные сети. НОУ «ИНТУИТ», 2016.
4. Хайкин С. Нейронные сети: полный курс, 2-е изд., испр. – М.: Издательский дом «Вильямс», 2006. – 1104 с.
5. Чень Ч., Ли Р. Математическая логика и автоматическое доказательство теорем. М.: Наука. Главная редакция физико-математической литературы, 1983.
6. Shapiro E.Y. Concurrent prolog: Collected papers. Cambridge. MIT Press, 1987
7. Боголюбов Д.П., Чанкин А.А., Стемиковская К.В. Реализация алгоритма обучения самоорганизующихся карт Кохонена на графических процессорах // Промышленные АСУ и контроллеры. 2012. № 10. С. 30-35.
8. Официальный сайт компании Nvidia посвящённый технологии CUDA.
9. URL: https://developer.nvidia.com/cuda-zone (дата обращения 20.01.2016)

10. Сандерс Дж., Кэндрот Э. Технология CUDA в примерах: введение в программирование графических процессоров: Пер. с англ. – М.:ДМК Пресс, 2011.

11. Методичні вказівки до виконання студентами-магістрантами наукового напрямку економічної частини магістерських кваліфікаційних робіт / Уклад. В.О. Козловський – Вінниця: ВНТУ, 2012. – 22 с.

12. Козловський В.О. Техніко-економічні обґрунтування та економічні розрахунки в дипломних проектах та роботах. Навчальний посібник. – Вінниця : ВДТУ, 2003. – 75с.

13. Стецко И.П., Кулик В.Д. Комп'ютерне проектування цифрових систем [Електронний ресурс] URL: http://www.rfe.by/media/kafedry/kaf4/publication/stetsko/program-cifr-elektonika/lab.pdf (дата звернення: 17.11.2020).

14. ГОСТ 19.701-90(ИСО 5807-85) Единая система программной документации.

15. Козловський В. О. Основи підприємництва. Курс лекцій. Часть. 1. / В. О. Козловський – Вінниця : ВНТУ, 2005. – 196 с.

16. Козловський В. О. Основи підприємництва. Курс лекцій. Част. 2 / В. О. Козловський – Вінниця : ВНТУ, 2006. – 184 с.

17. Козловський В. О. Інноваційний менеджмент : Навчальний посібник / В. О. Козловський – Вінниця : ВНТУ, 2007. – 210 с.

18. Козловський В. О., Лесько О. Й. Бізнес-планування: Навчальний посібник / В. О. Козловський, О. Й. Лесько [2-е вид., доп. та переробл.] – Вінниця : УНІВЕРСУМ-Вінниця, ВНТУ, 2008. – 241 с.

19. Козловський В. О., Лесько О. Й. Бізнес-планування: Навчальний посібник / В. О. Козловський, О. Й. Лесько [2-е вид., доп. та переробл.] – Вінниця : УНІВЕРСУМ-Вінниця, ВНТУ, 2008. – 241 с.

4 ВИКОНАВЕЦЬ

Вінницький національний технічний університет, кафедра радіотехніки, студент групи РТ-19м Дука В.А.

5 ВИМОГИ ДО ВИКОНАННЯ МКР

- кількість ядер NVIDIA CUDA 128;
- час обробки, не більше 1 мс;
- споживана потужність, не більше 5 ВА;
- живлення пристрою має відбуватися від джерела живлення ЕОМ з напругами ± 5В;
- вид обміну інформацією паралельний;
- час готовності до роботи, не більше 0,3 хв;
- пам'ять LPDDR4, 64-біт 4 Гб;
- флеш-пам'ять eMMC 16 Гб;
- інформаційна ємність даних 270 біт;
- вид передачі даних неперервний;
- маса пристрою, не більше 0,2 кг;
- габаритні розміри плати, не більше 70×45×50 мм

6 ЕТАПИ МКР І ТЕРМІНИ ЇХ ВИКОНАННЯ

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Термін виконання		Очікувані результати	Звітна документація
1.	Огляд літературних джерел. Вибір та узгодження теми МКР	03.09.2020	14.09.2020	Проведено огляд літературних джерел. Вибрана тема	Узгодження теми МКР на кафедрі
2.	Аналіз літературних джерел. Попередня розробка основних розділів	15.09.2020	21.09.2020	Проведений аналіз літературних джерел по даній тематиці. Підготовлений матеріал основних розділів	Вступ
3.	Затвердження теми. Розробка технічного завдання	21.09.2020	25.09.2020	Розроблене ТЗ	Наказ ВНТУ про затвердження теми. Додаток А
4.	Аналіз вирішення поставленої задачі. Розробка структурної схеми	26.09. 2020	09.10. 2020	Проведений аналіз. Розроблені схеми пристрою	Звіт по переддипломній практиці Вступ Розділ 1-3
5.	Електричні розрахунки. Експериментальне дослідження	10.10.2020	25.10. 2020	Проведені розрахунки та дослідження	Розділ 4-6
6.	Розділ моделювання	26.10. 2020	04.11. 2020	Проведено моделювання	Результати моделювання
7.	Розробка графічної частини МКР	05.11. 2020	15.11. 2020	Плакати. Структурні та електричні схеми	Графічна частина
8.	Аналіз економічної ефективності розробки	16.11. 2020	19.11. 2020	Економічна частина	Розділ 7
9.	Охорона праці (ОП)	19.11. 2020	22.11. 2020	Частина БЖДПБ	Розділ 8
10.	Оформлення пояснювальної записки та графічної частини	23.11. 2020	29.11. 2020	Оформлена документація	ПЗ та графічна частина
11.	Нормоконтроль	30.11. 2020	01.12. 2020	Підпис нормоконтроля	Оформлена ПЗ та графічна частина
12.	Попередній захист МКР, доопрацювання, рецензування МКР	02.12. 2020	04.12. 2020	Позитивні відзиви	Відзив. Рецензія
13.	Захист МКР ЕК	11.12. 2020	14.12. 2020	Позитивний захист	Протокол ЕК

7 ОЧІКУВАНІ РЕЗУЛЬТАТИ ТА ПОРЯДОК РЕАЛІЗАЦІЇ МКР

У результаті виконання роботи будуть розроблені:

- математичне моделювання основних характеристик автономного пристрою на Nvidia для задач штучного інтелекту;
- нові межі використання автономних пристроїв на Nvidia для задач штучного інтелекту;
- розділ безпеки життєдіяльності і ЦЗ;
- економічна частина МКР.

Результати, отримані в процесі виконання даної роботи, можуть бути впроваджені в різних галузях науки і техніки.

8 МАТЕРІАЛИ, ЯКІ ПОДАЮТЬ ПІСЛЯ ЗАКІНЧЕННЯ РОБОТИ ТА ПІД ЧАС ЕТАПІВ

За результатами виконання МКР до ЕК подаються пояснювальна записка, графічна частина МКР, відгук керівника і рецензія.

9 ПОРЯДОК ПРИЙМАННЯ МКР ТА ЇЇ ЕТАПІВ

Поетапно результати виконання МКР розглядаються керівником роботи та обговорюються на засіданні кафедри.

Захист магістерської кваліфікаційної роботи відбувається на відкритому засіданні ЕК.

10 ВИМОГИ ДО РОЗРОБЛЮВАНОЇ ДОКУМЕНТАЦІЇ

Документація, що розробляється в процесі виконання досліджень повинна містити:

- дослідження поставленого питання;

- проектування розроблюваних автономних пристроїв на Nvidia для задач штучного інтелекту;
- методи дослідження автономних пристроїв на Nvidia для задач штучного інтелекту;
- економічну частину та розділ БЖДПБ.

11 ВИМОГИ ЩОДО ТЕХНІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ З ОБМЕЖЕНИМ ДОСТУПОМ

У зв’язку з тим, що інформація не є конфіденційною, заходи з її технічного захисту не передбачаються.

Додаток Б
(обов’язковий)

Розробка автономного пристрою на Nvidia для задач штучного інтелекту

Нейронні мережі

Базисна функція

$$\text{Neti}(\omega_{ij}, x_j)$$

де значення x j - значення вхідних векторів, а ω ij - відповідні синаптичні ваги.

а) Лінійна: функція виконує операцію складання типу:

$$\text{Neti}(\omega_{ij}, x_j) = \sum_{j=1}^{n} x_j \times \omega_{ij}$$

б) Радіальна: значення обчислюється за формулою відстані від вхідного значення до деякого «шаблонного»:

$$\text{Neti}(\omega_{ij}, x_j) = \sqrt{\sum_{j=1}^{n}(x_j - \omega_{ij})^2}$$

Основні активаційні функції:

а) Лінійні:

$$\text{F(net)} = \begin{cases} min, net \le min \\ net, min < net < max \\ max, net \ge max \end{cases},$$

де net - поточне значення, min і max - деякі вибрані порогові значення.

б) Порогові функції типу «стрибок»:

$$\text{F(net)} = \text{sign(net)}, \text{sign(net)} = \begin{cases} -1, net < 0 \\ 0, net = 0 \\ 1, net > 0 \end{cases}$$

в) Сигмоїдальні функції:

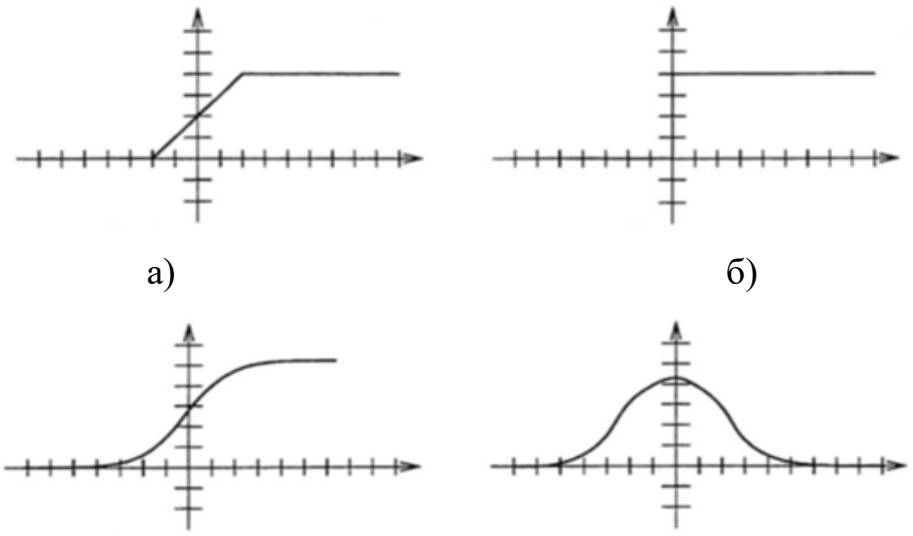
$$\text{F(net)} = \frac{1}{1+e^{-k \times net}},$$

де « k» - коефіцієнт нахилу функції. Зазвичай прийнято брати k = 1. При виборі нескінченно великого значення k, функція вироджується в порогову.

д) Функція Гауса :

$$\text{F(net)} = e^{\frac{-(S-R)^2}{2\sigma^2}},$$

де S = ||x - c|| - « відстань» між значеннями вхідного вектору, і деякого заданого шаблону; R - параметр, що відповідає за зсув функції по осі абсцис; σ - параметр, що визначає швидкість зміни значення функції, при віддаленні від центру.



а) б)

в) г)

Рисунок Б.1 – Види функцій: а) лінійна, б) порогова, в) сигмоїдальна, г) Гауса

$$V = \varepsilon \sum_{j=1}^{n} (V_j \omega_{ij} - h_i),$$

де $V_i = \begin{cases} 1, V \geq 1 \\ V, иначе \end{cases}$; $\varepsilon(x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases}$, Vj - величина сигналу, що поступає на j-й вхід нейрона.

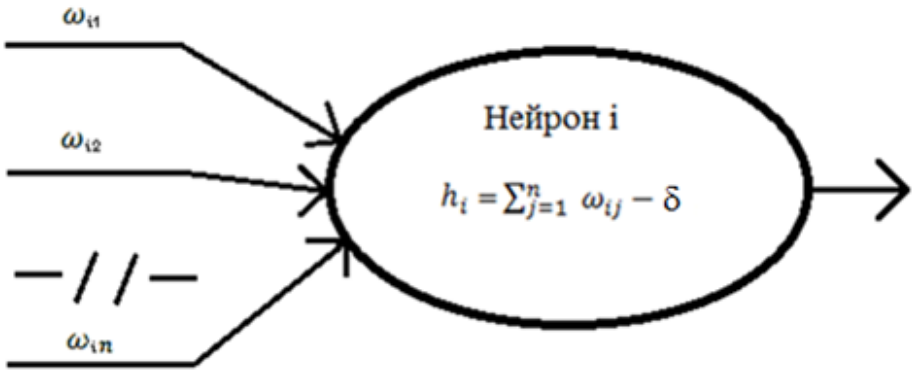


Рисунок Б.2 – Модель нейрона- кон’юнктора

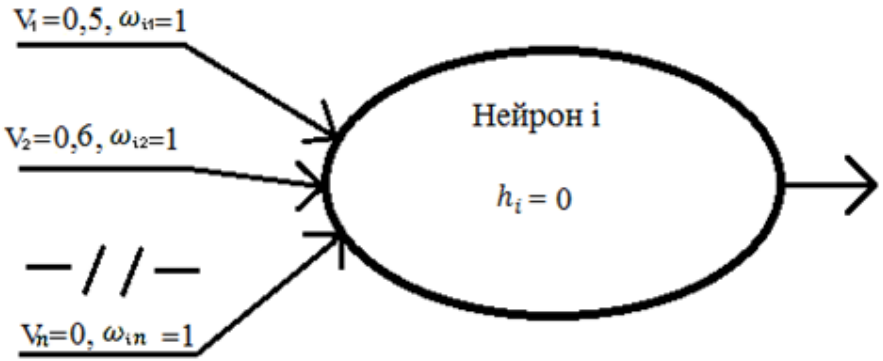


Рисунок Б.3 – Модель нейрона- диз’юнктора

Додаток В
(обов’язковий)

Розробка автономного пристрою на Nvidia
для задач штучного інтелекту

Організація пам'яті в CUDA

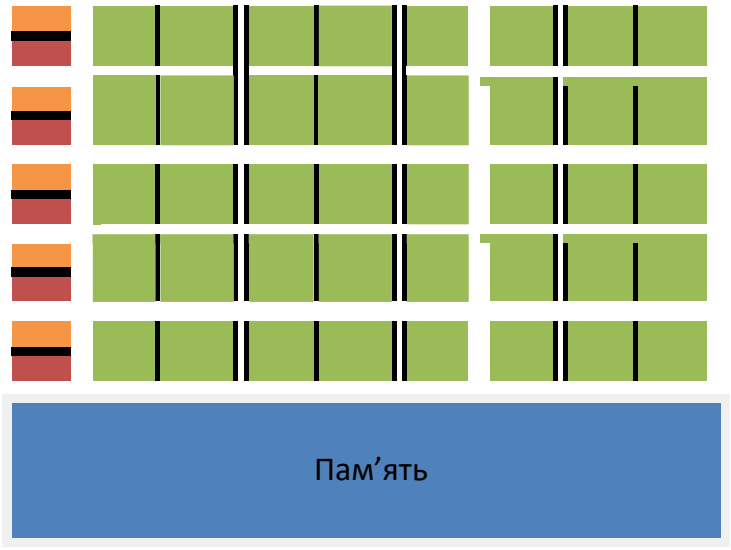


Рисунок В.1 – Будова GPU

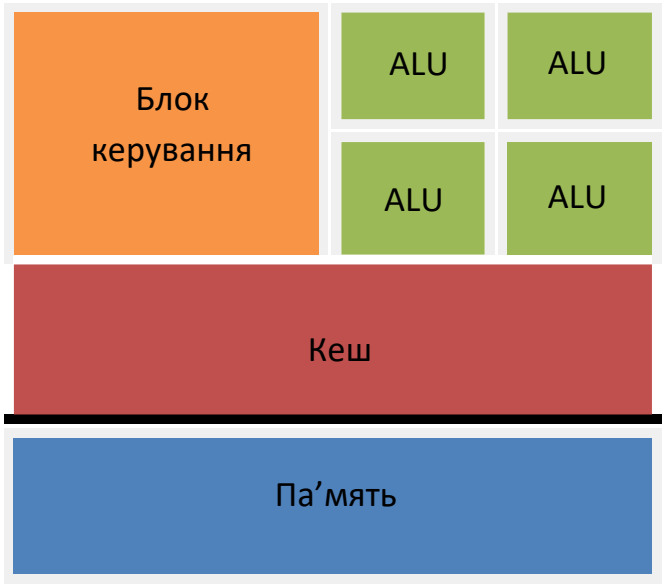


Рисунок В.2 – Будова CPU

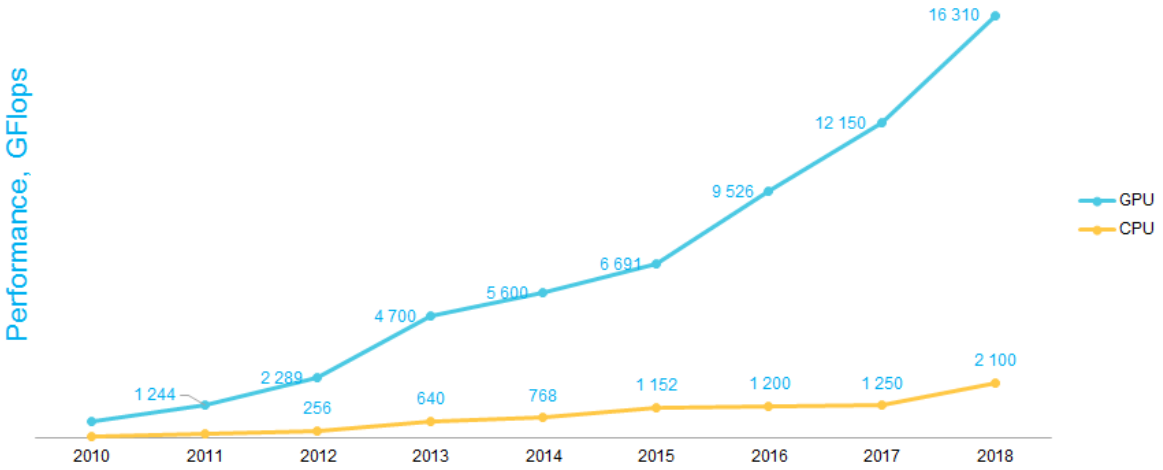


Рисунок В.3 - Динаміка зростання продуктивності GPU і CPU

Додаток Г
(обов’язковий)

Розробка автономного пристрою на Nvidia
для задач штучного інтелекту

Архітектура апаратної платформи Nvidia

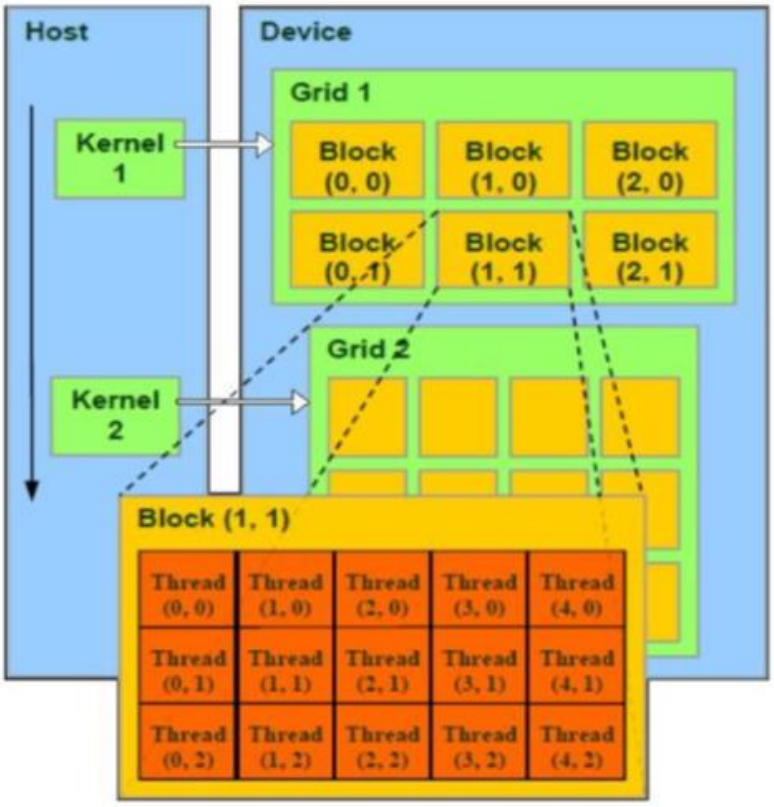


Рисунок Г.1 – Структура пам'яті в апаратній платформі Nvidia

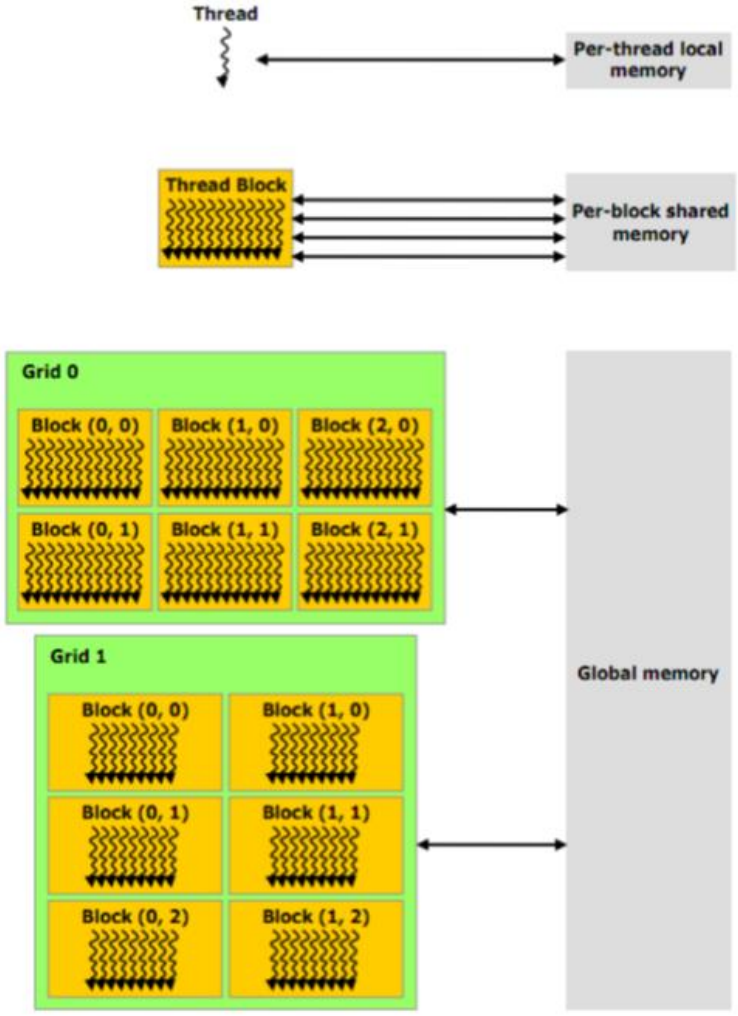


Рисунок Г.2 – Організація пам'яті в CUDA

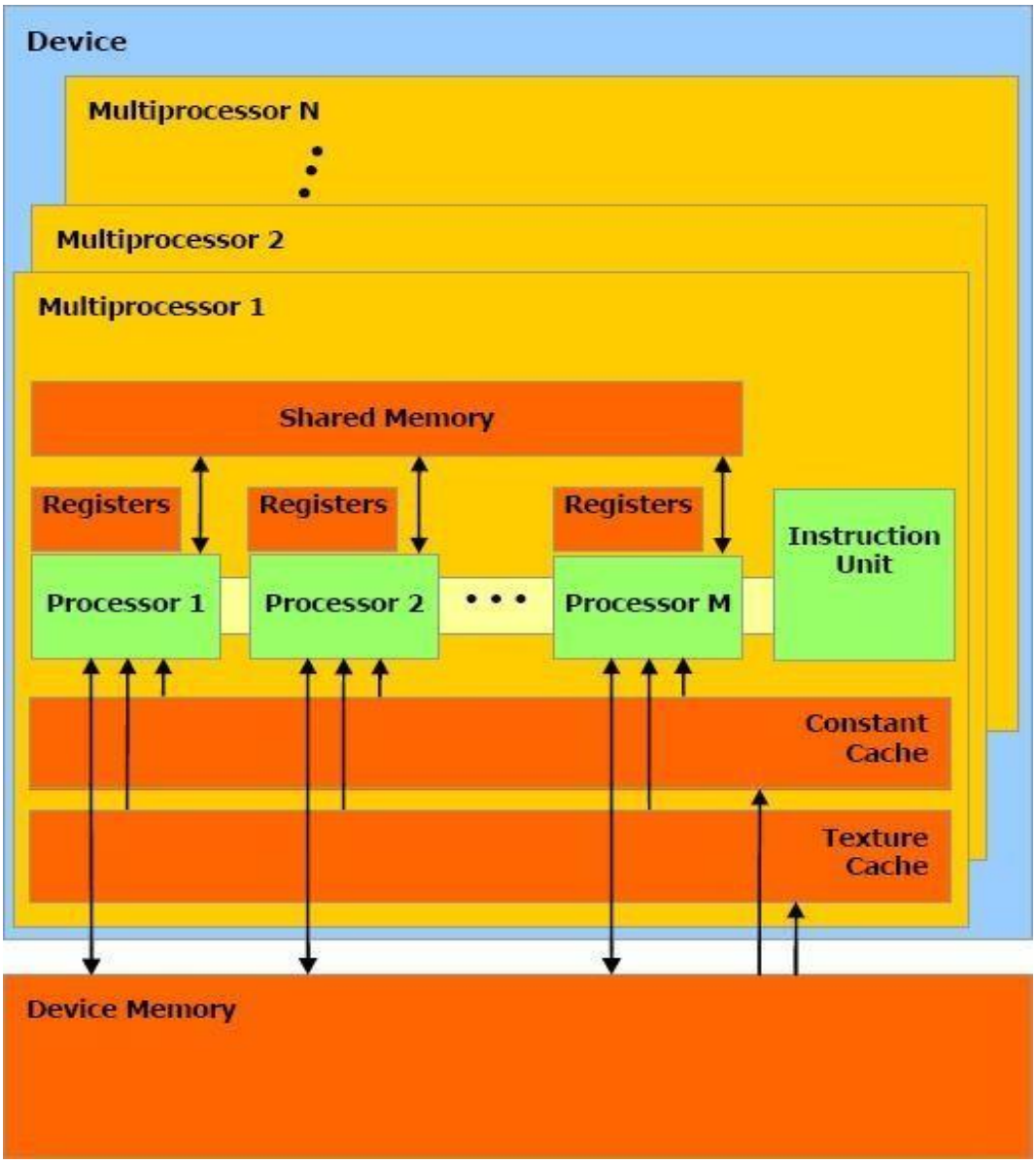


Рисунок Г.3 – Архітектура CUDA

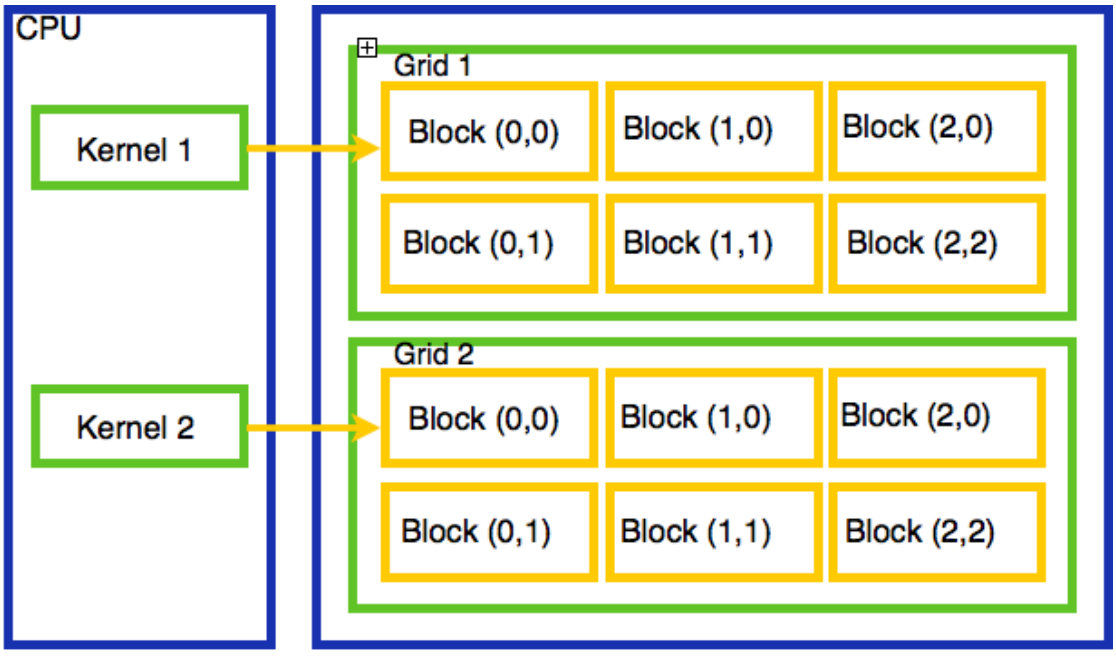


Рисунок Г.4 – Модель мультипроцесора NVIDIA

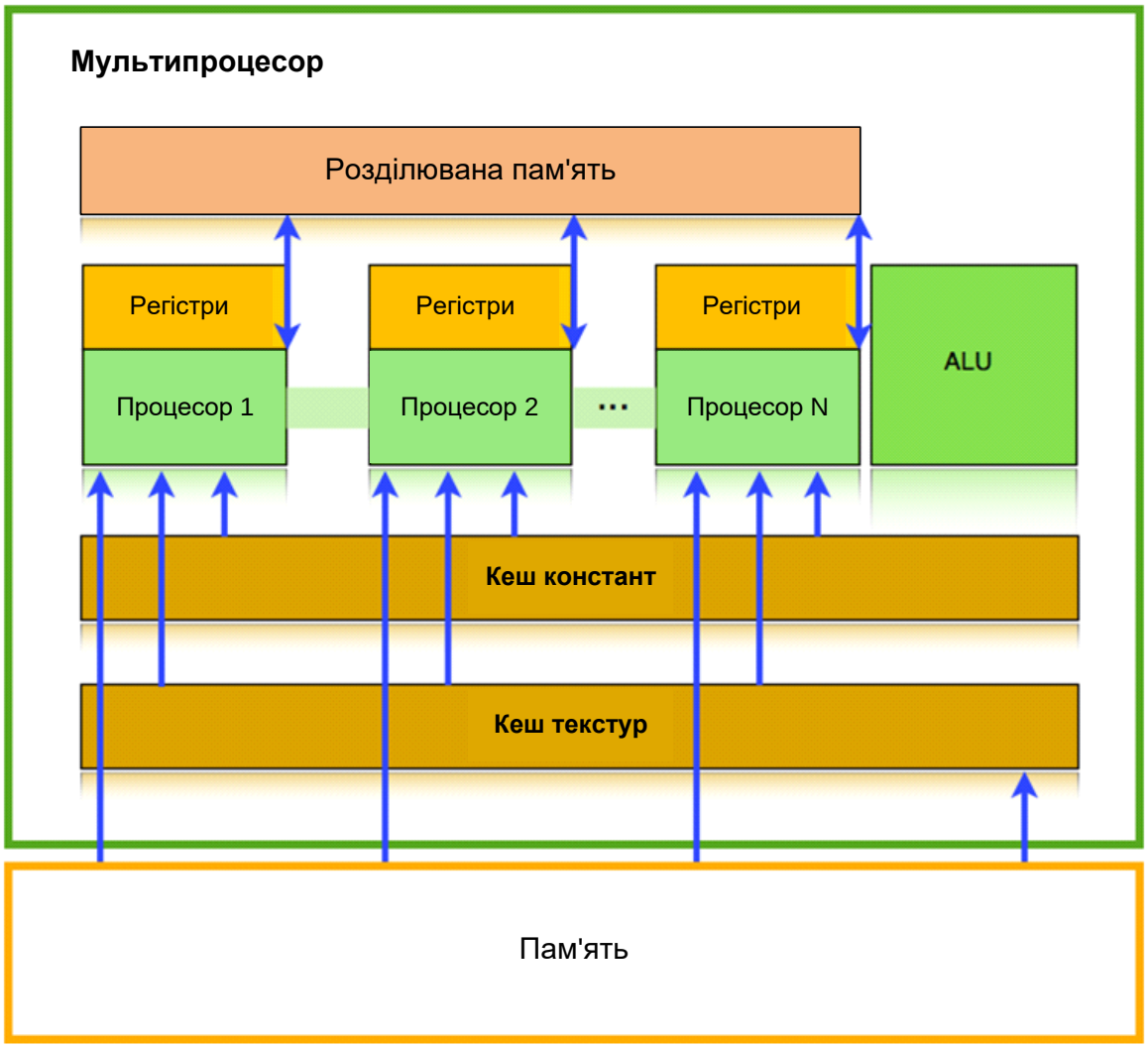


Рисунок Г.5 – Програмна модель пам'яті мультипроцесора

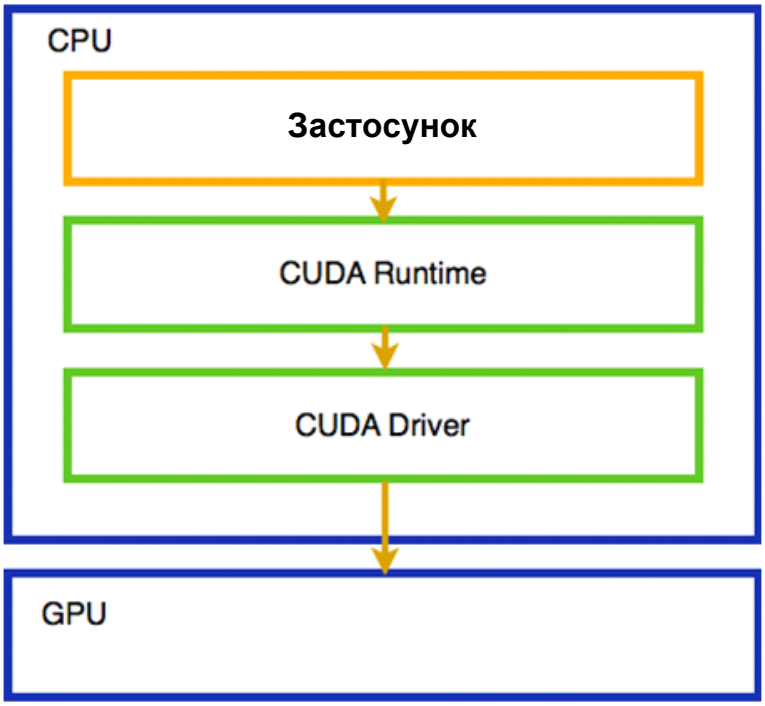


Рисунок Г.6 – Схема CUDA API

Додаток Д
(обов’язковий)

Розробка автономного пристрою на Nvidia
для задач штучного інтелекту

Опис алгоритму розрахунку схем адвекції

$$\frac{d\psi}{dt} = -\nabla * (v\psi) \tag{3.5}$$

$$\frac{d\psi}{dt} = -\frac{du\psi}{dx} \tag{3.6}$$

$$\psi_j^{n+1} = \psi_j^n - \frac{\Delta t}{\Delta x}[F_{j+1/2}^n - F_{j-1/2}^n] \tag{3.7}$$

$$F_{j+1/2}^n = \frac{\Delta x}{\Delta t}[c_j^+ \psi_j^n - c_j^- \psi_{j+1}^n] \tag{3.8}$$

$$c_{j+\frac{1}{2}}^n = u_{j+\frac{1}{2}}^n \frac{\Delta t}{\Delta x} \tag{3.9}$$

$$c_j^+ + c_{j-1}^- \le 1 \tag{3.10}$$

$$\psi_{ij}(x') = \sum_{k=0}^{l} a_{j,k} x'^k \tag{3.11}$$

Таблиця Д.1 - Значення коефіцієнтів aj, k у виразі (3.11)

l	$a_{j,0}$	$a_{j,1}$	$a_{j,2}$	$a_{j,3}$	$a_{j,4}$
0	ψ_j	–	–	–	–
1a	ψ_j	$\psi_{j+1} - \psi_j$	–	–	–
1b	ψ_j	$\psi_j - \psi_{j-1}$	–	–	–
2	ψ_j	$1/2(\psi_j - \psi_{j-1})$	$1/2(\psi_{j+1} - 2\psi_j - \psi_{j-1})$	–	–
3a	ψ_j	$1/6(-\psi_{j+2} + 6\psi_{j+1} - -3\psi_j - 2\psi_{j-2})$	$1/2(\psi_{j+1} - 2\psi_j - \psi_{j-1})$	$1/6(\psi_{j+2} - 3\psi_{j+1} - -3\psi_j - \psi_{j-2})$	–
3b	ψ_j	$1/6(2\psi_{j+1} - 6\psi_{j-1} + +3\psi_j - \psi_{j-2})$	$1/2(\psi_{j+1} - 2\psi_j - \psi_{j-1})$	$1/6(\psi_{j+2} - 3\psi_{j+1} - -3\psi_j - \psi_{j-2})$	–
4	ψ_j	$1/12(-\psi_{j+2} + 8\psi_{j+1} - -8\psi_{j-1} + \psi_{j-2})$	$1/24(-\psi_{j+2} + 16\psi_{j+1} - -30\psi_j + 16\psi_{j-1} - \psi_{j-2}$	$1/12(-\psi_{j+2} - 2\psi_{j+1} + +2\psi_{j-1} - \psi_{j-2})$	$1/24(\psi_{j+2} - 4\psi_{j+1} + +6\psi_j - 4\psi_{j-1} + \psi_{j-2}$

$$I_l^+\left(c_{j+\frac{1}{2}}\right) = \int_{\frac{1}{2}-c_j^+}^{1/2} \psi_{j,l}(x')\,dx' = \sum_{k=0}^{l} \frac{a_{j,k}}{(k+1)2^{k+1}}[1 - (1 - 2c_j^+)^{k+1}] \tag{3.12}$$

$$I_l^-\left(c_{j+\frac{1}{2}}\right) = \int_{-\frac{1}{2}}^{-\frac{1}{2}+c_j^-} \psi_{j+1,l}(x')\,dx' =$$

$$= \sum_{k=0}^{l} \frac{a_{j+1,k}}{(k+1)2^{k+1}}(-1)^k[1 - (1 - 2c_j^+)^{k+1}] \tag{3.13}$$

$$F_{j+1/2} = \frac{\Delta x}{\Delta t}[I_l^+\left(c_{j+\frac{1}{2}}\right) - I_l^-\left(c_{j+\frac{1}{2}}\right)] \tag{3.14}$$

$$0 \le I_l^+\left(c_{j+\frac{1}{2}}\right) + I_l^-\left(c_{j-\frac{1}{2}}\right) \le \psi_j \tag{3.15}$$

$$I_{i,j} = \int_{-1/2}^{1/2} \psi_{j,l}(x')dx' = \sum_{k=0}^{l} \frac{a_{j,k}}{(k+1)2^{k+1}}[(-1)^k + 1] \tag{3.16}$$

$$F_{j+1/2} = \frac{\Delta x}{\Delta t}\left[\frac{I_l^+\left(c_{j+\frac{1}{2}}\right)}{I_{l,j}}\psi_j - \frac{I_l^-\left(c_{j+\frac{1}{2}}\right)}{I_{l,j+1}}\psi_{j+1}\right] \tag{3.17}$$

Масив вхідних даних

Масив чисельних значень потоків через правий кордон комірку

Масив нових значень концентрації в наступний крок часу

Дані, що передаються в *i*- й потік для обробки

Дані, що повертаються *i*- м потоком і що зберігаються в локальній пам'яті для подальшої обробки

Дані, що повертаються *i*- м потоком в якості результату

Рисунок Д.1 - Передача даних між потоками

Додаток Е
(обов’язковий)

Розробка автономного пристрою на Nvidia
для задач штучного інтелекту

Апаратна реалізація

Рисунок Е.1 – Зовнішній вигляд модуля NVIDIA Jetson Nano

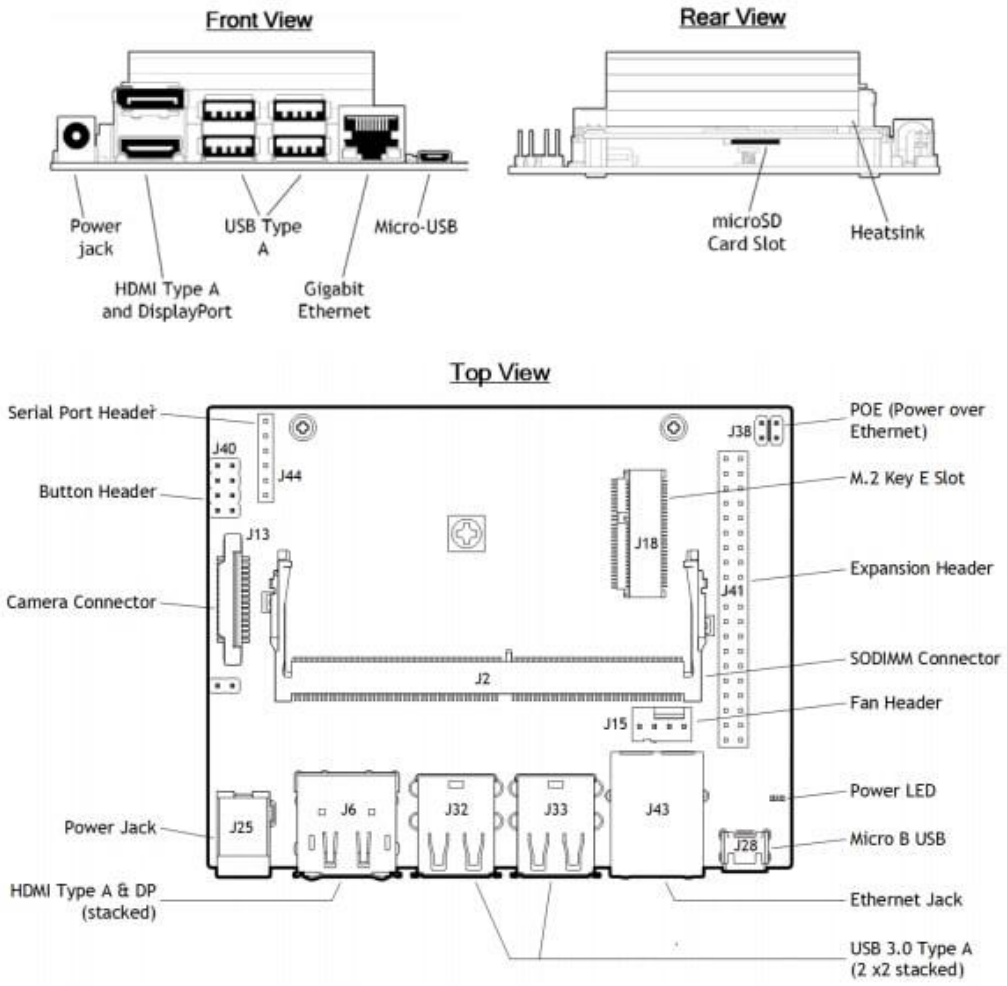


Рисунок Е.2 – Структурна схема NVIDIA Jetson Nano

Додаток Ж
(обов’язковий)

Розробка автономного пристрою на Nvidia
для задач штучного інтелекту

Результати моделювання

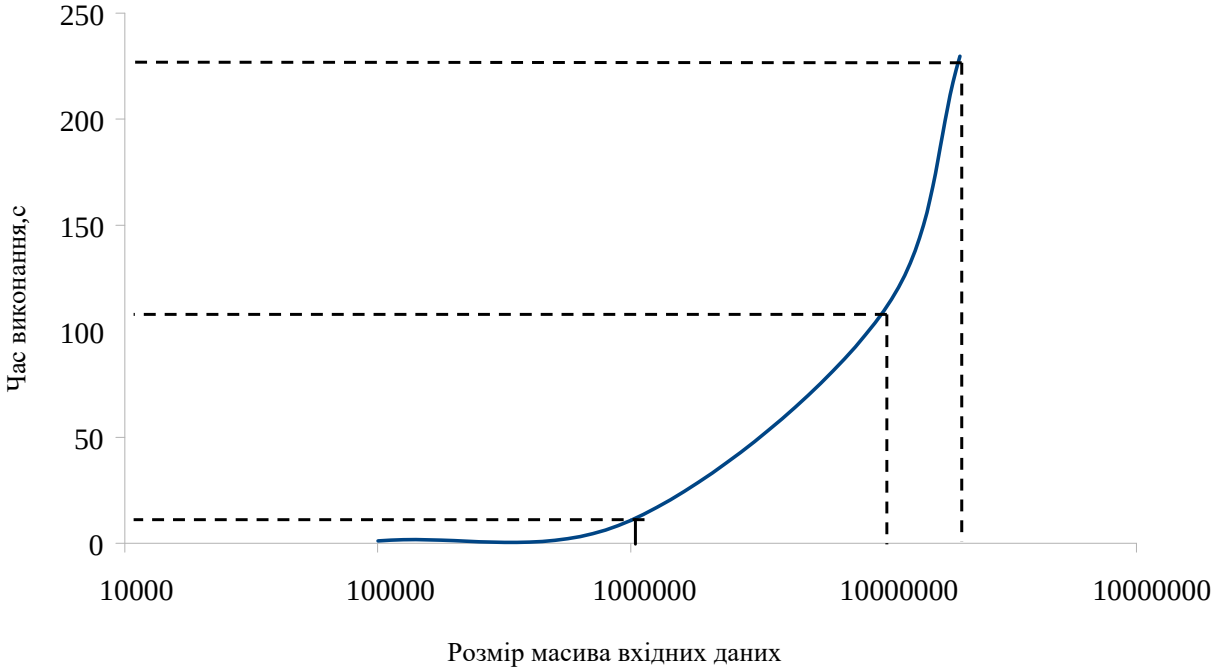


Рисунок Ж.1 - Графік залежності часу виконання від об'єму вхідних даних

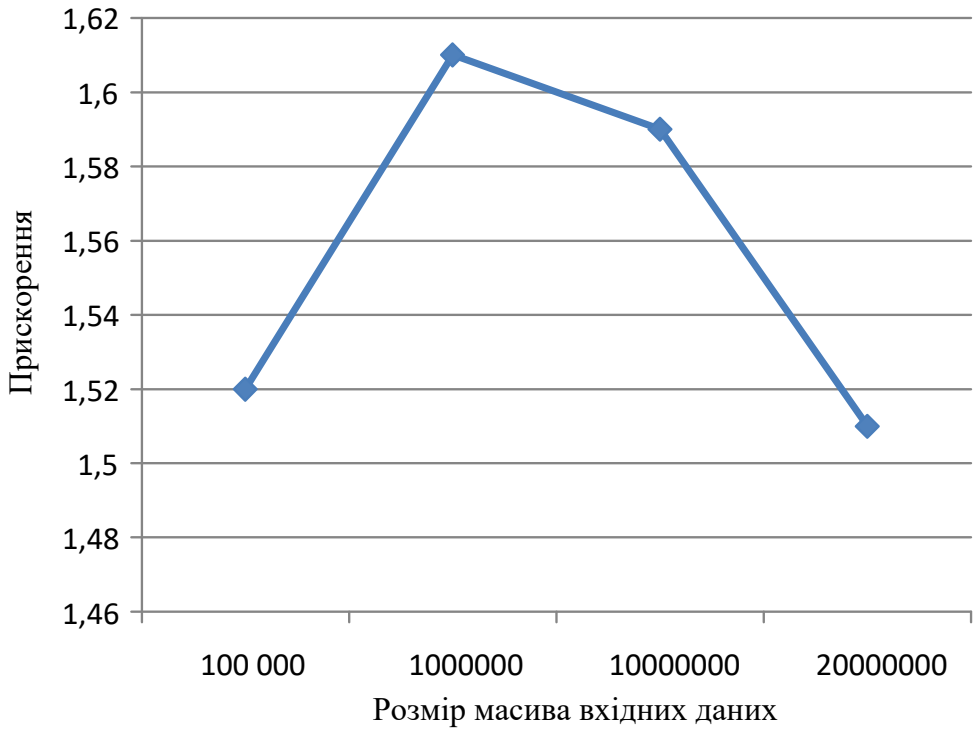


Рисунок Ж.2 - Графік залежності прискорення від об'єму вхідних даних

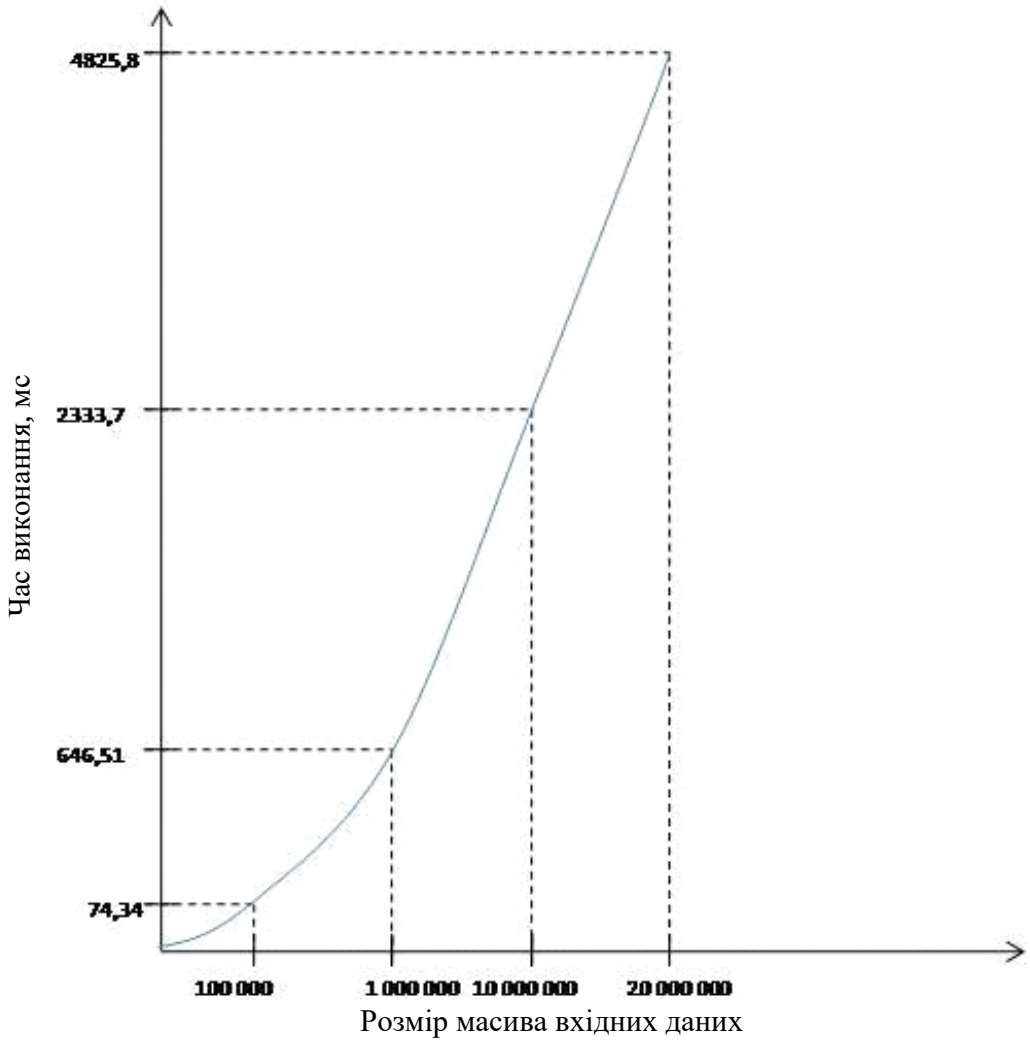


Рисунок Ж.3 – Графік залежності часу виконання від розміру вхідних даних

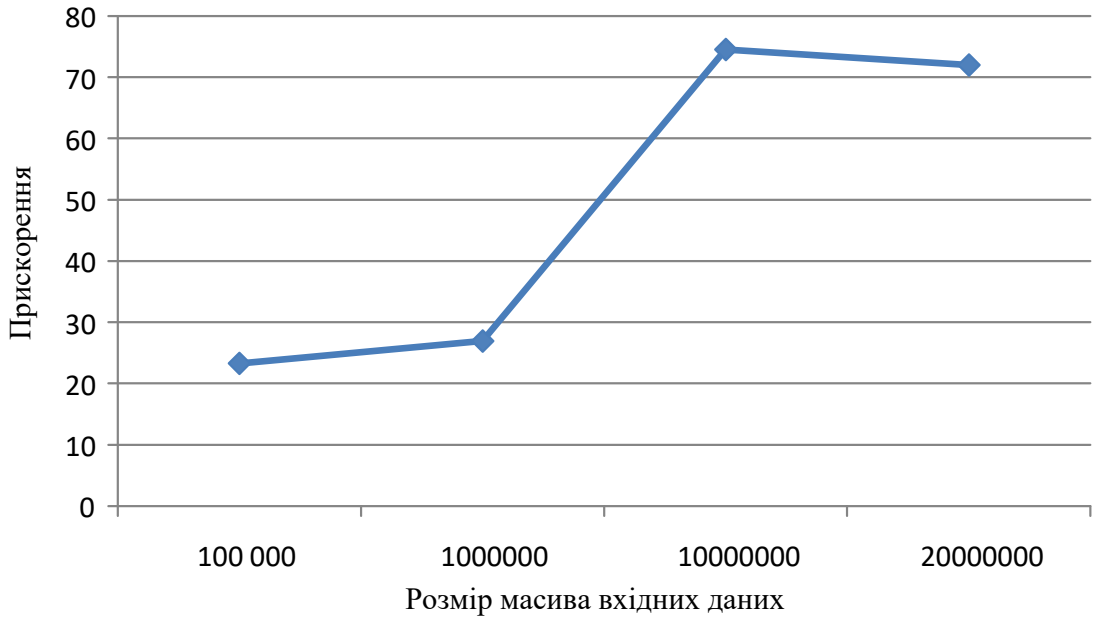


Рисунок Ж.4 – Графік залежності прискорення від об'єму вхідних даних

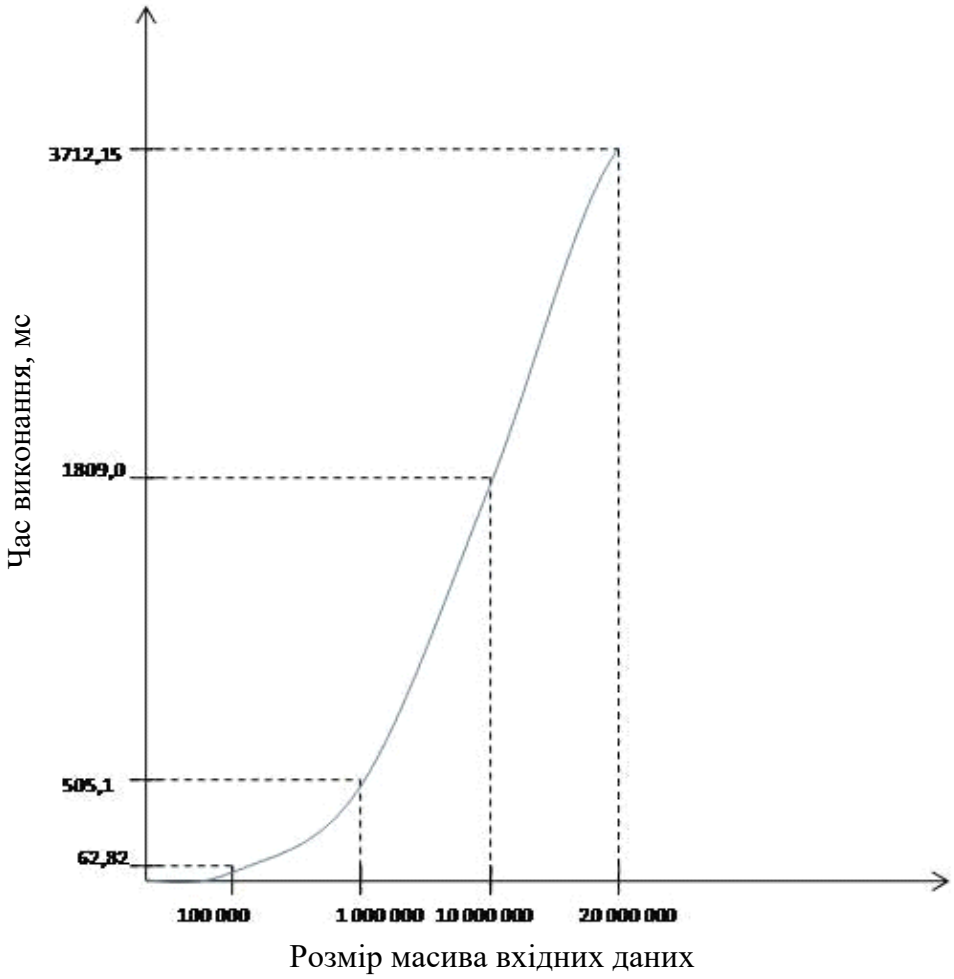


Рисунок Ж.5 - Графік залежності часу виконання від розміру вхідних даних

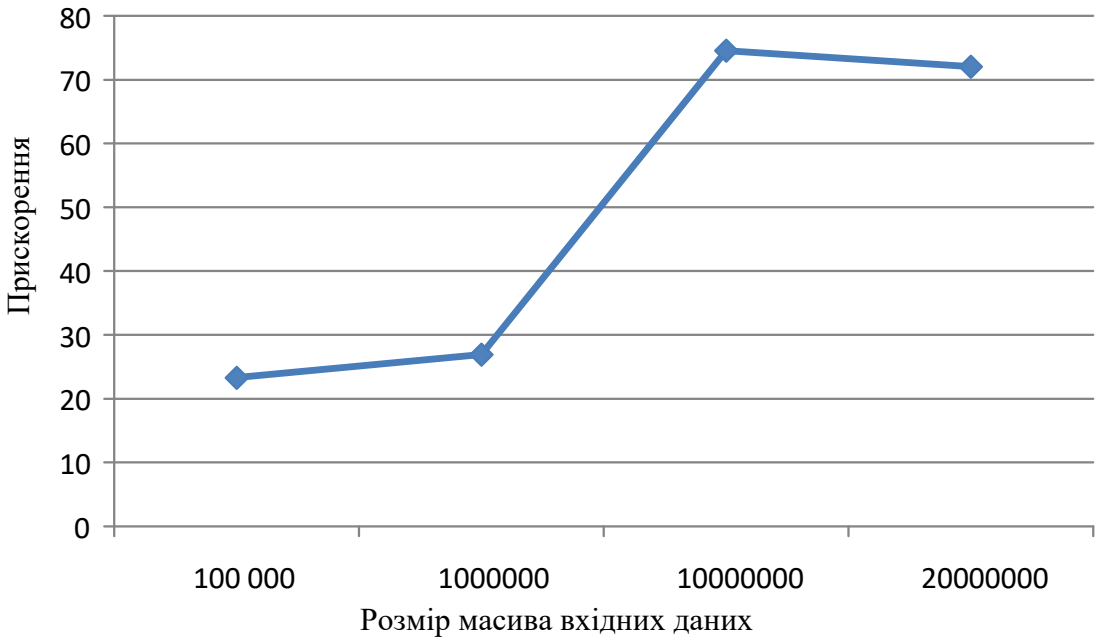


Рисунок Ж.6 - Графік прискорення алгоритму

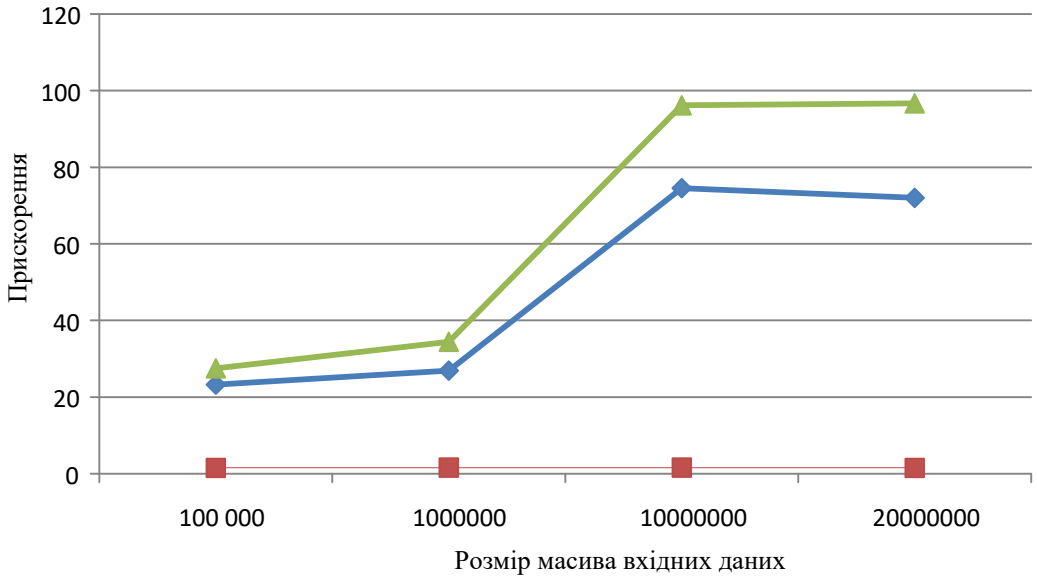


Рисунок Ж.7 – Прискорення усіх 3х алгоритмів

Таблиця Ж.1 – Залежність часу виконання від кількості потоків і об'єму даних

К-ть потоків/ розмір масиву даних	2	4	6	8	10
100 000	1,917	1,388	1,205	1,139	1,138
1 000 000	19,122	14,863	11,53	10,837	11,171
10 000 000	199,49	140,12	102,85	109,39	111,93
20 000 000	407,62	279,64	241,27	229,7	216,93

Таблиця Ж.2 – Залежність часу виконання від об'єму даних

Об'єм вхідних даних(ед.)	Час виконання (ms)
100 000	62,82
1 000 000	505,1
10 000 000	1809,0
20 000 000	3712,15

Додаток К

(довідниковий)

Розробка автономного пристрою на Nvidia для задач штучного інтелекту

Лістинги програми

Лістинг програми main.cpp

Початковий код програми на с++.

Файл main.cpp

```
#include <iostream>
#include <string>
#include <vector> // підключаємо заголовок списку
#include <iterator>
#include <stdlib.h>
#include <math.h>
#include "kernel.cu"

using namespace std;

int main()
{

char B = '0';
int e = 0;
string s;
while (e == 0)
{
vector<string> VS;
std::cout << "Enter number : " << '\n';
std::cin >> B;
std::cout << '\n';
if ((B == 'S') || (B == 's'))
{
while (s != ".")
{
std::cin >> s;
```

```
VS.push_back(s);
}
s = '0';
VS.pop_back();
}
if ((B == 'E') || (B == 'e'))
{
e++;
}
int PC = 0, NC = 0;
int ind[27];

string tot_elms, neg_elms;
for (unsigned int i = 65; i< 91; i++)
{
ind[i - 65] = i;
//cout<<ind[i-65]<<", "<<(char)i<<'\n';
}



if (VS.begin() != VS.end())
{

copy(VS.begin(), VS.end(), ostream_iterator<string>(cout, ","));
std::cout << '\n';
for (unsigned int i = 0; i< VS.size(); i++)
{
for (unsigned int j = 0; j< VS[i].size(); j++)
{
if (((int)VS[i][j] >= 65) && ((int)VS[i][j]<91))
{
```

```
if (j > 0) {
if (VS[i][j-1] == '!')
{
neg_elms = neg_elms + VS[i][j];
NC++;
}
}
for (unsigned int DN = 0; DN< 26; DN++)
{

if (((int)VS[i][j] == ind[DN]) && (int(VS[i][j]!= 0)))
{
tot_elms = tot_elms + VS[i][j];
ind[DN] = 0;
PC++;

}
}
}
}
}



//cout<<PC<<" <+> "<<tot_elms<<'\n'<<NC<<" <+>"<<neg_elms<<'\n';
//отримано кількість вхідних нейронів, множина невідбитих елементів і тих, для яких необхідне заперечення
//std::cout << '\n'<<neg_elms.length() << "==== " << tot_elms.length()<< " === " << VS.size() + 2 << '\n';

int TL = tot_elms.length();
```

```
int LS = VS.size() + 2;
int **array = new int*[TL];
for (int i = 0; i < TL; i++) { array[i] = new int[LS];}

for (unsigned int i = 0; i< tot_elms.length(); i++)
{
for (unsigned int j = 0; j< VS.size() + 2; j++)
{
array[i][j] = 0;
}

}
//допоміжна матриця для побудови структури роботи

for (int i = 0; i<tot_elms.length(); i++)
{
for (int j = 0; j<neg_elms.length(); j++)
{
if (tot_elms[i] == neg_elms[j])
{
array[i][1] = 1;
}
}
}

//стовпчик, що відповідає за наявність зворотних елементів
int *NOITN = new int [LS];//number of inputs to neuron;
for (unsigned int i = 0; i < LS; i++)
{
NOITN[i] = 0;
}
```

```
for (unsigned int i = 0; i< VS.size(); i++)
{
for (unsigned int j = 0; j< VS[i].size(); j++)
{
for (unsigned int k = 0; k < tot_elms.length(); k++)
{
if (j == 0) {
if (tot_elms[k] == VS[i][j])
{
array[k][i + 2] = 1; NOITN[i + 2]++;
}
}
if (j > 0) {
if ((tot_elms[k] == VS[i][j]) && (VS[i][j - 1]!= '!'))
{
array[k][i + 2] = 1; NOITN[i + 2]++;
}
if ((tot_elms[k] == VS[i][j]) && (VS[i][j - 1]== '!'))
{
array[k][i + 2] = -1; NOITN[i + 2]++;
}
}

}
}
}
cout << '\n';
// аналіз всіх вхідних формул

int NumOfInp = tot_elms.length();
int sr = pow(2, TL);
```

```
int tl2 = TL;
int **input = new int*[tl2];
for (int i = 0; i < tl2; i++) { input[i] = new int[sr]; }

for (unsigned int i = 0; i < TL; i++)
{
for (unsigned int j = 0; j < sr; j++)
{
input[i][j] = 0;
}
}
cout << '\n';
for (unsigned int i = 0; i<TL; i++)
{
for (unsigned int j = 0; j<sr; j++)
{

input[i][j] = (j >> i) & 0xF1; cout << input[i][j];
}cout << '\n';
}
cout << '\n';
//створння двійкової таблиці вхідних даних

double *ResultArray = new double[VS.size() + 2];

double *h = new double[VS.size() + 2];

double *W= new double[VS.size() + 2];

for (unsigned int i = 0; i<VS.size() + 2; i++)
{
```

```
ResultArray[i] = 0;
W[i] = 1;
h[i] = 0;
}
cudapeqi(array,input,ResultArray, TL,LS, sr,NOITN);
//cout << "----------" << '\n';

copy(VS.begin(), VS.end(), ostream_iterator<string>(cout, ","));
std::cout << '\n';
VS.clear();

}

}

return 0;
}
```

Лістинг програми kernel.cu

Початковий код програми на c++.
Файл “kernel.cu”

```
#include <iostream>
#include "cuda_runtime_api.h"
#include "cuda_runtime.h"
#include "cuda.h"
#include "device_launch_parameters.h"
#include <time.h>

using namespace std;
```

```
__global__ void add(double * ResultArray,int **array, int **input, int TL, int LS, int sr, int*NOITN,int TR) {
TR = 0;
int i = threadIdx.x; ResultArray[i] = 0;
int j = threadIdx.y;

if (array[i][j+2] == 1) { ResultArray[j+2] = ResultArray[j+2] + input[i][j]; NOITN[j+2]++; }
if (array[i][j+2] == -1) { ResultArray[j+2] = ResultArray[j+2] + ((input[i][j] + 1) % 2); NOITN[j+2]++;
}
for (unsigned int j = 2; j < LS; j++)
{
if ((ResultArray[j] == 0) || (ResultArray[LS - 1] == 0)) {
break;
}
else {ResultArray[j] = 0; TR = 1;
}
}
}
#define N 48
int cudapeqi(int **array,int **input,double *ResultArray, int TL,int LS,int sr,int *NOITN) {
cudaEvent_t start, stop;
float gpuTime = 0;
cudaEventCreate(&start);
cudaEventCreate(&stop);
cudaEventRecord(start, 0);

int TR=0;
```

```
int** dev_array;
int** dev_input;
double *dev_res;
int* dev_noi;

cudaMalloc((void **)&dev_array, sizeof(int)*TL*LS);
cudaMalloc((void **)&dev_input, sizeof(int)*TL*sr);
cudaMalloc((void **)&dev_res, sizeof(int)*LS);
cudaMalloc((void **)&dev_noi, sizeof(int)*LS);

cudaMemcpy(dev_array, &array, sizeof(int)*TL*LS, cudaMemcpyHostToDevice);
cudaMemcpy(dev_input, &input, sizeof(int)*TL*sr, cudaMemcpyHostToDevice);
cudaMemcpy(dev_res, &ResultArray, sizeof(int)*LS, cudaMemcpyHostToDevice);
cudaMemcpy(dev_noi, &NOITN, sizeof(int)*LS, cudaMemcpyHostToDevice);

cudaEventRecord(start, 0);
add << <TL, LS >> > (dev_res,dev_array, dev_input, TL, LS, sr, dev_noi,TR);
cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);
cudaEventElapsedTime(&gpuTime, start, stop);
cout << "time = " << gpuTime << '\n';
cudaMemcpy(ResultArray, dev_res, sizeof(int)*LS, cudaMemcpyDeviceToHost);

if (TR != 0) { cout << "true"; }
else {
```

```
cout << "false";
}

system("pause");
cudaFree(dev_array);
cudaFree(dev_input);
cudaFree(dev_res);
cudaFree(dev_noi);
cudaEventDestroy(start);
cudaEventDestroy(stop);

return 0;
}
```