МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

Бучковський Роман Васильович

Алгоритми обчислення суми квадратних різниць з використанням технології CUDA / Algorithms of the square differences sum calculation using CUDA technology

спеціальність:123 – Комп'ютерна інженерія
освітньо-професійна програма – Комп'ютерна інженерія

Випускна кваліфікаційна робота

Виконав студент групи КІм-21
Р.В. Бучковський
_____

Науковий керівник:
д.т.н., доцент, І.Г. Цмоць
_____

ТЕРНОПІЛЬ - 2019

РЕЗЮМЕ

Випускна кваліфікаційна робота на тему “Обчислення суми квадратних різниць з використанням технології CUDA” на здобуття освітнього ступеня “Магістр” зі спеціальності 123 “Комп’ютерна інженерія” написана обсягом 83 сторінок і містить 29 ілюстрації, 7 таблиці, 4 додатки та 51 джерела за переліком посилань.

Метою роботи є розроблення алгоритмів, структур обчислення суми квадратних різниць .

Методи досліджень. Для розв’язання поставлених задач у дипломній роботі використано: дослідження методів і алгоритмів обчислення суми квадратних різниць ; архітектуру графічного процесора GPU та програмну модель CUDA. Результати дослідження: розроблено ярусно-паралельну форму обчислення суми квадратних різниць, ярусно-паралельну форму обробки масиву на GPU, який забезпечує виявлення паралелізму.

Результати роботи можуть бути використані в науковій практиці, для розв’язання широкого кола задач із обчислення суми квадратних різниць.

Орієнтовні напрямки розвитку досліджень: обчислення суми квадратних різниць використовується при попередній обробці даних, розробці та моделюванні великого діапазону завдань. Обчислення суми квадратних різниць застосовується при обробці великих масивів інформації.

КЛЮЧОВІ СЛОВА: ГРАФІЧНИЙ ПРОЦЕСОР, АПАРАТНІ ЗАСОБИ, ЯРУСНА ПАРАЛЕЛЬНА ФОРМА, РЕАЛЬНИЙ ЧАС.

RESUME

The thesis on "Calculating the sum of square differences using CUDA technology" for obtaining the Master's degree in Computer Systems and Networks is written in 83 pages and contains 29 illustrations, 7 tables, 4 appendices and 51 sources by the list of links.

The purpose of the work is to develop algorithms, structures for calculating the sum of square differences.

Research methods. To solve the problems in the thesis used: research methods and algorithms for calculating the sum of square differences; GPU GPU architecture and CUDA software model. Results of the study: developed a tier-parallel form of calculating the sum of square differences, a tier-parallel form of processing an array on the GPU, which provides detection of parallelism.

The results of the work can be used in scientific practice to solve a wide range of problems of calculating the sum of square differences.

Guidelines for Research Development: The calculation of the sum of squared differences is used in preliminary data processing, development and modeling of a large range of tasks. The calculation of the sum of square differences is used when processing large arrays of information.

KEYWORDS: GRAPHIC PROCESSOR, HARDWARE, LEVEL PARALLEL FORM, REAL TIME.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

НВІС – надвелика інтегральна схема

ОП – операційними пристроями

ПЕ – процесорний елемент

ПЗ – програмне забезпечення

ФО – функціональні оператори

ПГА – потоковий граф алгоритму

БПП – багатопортова пам’ять.

CPU – центральний процесор

CUDA – Compute Unified Device Architecture програмно-апаратна архітектура паралельних обчислень

GPU – Graphics Processing Unit – графічний процесор

ВСТУП

Актуальність теми. На сучасному етапі розвитку інформаційних технологій намітилась тенденція збільшення ролі логічних обчислень, а саме обчислення суми квадратних різниць. Такі обчислення використовуються в нейромережах для нормалізації вхідних даних. Обчислення максимальних і мінмальних чисел мають ряд специфічних особливостей, які не дозволяють при їх реалізації використовувати відомі обчислювальні методи і алгоритми. Існуючі апаратні засоби в основному орієнтовані на реалізацію алгоритмів з перевагою обчислювальних операцій над логічними і вони не враховують специфіки обробки логічних даних. Особливістю реалізації логічних обчислень на сучасних апаратних засобах є неефективність використання багаторозрядних операційних пристроїв, що істотно знижує продуктивність та ефективність використання обладнання. У зв’язку з цим особливої актуальності набуває проблема розробки нових ефективних методів і алгоритмів логічних обчислень, орієнтованих на НВІС-реалізцію. Для переходу від алгоритмів логічних обчислень до НВІС-архітектур доцільно використовувати методи просторово-часового відображення таких алгоритмів у паралельні однорідні структури з високою ефективністю використання обладнання. Процес відображення алгоритмів у паралельні спеціалізовані НВІС-архітектури є складним і вимагає взаємної адаптації як алгоритмів, так і структур обчислювальних засобів.

Нормалізацію вхідних даних в нейромережах – це процедура попередньої обробки вхідних даних (навчальних, тестових і робочих вибірок), при якій значення ознак, які формують вхідний вектор, приводиться до деякого заданого діапазону. Після нормалізації всі значення вхідних признаків будуть приведені до деякого вузького діапазону (зазвичай, [0, 1] або [-1, 1]).

Мета і завдання дослідження. Метою роботи є розроблення алгоритмів, структур обчислення суми квадратних різниць.

Для досягнення поставленої мети в роботі необхідно виконати такі завдання:

– проаналізувати галузі застосування, методи та засоби обчислення суми квадратних різниць;

– розробити алгоритми і процесорні елемнти для обчислення суми квадратних різниць;

– розробку та моделювання пристрою для обчислення суми квадратних різниць.

Об'єкт дослідження – процеси обчислення суми квадратних різниць.

Предмет дослідження - методи, алгоритми та засоби обчислення суми квадратних різниць.

Методи досліджень. Для розв'язання поставлених задач у дипломній роботі використано: дослідження методів і алгоритмів обчислення суми квадратних різниць; архітектуру графічного процесора GPU та програмну модель CUDA.

Наукова новизна одержаних результатів:

– розроблений алгоритм обчислення суми квадратних різниць, який за рохунок використання спільної шини забезпечує високу швидкодію.

– розролена структура о днорідною з регулярними звязками, що забезпечує ефективну реалізацію у вигляді НВІС;

– розроблена модель пристрою обчислення суми квадратних різниць показує, що швидкодія визначається в основному розрядністю чисел.

Публікації та апробація. Публікацію тез ВКР та тему «Апаратна реалізації обчислення максимального і мінімального чисел в масиві даних» надруковані у виданні «VI Всеукраїнської школи-семінару молодих вчених і студентів».[51]

Впровадження результатів. Магістерська робота та тему «Обчислення суми квадратних різниць з використанням технології CUDA» має практичну значимість і планується до впровадження у науково-дослідному інституті інтелектуальних комп'ютерних систем».

В першому розділі проаналізовано галузі застосування, методів та засобів обчислення суми квадратних різниць з використанням технології CUDA.

В другому розділі розроблено алгоритми і процесорні елементи для обчислення суми квадратних різниць

В третьому розділі розроблено програм.не забезпечення та здійснено синтез апаратного пристрою сортування даних методом злиття.

1 АНАЛІЗ ГАЛУЗЕЙ ЗАСТОСУВАННЯ, МЕТОДІВ ТА ЗАСОБІВ ОБЧИСЛЕННЯ СУМИ КВАДРАТНИХ РІЗНИЦЬ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ CUDA

1.1 Галузі застосування алгоритмів обчислення суми квадратних різниць

У сучасних комп’ютерах висока продуктивність обробки даних досягається завдяки використанню просторового і часового паралелізму. Особливо цікавою є концепція за допомогою якої підвищення продуктивності комп’ютерного пристрою досягається шляхом наближення його структури до структури виконуваного алгоритму [1]. Перетворення алгоритму в його апаратну модель відбувається шляхом повного апаратного відображення потокового графа виконуваного алгоритму (ПГА) операційними пристроями (ОП), які виконують функціональні оператори (ФО) алгоритму і з’єднані між собою відповідно до графа алгоритму [2]. Пристрої, структура яких адаптована до ПГА, мають наступні переваги.

Ви можете виконувати операції на всіх операндах незалежно від стану інших операцій (синхронізація не потрібна).

Обмін даними між операціями чітко визначений. Процеси керуються передачею даних між ними.

Алгоритм працює за один хід, що прискорює завдання в кілька разів. Перш за все, існує багато питань щодо того, як можна побудувати та модифікувати графіки алгоритмів для ваших апаратних реалізаторів [1-4].

Розглядаються різні варіанти синтезу пристроїв сортування паралельних обчислень та проектування програмного опису алгоритму аж до апаратної реалізації з можливістю зміни ширини паралельної форми.

Відсоток сортування серед процесів, створених комп'ютером, досить високий, завдання є однією з найпоширеніших проблем обробки даних і,

звичайно, розуміється як завдання розміщення невпорядкованих елементів набору значень $\overline{X} = \{x_1, x_2, ..., x_N\}$ в порядку монотонного зростання або спадання $\overline{X} \approx \overline{Y} = \{(y_1, y_2, ..., y_N) : y_1 \le y_2 \le ... \le y_N\}$. Сортування на основі ПГА будуються комутуючі мережі, які забезпечують здатність до обміну даними між компонентами багатопроцесорної комп'ютерної системи [2]. Також, алгоритми сортування – зручний засіб для визначення переваг тієї або іншої моделі та для їх порівняння між собою.

Обчислювальна складність процесу замовлення досить висока. Тому для деяких відомих простих методів (сортування бульбашок, сортування за включенням тощо) кількість необхідних операцій визначається квадратичною залежністю від кількості даних замовлення [5]. Паралельне виконання операцій алгоритму сортування декількома операційними пристроями одночасно значно прискорює час виконання алгоритму. Серед алгоритмів сортування можна проводити паралельне виконання операцій для алгоритмів, у яких послідовність операцій залежить лише від кількості вхідних даних, а не від значень їх ключів (неадаптованих алгоритмів) [6]. Неадаптивні алгоритми сортування включають: алгоритм сортування за методом «прямої» перестановки [7], алгоритм сортування за допомогою модифікованого методу «бульбашки» [6], метод сортування за методом Бетчера [5]. Деякі недоліки виникли під час проектування, які стають суттєвими із ускладненням проектів. Одним з недоліків є те, що вам часто доводиться працювати вручну, щоб паралелізувати алгоритми або змінити ширину паралельної форми. Виявляється, цей вид робіт займає додатковий час, є висококваліфікованим, і все ще не гарантує підтримання простого рішення, що робить проект невигідним на певному етапі ускладнення. Це змушує шукати нові способи прискорити дизайн.

Використання нейронних мереж є одним із основних способів прискорити розробку інструментів пошуку суми квадратних різниць.

Вище вказані паралельні способи сортування базуються на використанні тієї ж основної операції "порівняння та упорядкування", що і складається з

порівняння тієї чи іншої пари ключів з набору даних з метою сортування та впорядкування цих значень, якщо їх порядок не відповідає вимогам сортування. Ці методи відрізняються лише порядком порівняння пар, і основна операція "порівняння та перестановка" однакова для таких алгоритмів. Дані PGA, отримані за алгоритмами сортування 16 вхідних значень, наведені на малюнку 1.1

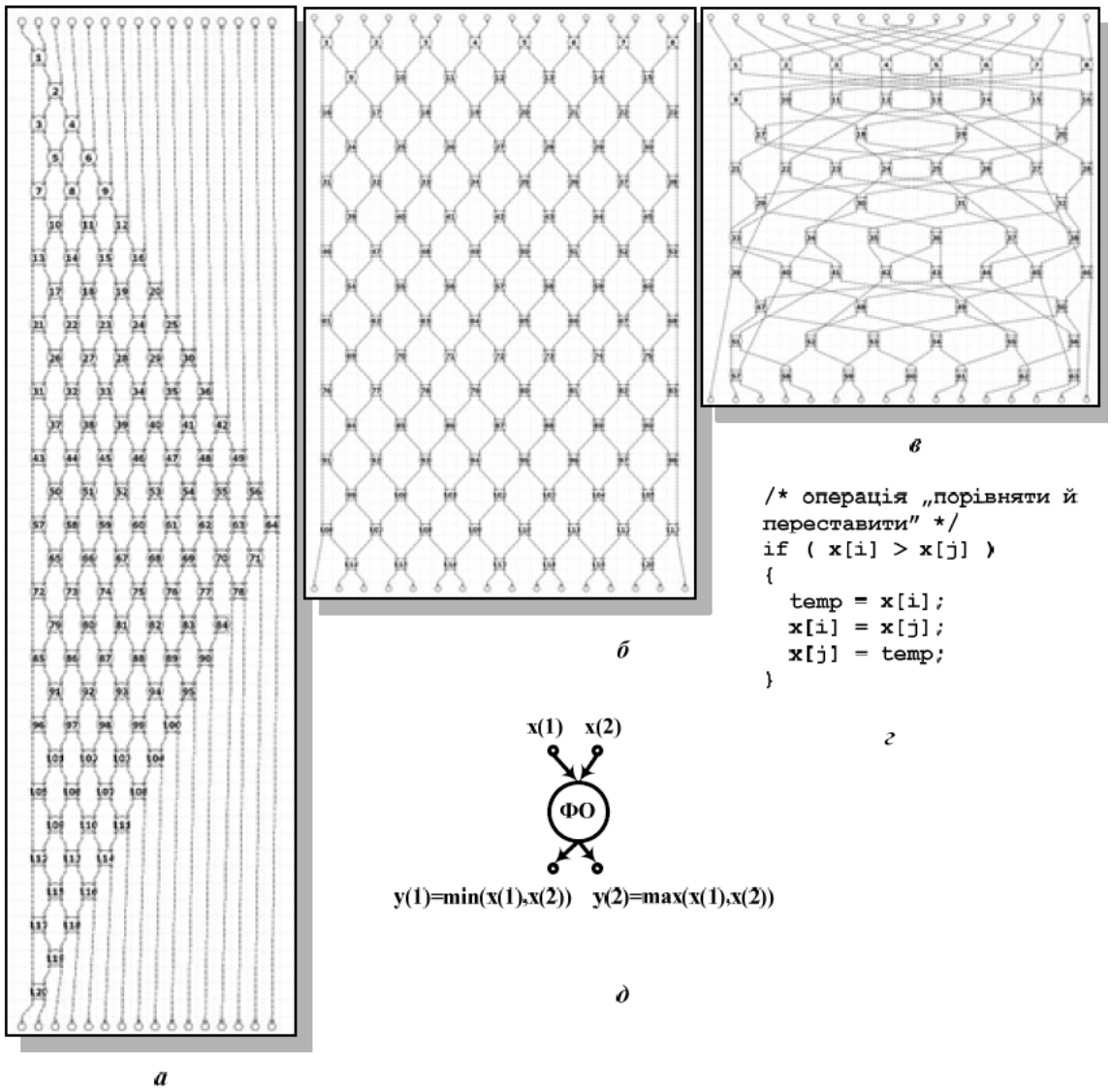

Рисунок 1.1 – Алгоритми сортування 16 значень: а) ПГА сортування модифікованим методом “бульбашки”; б) ПГА сортування “парно-непарної” перестановки; в – ПГА сортування Бетчера; г,
д) базова операція “порівняти й переставити”

Для однакової кількості вхідних значень N дані алгоритми мають однакову ширину ПГА – $\left[\frac{N}{2}\right]$ але виконують різну кількість операцій „порівняти й переставити” та мають різну кількість ярусів.

Загальна кількість ФО ПГА дорівнює загальній кількості операцій „порівняти й переставити”. Для алгоритмів сортування за методом “парно-непарної” перестановки та модифікованим методом “бульбашки” для N вхідних значень кількість ФО дорівнює $N(N-1)/2$ [2], а для алгоритму сортування за методом Бетчера – $0{,}48N\ln^2 N$, що підтверджено результатами моделювання. Висота ПГА для цих алгоритмів різна і є найбільшою для модифікованого алгоритму сортування за методом “бульбашки” – $2N-3$ [2]. Для алгоритму сортування за методом “парно-непарної” перестановки висота ПГА дорівнює кількості вхідних значень N , і для алгоритму сортування за методом Бетчера є найменшою і дорівнює $\frac{1}{2}\lceil\log_2 N\rceil(\lceil\log_2 N\rceil+1)$. Оскільки алгоритми виконують одну операцію „порівняти й переставити”, можна вважати, що часова затримка всіх ФО однакова і дорівнює одиниці. Апаратна складність цих обчислювальних пристроїв дорівнюватиме кількості ФО ПГА, а часова затримка – кількості ярусів ПГА.

Проаналізувавши таблиця 1.1, було зроблено висновок, що з усіх алгоритмів алгоритм сортування за методом Бетчера має найкращі часові властивості за найменших апаратних затрат. Оскільки ПГА сортування за методом Бетчера для N вхідних даних містить $\frac{1}{2}\lceil\log_2 N\rceil(\lceil\log_2 N\rceil+1)$ ярусів. [5], то час виконання алгоритму становитиме $0(\frac{1}{2}\lceil\log_2 N\rceil(\lceil\log_2 N\rceil+1))$.

Таблиця 1.1 – Обчислювальні характеристики алгоритмів сортування залежно від кількості вхідних значень

Алгоритм сортування	Кількість операцій, w	Кількість ярусів, t
модифікований “бульбашки”	$N(N-1)/2$	$2N-3$
“парно-непарної” перестановки	$N(N-1)/2$	N
за методом Бетчера	$0{,}48N\ln^2 N$	$\frac{1}{2}\lceil\log_2 N\rceil(\lceil\log_2 N\rceil+1)$

Штучні нейронні мережі поширюються протягом останніх 20 років, завдяки чому можна вирішити складні проблеми обробки даних, які часто значно перевищують точність інших методів статистичного та штучного інтелекту або є єдино можливим методом вирішення окремих проблем. Нейронна мережа схожа за будовою та властивостями нервової системи живих організмів: Нейронна мережа складається з великої кількості простих обчислювальних елементів (нейронів) і є більш складною, ніж можливості кожного окремого нейрона. На вході нейронна мережа отримує серію вхідних сигналів і видає відповідну відповідь, тобто вихідні сигнали, що є вирішенням проблеми.

Області застосування для нейронних мереж.

- Економіка та бізнес: прогноз ринку, автоматична торгівля, оцінка кредитного ризику, прогноз неплатоспроможності, оцінка вартості майна, виявлення та заниження компаній, автоматичний рейтинг, оптимізація портфеля, оптимізація товарів і грошових потоків, автоматичне зчитування чеків та операцій, безпека на пластикові картки.

- Медицина: медична візуалізація, моніторинг пацієнтів, діагностика, факторний аналіз ефективності лікування, очищення показань приладу від шуму.

- Авіоніка: автопілот учнів, виявлення радіолокаційного сигналу, адаптивне управління важко пошкодженим літальним апаратом.

Зв'язок: стиснення відео, швидке кодування / декодування, оптимізація стільникової мережі та схеми маршрутизації пакетів.

- Інтернет: пошук асоціативної інформації, секретарі електронної пошти та користувацькі агенти, фільтрація інформації в push-системах,

- Спільна фільтрація, заголовки новин, цільова реклама, маркетинг по електронній пошті.

- Автоматизація виробництва: оптимізація виробничих процесів, комплексна діагностика якості продукції (ультразвук, оптика, гамма-випромінювання, ...), моніторинг та візуалізація багатовимірної інформації про доставку, попередження аварій, робототехніки.

- Політичні технології: аналіз та узагальнення соціологічних опитувань, прогнозування динаміки оцінювання, виявлення значущих факторів, об'єктивне групування виборців, візуалізація соціальної динаміки населення.

Системи безпеки: системи ідентифікації, розпізнавання голосу, масове розпізнавання, розпізнавання номерів транспортних засобів, аерокосмічний аналіз, моніторинг потоку інформації, виявлення підробок.

- Введення та обробка інформації: перевірка рукописного тексту, розпізнавання підписів, відбитки пальців та обробка мови. Введіть у свій комп’ютер фінансові та податкові документи.

- Геологічна розвідка: аналіз сейсмічних даних, асоціативних методів розвідки корисних копалин, оцінка ресурсів родовищ.

Існує кілька широко використовуваних комерційних програмних пакетів для нейронних мереж (Statistica Neural Networks, NeuroShell, Matlab Neural Network Toolbox, NeuroSolutions, BrainMaker). Існує ще багато спеціалізованих, некомерційних або досліджених дослідників для власних потреб у нейропрограмах.

Нейронні мережі широко поширені в наші дні: Нейронні мережі - це не що інше, як новий інструмент аналізу даних. І найкраще, що це може викликати фахівця своєї галузі. Основними перешкодами для поширення нейротехнологій

є нездатність широкого кола професіоналів сформулювати свої проблеми так, щоб було можливим просте рішення нейронних мереж.

Основними цікавими особливостями нейронних мереж є:

Наявність алгоритмів для швидкого навчання: Навіть при сотнях входів і десятках тисяч опорних ситуацій нейронну мережу можна швидко навчити на звичайному комп'ютері. Тому нейронні мережі мають широкий спектр застосувань і дозволяють вирішувати складні проблеми прогнозування, класифікації або діагностики.

Можливість роботи за наявності великої кількості неінформативних вхідних сигналів шуму - попередній скринінг не потрібен, сама нейронна мережа визначає, що вона непридатна для завдання, і може явно її відхилити.

Здатність працювати з корельованими незалежними змінними з дискретно-значущою, кількісною та якісною інформацією, що часто ускладнює статистичні методи

Нейронна мережа може вирішити кілька проблем одночасно з одним набором вхідних сигналів. За допомогою декількох виходів можна передбачити значення декількох показників.

Алгоритми навчання ставлять певні вимоги до структури нейронної мережі та властивостей її нейронів. Тому, якщо у вас є спеціальні знання або спеціальні вимоги, ви можете вибрати зовнішній вигляд і властивості нейронів і нейронних мереж, щоб вручну визначити структуру нейронної мережі з окремих елементів і визначити бажані властивості для кожного з них.

1.2 Методи обчислення суми квадратних різниць

Оскільки всі штучні нейронні мережі базуються на концепції нейронів, зв'язків та передавальних функцій, то між різними структурами або

архітектурами нейронної мережі існує подібність. Більшість змін є результатом різних правил навчання. Давайте подивимось на деякі найвідоміші штучні нейронні мережі.

Перцептрон Розенбальта

Перша модель нейронних мереж - це перцептрон Розенбальта. Основою багатьох типів нейронних мереж із штучним прямим поширенням є теорія перцептронів, яку доводиться вивчати.

Одношаровий перцептрон розпізнає найпростіші зображення. Середньозважена сума сигналів вхідних елементів обчислюється через окремий нейрон, який віднімає значення зміщення і передає результат через функцію жорсткого порогового значення, вихід якого однаковий або є. Рішення приймається залежно від значення вихідного сигналу:

- вхідний сигнал належить до класу;

- вхідний сигнал належить до класу.

Області прийняття рішень визначають, які вхідні зображення присвоюються класу та які призначаються класу. Перцептрон, який складається з нейрона, утворює дві найважливіші області, які розділені гіперпланом.

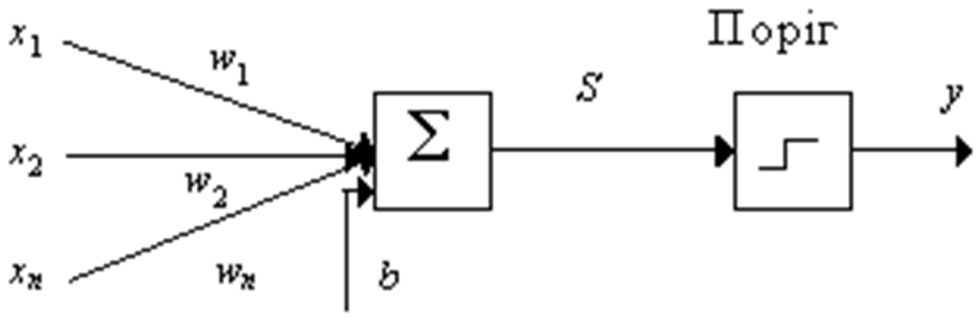

Рисунок 1.2 – Схема нейрона

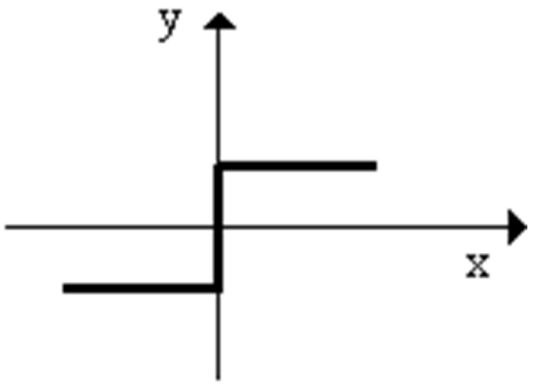

Рисунок 1.3 – Графік передатної функції

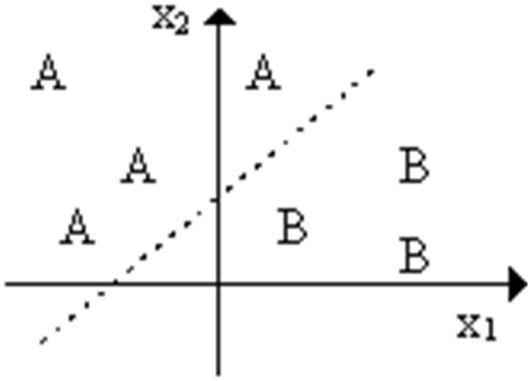

Рисунок 1.4 – Поділяюча поверхня

На рисунку показаний випадок із розмірністю вихідного сигналу - 2.

Інтерфейс - це пряма лінія в площині. Рівняння, яке визначає лінію ділення, залежить від значень синаптичних ваг і переміщення.

Нейромережа заднього розповсюдження

Архітектура FeedForward BackPropagation була розроблена на початку 1970-х кількома незалежними авторами: Werbor; Паркер; Румельхарт, Хінтон та Вільямс. Парадигма VacPropagation є популярною, ефективною та легкою для засвоєння моделлю для складних багаторівневих мереж. ЇЇ Використання в різних типах застосувань призвело до створення великого класу нейронних мереж з різною структурою та методами навчання.

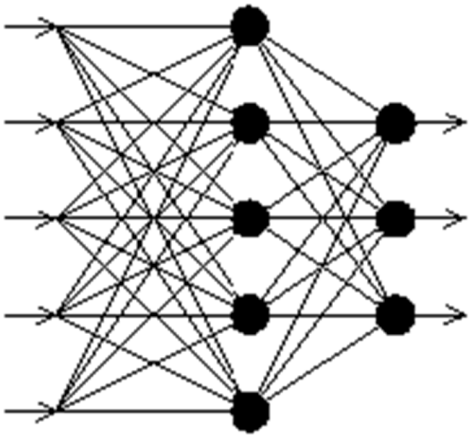

Рисунок 1.5 – Мережа зворотного поширення похибки

Типова мережа BacPropagation має рівень входу, вихідний рівень і принаймні один прихований рівень. Теоретично обмежень щодо кількості прихованих шарів немає, але один або два практично використовуються.

Нейрони організовані в шаровій структурі з прямою (вперед) передачею сигналу. Кожен мережевий нейрон генерує зважену суму своїх входів, передає це значення через функцію передачі та виводить вихідне значення. Мережа може моделювати функцію практично будь-якої складності, кількість шарів та кількість нейронів у кожному шарі, що визначають складність функції.

Визначення кількості прошарків та кількості нейронів у них є важливим аспектом мережевого моделювання. Більшість дослідників та інженерів застосовують загальні правила, зокрема:

- кількість входів і виходів мережі визначається кількістю вхідних і вихідних параметрів досліджуваного об'єкта, явища, процесу тощо;

- зі збільшенням складності між вихідними та бажаними вихідними даними, кількість нейронів у прихованому шарі також повинна збільшуватися;

- якщо модельований процес можна розділити на кілька фаз, необхідні додаткові приховані рівні. Якщо процес не розділений на фази, додаткові рівні можуть зберегти і відповідно зробити неправильне спільне рішення можливим.

Розглянувши всі правила, визначте кількість шарів та кількість нейронів у кожному з них. Потрібно знайти значення для мережевих синаптичних ваг і

порогових значень, які можуть мінімізувати помилку в отриманому результаті. Для цього існують алгоритми навчання, які адаптують мережеву модель до наявних навчальних даних. Переглядаючи мережу всіх прикладів навчання та порівнюючи результати з потрібними значеннями, визначається похибка для конкретної мережевої моделі. Набір помилок створює функцію помилок, яку можна розглядати як мережну помилку. Сума квадратів помилок найчастіше використовується як функція помилок.

Якщо подивитися на поняття поверхні станів, то можна краще зрозуміти алгоритм вивчення зворотного поширення. Будь-яке значення синаптичних ваг і

Порогові значення мережі (параметри вільних номерів моделі) відповідають розмірності в багатовимірному просторі. -це розмір відповідає мережевій помилці. Для різних вагових комбінацій відповідна мережева помилка може бути представлена точкою в просторі, завдяки чому всі ці точки утворюють певну поверхню - поверхню станів. Метою перебування на багатовимірній поверхні найнижчої точки є вивчення нейронної мережі.

Поверхня штатів має складну структуру і досить неприємні характеристики, зокрема наявність місцевих мінімумів (найнижчі точки в певній чверті, але вищі, ніж глобальний мінімум), рівнинних районів, сідлових точок та довгих вузьких ущелин. Неможливо визначити положення глобального мінімуму на поверхні станів аналітичними засобами, саме тому дослідження нейронної мережі по суті є вивченням цієї поверхні.

Враховуючи початкові конфігурації ваг і порогів (від випадково вибраної точки на поверхні), алгоритм навчання поступово знаходить глобальний мінімум. Розраховується вектор градієнта поверхні дефектів, що вказує напрям найменшого падіння на поверхню від заданої точки. Щоб зменшити помилку, потрібно трохи проїхатися. Нарешті, алгоритм зупиняється на нижчій точці, яка може бути лише локальним мінімумом (в ідеалі глобальним мінімумом).

Складність полягає у виборі довжини кроку. Конвергенція швидша з тривалими кроками, але є ризик перестрибнути рішення або піти в

неправильному напрямку. Правильний напрямок розпізнається невеликим кроком, але кількість ітерацій збільшується. На практиці розмір кроку пропорційний нахилу з певною постійною, яка є швидкістю навчання. Правильний вибір швидкості тренувань залежить від заданого завдання і здійснюється емпірично. Ця константа також може залежати від часу і зменшуватися в міру просування алгоритму.

Алгоритм працює ітераційно, його кроки називаються епохами. При вході в мережу всі приклади навчання з відповідної епохи подаються по черзі, початкові значення мережі порівнюються з цільовими значеннями та обчислюється похибка. Значення похибки та градієнта поверхні станів використовуються для корекції ваги і кроки повторюються. Коли певна кількість епох минула, або якщо помилка досягає певного рівня незначності або якщо помилка більше не зменшується (користувач бажано обирає бажаний критерій зупинки), процес навчання припиняється.

Кохоненська мережа

На початку 1980-х рр. розроблена мережа Тойво Кохоненом яка принципово відрізняється від розглянутих вище мереж, відмінісьть полягає в тому що використовується неконтрольоване навчання і навчальна множина складається лише із значень вхідних змінних.

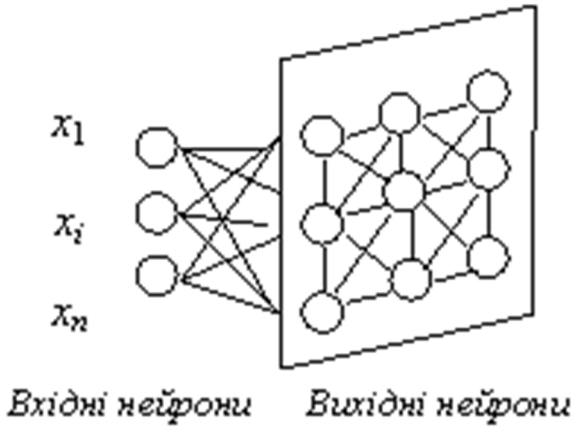

Рисунок 1.6 – Мережа Кохонена

Мережа в навчальних даних розпізнає кластери, а потім розподіляє їх у відповідних кластерах. Якщо наступна мережа зустрічає запис, який не відповідає жодному з відомих прикладів, вона призначає його новому кластеру. Проблеми з класифікацією можуть бути вирішені мережею, якщо дані містять мітки класів. Мережі Kohonen також можна використовувати для завдань, де відомі класи. Перевага полягає у здатності мережі розпізнавати схожість між різними класами.

Мережа Kohonen складається лише з двох рівнів: вхід і вихід. Елементи карти розташовані в кімнаті, як правило, двовимірні. Мережа Кохонена відбувається шляхом послідовного наближення. Під час навчання вводу даних надаються дані, але мережа адаптована не до стандартного вихідного значення, а до закономірностей вхідних даних. З випадкове розташування центрів, починається навчання.

Послідовне введення в мережу навчальних прикладів визначає найбільш схожий нейрон (той, у якого скалярний добуток ваг і вхід у вектор мінімальний). Цей нейрон оголошений переможцем і є центром коригування ваги в сусідніх нейронах. Таке правило передбачає "змагальну" підготовку з урахуванням відстані між нейронами та "переможним нейроном".

Для найкращого відповідності навчальним введенням йдеться не про мінімізацію похибки, а про коригування ваг (внутрішні параметри нейронної мережі).

Основний ітеративний алгоритм Кохонена послідовно проходить через ряд епох, кожна з яких містить вибірку з обробленого навчального зразка. Вхідні сигнали подаються в мережу послідовно, бажані вихідні сигнали не визначаються. Якщо вхідних векторів достатньо, синаптичні ваги мережі можуть ідентифікувати кластери. Терези організовані так, що топологічно близькі вузли реагують на подібні вхідні сигнали.

Центр кластера розміщується у конкретній позиції на основі алгоритму, який виконує приклади кластерів, для яких даний нейрон є "переможцем". В

результаті мережевих тренувань необхідно визначити ступінь сусідства нейронів, тобто нейрона переможця, який представляє кілька нейронів, що оточують нейрон переможця.

Спочатку велика кількість нейронів належить до навколишнього середовища, потім їх розміри поступово зменшуються. Мережа утворює топологічну структуру, в якій подібні приклади утворюють групи прикладів, тісно розташованих на топологічній карті.

Мережа Хопфілда

У 1982 році Джон Хопфілд вперше представив свою асоціативну мережу в Національній академії наук. З цієї причини мережева парадигма на честь Хопфілда, а новий підхід моделювання називається мережею Хопфілда. Мережа Хопфілда заснована на аналогії фізики динамічних систем. Початковий Використання мережі включало асоціативне зберігання на основі вмісту та вирішення проблем оптимізації.

Мережа Хопфілда використовує три шари: вхідний, шар Хопфілд і вихідний шар. Кількість нейронів у кожному шарі однакова. Виходи нейронів вхідного шару подаються на входи відповідних нейронів шару Хопфілда. Тут посилання мають фіксовану вагу. Виходи шару Хопфілда з'єднані з входами всіх нейронів шару Хопфілда, крім них самих і до відповідних елементів у вихідному шарі. Мережа передає дані з вхідного рівня на рівень Хопфілда, передача відбувається під час тренінгу. Поки певна кількість циклів не буде завершена, шар Хопфілда вібрує, і робочий стан сигналів нейронів шару переноситься на вихідний шар. Цей статус відповідає зображенню, яке зберігається в Інтернеті.

Для мережевих тренувань і вхідний, і вихідний рівень повинні відображатися одночасно. Рекурсивний характер шару Хопфілда надає можливість виправити всі ваги спільного вагу. Відповідні пари вводу / виводу повинні бути різними для належної підготовки мережі.

Коли мережа Hopfield використовується як сховище вмісту, є два основних обмеження.

Суворе обмеження кількості зображень, які можна зберегти та точно відтворити. Якщо збережено занадто багато зображень, мережа може збігатися з новим неіснуючим зображенням, відмінним від усіх запрограмованих зображень, або може взагалі не збігатися. Близько 15% кількості нейронів становить межу ємності мережі в шарі Хопфілда.

Шар Хопфілда може стати нестабільним, якщо приклади тренувань занадто схожі. Нестабільна картина зображення вважається, якщо вона застосовується в нульовий час і мережа ідентична іншому зображенню в навчальному наборі. Цю проблему можна вирішити, вибравши більше ортогональних прикладів навчання.

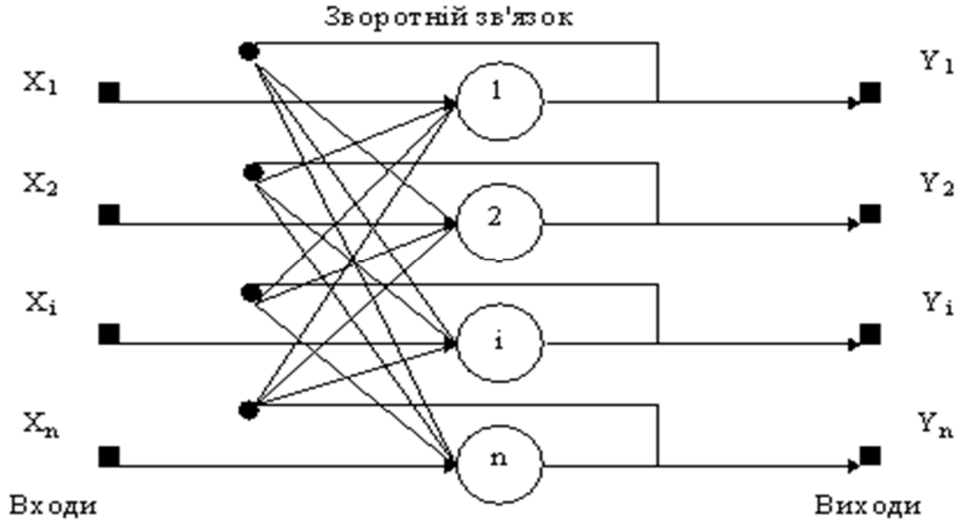

Рисунок 1.7 – Мережа Хопфілда

Існує ряд бінарних сигналів (зображення, аудіо, інші дані, що описують об'єкти або властивості процесів) для вирішення проблеми асоціативної пам'яті, що вважається зразковим. Мережа повинна бути в змозі ідентифікувати відповідний зразок ("отримати" з частковою інформацією) або "зробити висновок", що вхід не відповідає жодному з вибірок із шумового сигналу, застосованого до його входу.

Загалом кожен сигнал може бути описаний вектором - кількістю нейронів у мережі та значенням вхідного та вихідного векторів. Кожен елемент такий же,

як або. Використовуйте вектор, який описує і-й зразок або його компоненти, щоб позначити кількість зразків. Якщо мережа виявляє (або "запам'ятовує") зразок на основі даних, які вона надає, вихід включає його. Н. Де вектор вихідних значень мережі: інакше вихідний вектор не відповідає вибірці.

Якщо, наприклад, сигнали знаходяться на конкретному зображенні, графічне зображення даних з мережевого виходу дає вам зображення, повністю ідентичне зображенню в моделі (у разі успіху) або "вільній імпровізації" мережі (якщо не вдасться). ,

Мережа Хемінга

Мережа Хопфілда - це розширення мережі Хеммінг. Ця мережа була розроблена Річардом Ліппманом у середині 1980-х.

Мережа Хемінга реалізує класифікатор на основі найменшої помилки для бінарних вхідних векторів, при цьому похибка визначається відстані Хемінга. Кількість бітів, які відрізняються між двома відповідними вхідними векторами фіксованої довжини, визначається як відстань Хемінга. Вхідний вектор - це галасливий приклад зображення, інший - спотворене зображення. Вихідний вектор навчального набору - це вектор занять, до яких належать зображення. У режимі навчання вхідні вектори поділяються на категорії, для яких відстань між вхідними векторами сканування та вектором, що рухається, є мінімальним.

Мережа Хемінга складається з трьох рівнів: вхідний рівень з кількістю вузлів, скільки існує окремих двійкових символів; рівень категорії (рівень Хопфілда) з кількістю вузлів, скільки категорій або класів існує; Рівень джерела, який відповідає кількості вузлів на рівні категорії.

Мережа - це проста, зрозуміла архітектура з початковим рівнем, повністю пов'язаним із рівнем категорії. Кожен нейрон на рівні категорії пов'язаний з кожним нейроном на одному рівні і безпосередньо пов'язаний з вихідним нейроном. Конкуренція створює вихід з рівня категорії до початкового рівня.

Навчання в мережі Хемінга схоже на методологію Хопфілда. Бажане тренувальне зображення подається на вхідний рівень, а вихід бажаного рівня

отримує значення бажаного класу, до якого належить вектор. Вихід містить лише значення класу, до якого належить вектор введення. Рекурсивний характер шару Хопфілда надає можливість виправити всі ваги спільного вагу.

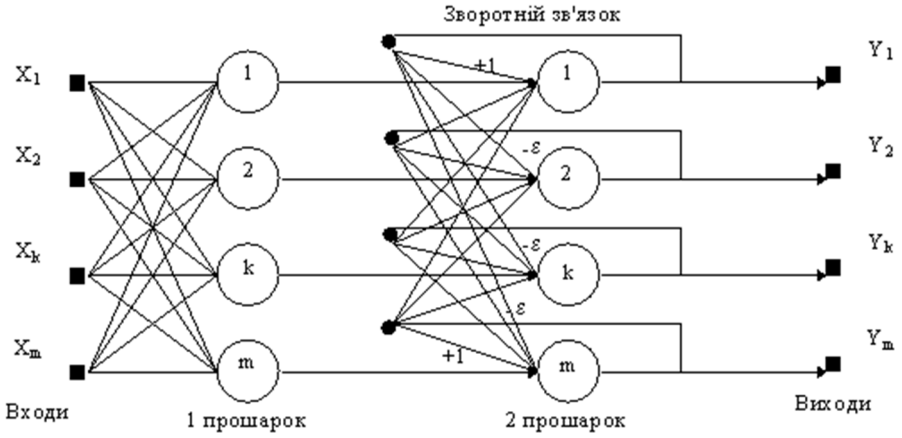

Рисунок 1.8 – Мережа Хемінга

1.3 Структури апаратних засобів обчислення суми квадратних різниць

Принципи обміну інформацією повинні враховуватися при впровадженні замісних пристроїв. Обмін інформацією може бути синхронним та асинхронним. Вибір принципу обміну є важливим аспектом, який впливає не тільки на пропускну здатність, але і на фізичну довжину каналу обміну та кількість пристроїв, які можуть підключитися.

Принцип синхронного обміну

Обмін даними передається блоками за цим принципом. Передача даних базується на координації таймерів передавача та приймача. Синхронізаційні сигнали визначають певний часовий інтервал, протягом якого зчитується інформація з каналу комутації. Для запуску синхронізації використовуються спеціальні сигнали синхронізації. Розташування адресних та даних сигналів відносно сигналів синхронізації визначається зафіксованим протоколом.

Додаткової логіки практично не потрібно, щоб вирішити, що робити далі, тому синхронна передача може бути швидкою і недорогою. Канал обміну використовується більш ефективно для синхронної передачі, високої ефективності, швидкості передачі даних та вбудованого надійного механізму виявлення помилок. Основним недоліком є те, що через проблему спотворення часу шини не можуть довго працювати в синхронній передачі, а інтерфейсне обладнання дорожче та складніше.

Принцип асинхронного обміну

Передача обмежена часом. Можуть виникнути додаткові затримки в передачі даних, оскільки немає зв'язку між часом доступу до передавача та часом, коли він готовий для виведення даних. Інформація про службу записується на початку та в кінці кожного блоку даних. Цей принцип полегшує підключення різноманітних властивостей пристрою до шини, і цей принцип має кращий захист від перешкод. Не турбуючись про спотворення сигналів синхронізації збільшити довжину шини. Недоліком є те, що порівняно з синхронною передачею швидкість передачі даних є повільною через велику кількість переданої ділової інформації. Принцип асинхронної передачі даних підходить для систем, в яких обмін даними не є безперервним і не вимагає високої швидкості передачі даних.

Аналіз можливостей вирішення конфліктів у пристрої заміни.

Якщо кілька пристроїв у системі одночасно передають дані на пристрій заміни, можуть виникнути конфлікти. Існує кілька хитрощів для вирішення цих конфліктів: призначення унікального пріоритету для кожного пристрою, призначення фіксованих часових інтервалів для кожного пристрою та використання багатопортової пам’яті.

Вирішення конфлікту з декількома портами.

Вбудована пам'ять має незалежні набори адрес, даних та шини управління, які дозволяють одночасно та незалежно отримувати доступ до пам'яті пристрою. Ця властивість може значно спростити створення складних систем, але можуть

виникнути конфлікти і використовувати методи їх вирішення. Конфлікти виникають, коли кілька активних пристроїв отримують доступ до однієї комірки пам'яті під час запису даних з декількох портів або під час запису через один порт і читання через інший. Існують різні способи вирішення конфліктів у синхронній та асинхронній багатопортовій пам'яті (PDU). Синхронний блок живлення дає можливість взаємної синхронізації активних пристроїв із системним таймером, так що додаткова логіка не використовується для вирішення конфліктних ситуацій. Асинхронні джерела живлення вирішують конфлікти, використовуючи арбітражну логіку, семафори, запити переривання та майстер / підлеглий.

1.4Постановка завдання на дипломну роботу

Залежно від схем розподілу пам’яті, існує кілька способів створення пристроїв спільного використання: спільне зберігання та спільне зберігання. Існує дві основні моделі обміну: обмін повідомленнями (для спільної пам'яті) та спільна пам'ять.

Спільна пам'ять

Виділяючи спільне сховище для всіх вхідних портів, ви можете ефективно використовувати його відповідно до типу вхідного потоку. У цьому випадку всі важливі обчислювальні вузли підключаються до пам'яті за допомогою загальної шини. Переваги цього підходу полягають у тому, що обмін здійснюється за допомогою запису / зчитування інформації з осередків спільної пам’яті, до яких можна отримати доступ до всіх вузлів, і тому часу не витрачається на переадресацію даних. До недоліків можна віднести можливість конфліктів при одночасному доступі до однієї комірки пам’яті, проблему повільного доступу до оперативної пам’яті та її обмеженої ємності та проблему масштабованості. Чим більше вузлів використовується, тим вищі витрати на розробку і нижчий ККД.

Системи карт з однорідним та неоднорідним доступом до пам'яті. У однорідних системах всі вузли можуть одночасно отримувати доступ до пам'яті. Цей тип організації обміну найчастіше використовується для створення пристроїв заміни. У системах з неоднорідним доступом до пам'яті кожен вузол виділяє частину спільної пам'яті. Ця пам'ять має єдиний адресний простір, тому ви можете безпосередньо отримати доступ до спільної комірки пам'яті через її адресу. Час доступу до модулів спільної пам'яті з різних вузлів різне.

Виділена пам'ять.

У цьому випадку кожен вузол має доступ до певної фіксованої кількості виділених комірок пам'яті. Вузли з'єднані каналами зв'язку для забезпечення обміну інформацією. Пакет, призначений для конкретного виходу вузла, втрачається, якщо блок пам'яті, виділений для цього виходу вузла, переповнюється, хоча інші блоки можуть бути порожніми на даний момент. Система обмінюється зв'язком та отриманням повідомлень. Ця схема розподілу пам’яті використовується для завдань, які потребують невеликого обміну даними та багато пам’яті.

До переваг можна віднести масштабованість, а це означає, що ви можете комбінувати велику кількість вузлів, не знижуючи істотно ефективності їх взаємодії. Вартість системи пропорційна кількості вузлів. До недоліків можна віднести проблему обміну даними та високе споживання енергії. Обмін даними в таких системах дуже повільний порівняно зі швидкістю обчислень (і з великими затримками). Тому неможливо ефективно вирішувати завдання в цих системах, які потребують інтенсивного обміну. Проблема:

1. аналіз існуючих рішень в області обчислення суми квадратних різниць;

2. розробіть блок-схему апаратного пристрою для обчислення суми квадратних різниць.

2 АЛГОРИТМИ І ПРОЦЕСОРНІ ЕЛЕМЕНТИ ДЛЯ ОБЧИСЛЕННЯ СУМИ КВАДРАТНИХ РІЗНИЦЬ

2.1 Формування вимог до пристроїв обчислення суми квадратних різниць

Алгоритми визначення максимального та мінімального чисел для НВІС-реалізацій повинні забезпечувати детерміноване переміщення даних, бути добре структурованими та орієнтованими на реалізацію на множині взаємозв'язаних процесорних елементів (ПЕ). Від вимог, що висуваються до часу реалізації алгоритму залежить структура та операції, які виконують ПЕ. Переважно базові операції алгоритмів обчислення максимального та мінімального чисел реалізує ПЕ. Потрібно одночасно враховувати багато взаємопов'язаних факторів при розробці або виборі алгоритмів обчислення максимального та мінімального чисел.

Алгоритми повинні були рекурсивними та локально залежними. Всі ПЕ повинні виконувати приблизно однакові операції в рекурсивному алгоритмі. Кожний із ПЕ буде повторювати виконання фіксованого набору операцій над послідовністю даних, що надходять. Ефективність відображення алгоритму на ПЕ безпосередньо пов'язана зі способом декомпозиції розв'язання задачі та перетворення на незалежні базові операції, що виконуються паралельно, або на залежні, що виконуються у конвеєрному режимі.

В декартовій системі координат можна утворити точкові системи (решітки) з множини ПЕ, які є моделлю паралельних апаратних структур [23]. Така модель дозволяє оцінити часову та апаратну складність реалізації алгоритму. В решітковій моделі кожному ПЕ ставиться у відповідність часовий і та просторовий j індекси, які вказують коли і де виконується кожна із операцій алгоритму. В алгоритмах з локальними пересилками даних різниця між просторовими індексами j на кроці рекурсії обмежена деякою константою, оскільки в таких алгоритмах обміни здійснюються тільки між сусідніми ПЕ. До

класу алгоритмів з глобальними зв'язками відносяться алгоритми, які при рекурсії мають рознесені просторові індекси.

Забезпечення високої швидкодії є однією з основних вимог, що ставиться до пристроїв визначення максимального та мінімального чисел із масиву чисел. Проблема виникає, при використанні пристроїв для розв'язання задач в реальному часі, який накладає певні обмеження на процес визначення максимального та мінімального чисел із масиву чисел. В першу чергу, ці обмеження пов'язані з часом розв'язання задачі T_p, який не повинен перевищувати часу обміну повідомленнями $T_{обм}$, тобто:

$$T_p \le T_{обм} \tag{2.1}$$

Час обміну залежить як від обсягу масиву N;

розрядності n і частоти надходження вхідних даних F_d, так і від кількості k каналів та їх розрядності n_k. Такий час визначається за формулою:

$$T_{обм} = \frac{Nn}{F_d k n_k} \tag{2.2}$$

Одним із основних інтегральних параметрів оцінки НВІС-пристроїв обчислення максимального та мінімального чисел із масиву чисел є ефективність використання обладнання, який враховує кількість виводів інтерфейсу, однорідність структури, кількість і локальність зв'язків, зв'язує продуктивність з витратами обладнання та дає оцінку елементам пристрою за продуктивністю [23]. Кількісна величина ефективності використання обладнання визначається так:

$$E = \frac{R}{t_o(k_1 \sum_{i=1}^{s} W_{ПЕ_i} d_i + k_2 Q + k_3 Y)} \quad (2.3)$$

де R - складність алгоритму обчислення максимального та мінімального чисел;

t_o - час обчислення максимального та мінімального чисел;

$W_{ПЕ_i}$ - витрати обладнання на реалізацію i-го процесорного елемента;

d_i - кількість функціональних вузлів i-го типу;

k_1 - коефіцієнт врахування однорідності $k_1 = f(s)$;

s - кількість видів функціональних вузлів;

Q - загальна кількість зв'язків;

k_2 - коефіцієнт врахування регулярності зв'язків $k_2 = f(\Delta j)$;

Δj -просторова зв'язкова віддаіль;

Y - кількість виводів інтерфейсу;

k_3 - коефіцієнт врахування кількості виводів інтерфейсу зв'язку $k_3 = f(Y)$.

При апаратній реалізації алгоритмів визначення максимального та мінімального чисел із масиву чисел висока ефективність використання обладнання досягається узгодженням інтенсивності надходження даних $P_d = knF_d$:

де k - кількість каналів надходження даних;

n - розрядність каналів надходження даних;

F_d - частота надходження даних, із інтенсивністю обчислень (обчислювальною здатністю) апаратних засобів, яку визнають так[37]:

$$D_k = \frac{mn_m}{T_k} \quad (2.4)$$

де m - кількість каналів надходження даних у сходинках конвеєра;

n_m - розрядність каналів надходження даних у сходинках конвеєра;

T_k - такт конвеєра.

Задача розроблення пристроїв для визначення максимального та мінімального чисел із масиву чисел, орієнтованих на НВІС-реалізацію, з високою ефективністю використання обладнання зводиться до мінімізації апаратних затрат, кількості виводів інтерфейсу, збільшення однорідності

Структура та регулярність спілкування в режимі реального часу. Доцільно базувати розробку таких пристроїв на наступних принципах, щоб повністю використовувати переваги сучасної технології VLSI та відповідати цим вимогам [23-38]:

- рівномірність і регулярність структури;

Локалізація лізаці посилань та спрощення зв’язків між елементами;

- модульна структура;

- конвеєрний та просторовий паралелізм обробки даних;

- узгодження інтенсивності даних з обчислювальною інтенсивністю пристрою;

Програмування архітектури пристрою за допомогою інтегральних схем з програмованою логікою.

2.2 Вибір принципів та елементів елемента для апаратної реалізації обчислення суми квадратних різниць

Пропонується розробити нейроорієнтовані комп'ютерні системи на основі інтегрованого підходу, який включає:

- сучасні елементи бази, апаратного та програмного забезпечення;

- нейромережеві методи та алгоритми;

Методи обчислення, алгоритми та структури VLSI для здійснення основних операцій нейро-алгоритмів.

Основою для деталізації моделі нейронного елемента може стати встановлення нових фактів у галузі нейрофізіології, зокрема [5]:

1. наявність декількох ділянок із синаптичним контактом;

2. дихотомічне розгалуження дендритів різного порядку відповідно до логічних посилань «Я», «АБО», «Ексклюзивне АБО», призначення максимального або мінімального сигналу в технічних аналогах;

3. Різні діаметри стовбурових дендритів, що безпосередньо примикають до тіла нейрона, і діаметр визначають важливість інформації, що протікає через дендрит.

4. наявність «доріжок» на поверхні соми, що простягаються від головних дендрітів стебла до аксона, що обумовлює існування паралельних шляхів обробки інформації та дає можливість використовувати логічні операції над сигналами, що походять від різних кореневих дендритів;

5. Функціональні особливості аксонального горбка; Сам горбок аксона визначає передавальну функцію нейрона, яка має набагато складнішу форму, ніж сигмоїдальні або лінійні функції передачі, що використовуються в нейронних мережевих технологіях.

6. наявність дихотомічної гілки аксона; вузли розгалуження контролюють проходження сигналу, що залежить від співвідношення діаметрів різних гілок аксона; У математичному моделюванні ці функції можна реалізувати за допомогою логічних операцій.

7. Наявність аксосомного зворотного зв’язку, який вже був реалізований під час побудови повторюваних нейронних мереж.

Поглиблені знання структури біологічного нейрона як ефективного інструменту трансформації можна розглядати як джерело фундаментальних ідей та концепцій створення нових парадигм нейронної мережі як для сьогодення, так і для майбутнього.

Необхідно будувати побудову нейроорієнтованих комп'ютерних систем на принципах, що скорочують витрати, час та масштаби їх застосування. Аналіз показує, що ці вимоги можна задовольнити, використовуючи такі принципи проектування:

Змінний склад обладнання, який передбачає наявність обчислювального ядра та змінних спеціалізованих апаратних та програмних модулів, за допомогою яких ядро адаптується до вимог конкретної програми;

- модульність, при якій компоненти нейроорієнтованих комп'ютерних систем розробляються у вигляді модулів, що мають доступ до стандартного інтерфейсу;

- конвеєрний та просторовий паралелізм обробки даних у спеціалізованих апаратних та програмних модулях;

- відкритість програмного забезпечення, яке пропонує можливості

- удосконалення та вдосконалення для максимального використання стандартних драйверів та програмного забезпечення;

- узгодженість та адаптація апаратних та програмних модулів до інтенсивності даних та структури нейро-алгоритмів;

- програмованість архітектури за допомогою використання інтегральних схем з перепрограмованою логікою.

2.3 Алгоритм та структура процесорного елемента для одночасного обчислення суми квадратних різниць

Комп'ютерна система може бути представлена у вигляді постійної частини B - універсального обчислювального ядра і змінної частини V - спеціалізованих модулів, які виконують основні операції нейро-алгоритмів. Структура

нейроорієнтованої комп'ютерної системи показана на рисунку 3, на якому PDS є багатопортовою пам'яттю.

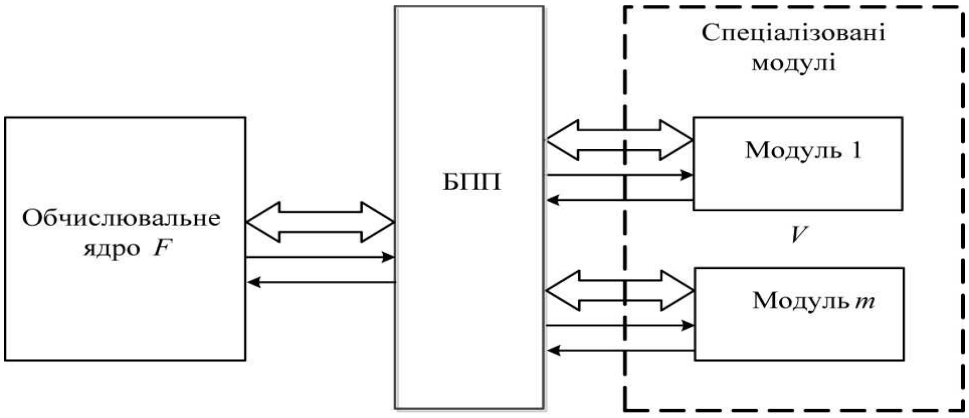

Рисунок 2.1 – Структура нейроорієнтованої комп’ютерної системи

Основними компонентами нейроорієнтованої комп'ютерної системи є обчислювальне ядро, ряд спеціалізованих модулів та PDU. Паралельність обробки даних в нейроорієнтованій комп'ютерній системі ставить вимоги до організації обміну між обчислювальним ядром та низкою спеціалізованих модулів. Такий обмін в неорієнтованій комп'ютерній системі доцільно здійснювати за допомогою BPP, який забезпечує паралельний доступ до великої кількості даних як з обчислювальної ядра, так і з спеціальних модулів. Нейропроцесор (нейрочіп) є основою обчислювального ядра нейроорієнтованої комп'ютерної системи. Він має набір інструкцій, який добре підходить для виконання основних операцій з нейро-алгоритмом, і повний набір інструкцій загального призначення. Переважна більшість алгоритмів нейропарадигми обмежена обмеженою кількістю основних операцій, таких як додавання та множення.

Основними перевагами нейрочіпів є:

Сно відносно швидша продуктивність (порівняно з процесором);

Сприяння реалізації підключень "один до всіх" (для розробників нейронної мережі);

- низьке енергоспоживання;

- відносно доступна ціна.

- основні недоліки:

Структури відрізняються високою складністю конструкції та низькою надійністю системи;

- велика складність ефективного здійснення процесу навчання, самонавчання, самоорганізації;

Значне збільшення споживання енергії та втрата працездатності при одночасному збільшенні ступеня інтеграції нейрочіпів.

- Заздалегідь визначена топологія.

Архітектура та технічні властивості нейропроцесора визначають основні властивості обчислювального ядра нейроорієнтованої комп'ютерної системи. До таких властивостей належать: довжина інформаційного слова; кількість основних інструкцій та час їх виконання; Знімна ємність зберігання Ємність даних та додатків, а також кількість внутрішніх регістрів.

Структура нейроорієнтованих комп'ютерних систем залежить від конкретних вимог та набору нейро-алгоритмів, що використовуються для вирішення проблем. При вирішенні конкретної задачі алгоритми розв’язання задачі розподіляються між пристроями F і V

$$N = N_F + N_V \qquad (2.5)$$

де N_F - множина алгоритмів, які виконуються на обладнанні F;

N_V - множина алгоритмів, які виконуються на обладнанні V.

В залежності від співвідношення N_F і N_V комп’ютерні нейроорієнтовані системи діляться на такі типи:

з переважним використанням процесорного ядра (постійного обладнання F), коли

$$N_F \wedge N, N_V \wedge \wedge N_F >> N_V \quad (2.6)$$

з переважним використанням спеціалізованих модулів (змінної частини V), коли

$$N_F \wedge 0, N_V \wedge N, N_F << N_V \quad (2.7)$$

з рівномірним використанням постійного обладнання F і змінної частини V, коли $N_F \sim \mathrm{N_V}$.

Перший тип нейроорієнтованої комп'ютерної системи відрізняється тим, що основна частина процесорної потужності зосереджена в ядрі процесора.

У другому типі нейроорієнтованих комп'ютерних систем основні алгоритми обчислення реалізуються за допомогою спеціалізованих модулів, а ядро процесора використовується для виконання сервісних функцій.

Третій тип нейроорієнтованої комп'ютерної системи характеризується тим, що ядро процесора забезпечує реалізацію алгоритмів управління, операцій вводу / виводу та функцій обслуговування, а спеціалізовані модулі реалізують автоматизовані нейро-алгоритми, що вимагають великих обчислювальних зусиль.

У більшості випадків обробка даних та реалізація алгоритмів майнінгу даних вимагають нормалізації вхідних даних. Нормалізація - це процедура попередньої обробки вхідних даних (навчальні, тестові та робочі зразки), в якій значення ознак, що утворюють вхідний вектор, зводяться до певного діапазону, як правило, до цього діапазону [0, 1] або [- 1, 1]. Існує багато способів нормалізації вхідних значень. Найпростіший, але в більшості випадків ефективний метод - це лінійна нормалізація. Якщо вихідні дані слід зменшити до діапазону [0, 1], дійте так:

$$x_i^* = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}} \tag{2.8}$$

Для приведення початкових даних до діапазону [-1, 1] лінійна нормалізація здійснюється таким чиом:

$$x_i^* = \frac{x_i}{x_{\max}} \tag{2.9}$$

Якщо вхідні дані щільно заповнюють певний інтервал, лінійна нормалізація є оптимальною, оскільки не потрібно складних обчислень.

Основні операції нормалізації даних - визначення суми квадратних різниць. Щоб забезпечити обробку потоків даних у режимі реального часу, такі операції потребують високої продуктивності, що може бути досягнуто шляхом паралелізації процесу обчислення та апаратної реалізації. Тому актуальною проблемою є розробка апаратно орієнтованих паралельних алгоритмів обчислення суми квадратних різниць.

Відомі способи визначення обчислення суми квадратних різниць з масиву чисел засновані на послідовному зіставленні значень кожного числа, починаючи з другого, з поточним значенням максимального числа, що на першому кроці є значенням першого числа. Якщо значення числа більше поточного значення максимального числа, воно стає поточним максимальним значенням. Тому поточне значення максимального числа містить найбільше значення переданої частини масиву для кожної ітерації обчислення, а в кінці обчислення поточне значення є максимальним числом у всьому масиві [37-38].

Операції з визначення максимальної та мінімальної кількості масиву чисел мають ряд специфічних особливостей, які не дозволяють реалізувати відомі методи та алгоритми обчислення. Відоме обладнання в основному спрямоване на реалізацію алгоритмів з перевагою арифметичних операцій над логічними та

не враховує особливості обчислення суми квадратних різниць. Особливістю сучасного обладнання є неефективність використання багатобітних операційних пристроїв, що значно знижує продуктивність та ефективність використання пристроїв [39-42].

З аналізу літератури ми можемо зробити висновок, що недоліком відомих методів та алгоритмів є те, що вони не зосереджуються на апаратній реалізації та використанні існуючого обладнання обчислення суми квадратних різниць неефективно.

Тому метою роботи є розробка алгоритмів та спеціальних структур VLSI для обчислення суми квадратних різниць з високою ефективністю використання пристроїв.

Паралельні алгоритми та структури VLSI для обчислення суми квадратних різниць. Аналіз методів та алгоритмів для обчислення суми квадратних різниць показав, що алгоритми, засновані на методі порозрядного порівняння, є найбільш ефективними для реалізації VLIS [42]. Розрахунок максимального $A_{\max}$ і мінімального $A_{\min}$ чисел із групи чисел $A_1, A_2, ..., A_j, ...A_m,$ за таким методом виконується послідовним порівнянням розрядів всіх чисел починаючи зі старшого. При кожному порівнянні отримуємо i-і розряди максимального і мінімального чисел, обчислення яких здійснюється за формулами:

$$\overline{A_{i\max}} = \bigwedge_{j=1}^{m} \overline{a_{ji} \wedge y_{ij}}, y_{1j} = 1, \tag{2.10}$$

$$A_{i\min} = \bigwedge_{j=1}^{m} \overline{\overline{a_{ji}} \wedge z_{ij}}, z_{1j} = 1, \tag{2.11}$$

y_{ij}, z_{ij} - i-і розряди j-х слів управління;

a_{ji} - i-розряд j-о числа; m-кількість чисел у групі.

Формування $(i+1)-x$ розрядів j-х слів управління виконується за формулами (2.12) і (2.13) та виконання наступних логічних обчислень :

$$y_{(i+1)j} = (\overline{A}_{i\max} \vee x_{ji}) \wedge y_{ij}, \tag{2.12}$$

$$z_{(i+1)j} = (A_{i\min} \vee x_{ji}) \wedge z_{ij}, \tag{2.13}$$

Процес синтезу паралельних структур VLSI для обчислення суми квадратних різниць з групи чисел виконується для виконання наступних етапів:

Призначення основної операції;

- просторово-часове відображення алгоритму;
- Розробка схеми процесорного елемента (ПЕ), що реалізує базу
- як працює алгоритм;
- синтез структур VLSI на основі ПЕ;
- Організація інтерфейсу VLSI.

Аналіз алгоритмів паралельного обчислення суми квадратних різниць на основі методу бітового порівняння дозволив призначити базову операцію для реалізації VLSI. Ця основна операція включає формування бітів контрольних слів за формулами $y_{(i+1)j} = (\overline{A}_{i\max} \vee x_{ji}) \wedge y_{ij}$ і $z_{(i+1)j} = (A_{i\min} \vee x_{ji}) \wedge z_{ij}$ та виконання наступних логічних обчислень:

$$\overline{A_{ji\max}} = \overline{a_{ji} \wedge y_{ij}}, \tag{2.14}$$

$$\overline{A_{ji\min}} = \overline{\overline{a}_{ji} \wedge z_{ij}}, \tag{2.15}$$

Вибрана основна операція реалізована у вигляді ПЕ, діаграми якої наведені на Рисунку 2.2. Застосовується наступне:

а - єдиний підйомний пристрій РЕ;

б - ПЕ конвеєр;

в - РЕ пристрою з вертикальною обробкою введених номерів.

Особливістю розробленого ПЕ є використання загальних шин результатів, які підключаються до відкритого колектора за допомогою логічних елементів 2I-NOT. Вертикальне підключення до загальних шин призводить до збільшення числового поля, з якого визначаються максимальні та мінімальні значення. Використання спільної шини результатів забезпечує високу продуктивність, незалежну від кількості чисел у масиві, але залежить лише від часу затримки РЕ.

Вартість пристроїв VLSI для обчислення суми квадратних різниць в основному залежить від площі кристала, яка виходить як від вартості пристроїв (кількості транзисторів), так і від кількості зовнішніх з'єднань, кількості який обмежений рівнем техніки та розміром кристала. Вирівнювання структур сортування у реалізації VLSI вимагає зменшення кількості інтерфейсних штифтів та кількості з'єднань між РЕ. Цим вимогам можна задовольнити, використовуючи паралельно-вертикальний алгоритм для обчислення суми квадратних різниць, в яких кількість і вихід результатів здійснюється бітовими фрагментами.

Структура паралельно-вертикального НВІС-пристрою обчислення суми квадратних різниць наведена на Рисунку 2.3,

З метою підвищення ефективності визначення максимальної кількості групи чисел рекомендується паралельно-вертикальний підхід, при якому час визначення максимальної кількості групи чисел не залежить від кількості чисел. При цьому кожен біт роботи над інформаційними входами пристрою отримує бітові фрагменти всіх чисел з найвищих місць.

Структура такого пристрою показана на малюнку 2.2.4.

ТЕ - тактовий вхід;

В - введення в початкову установку тригерів;

x1, ..., xm - одноцифрові введення інформації, де m - кількість порівняних чисел;

ВР1, ..., ВРm - це блоки порівняння;

Т1 і Т2 - тригери D;

Вихід - це результат результату.

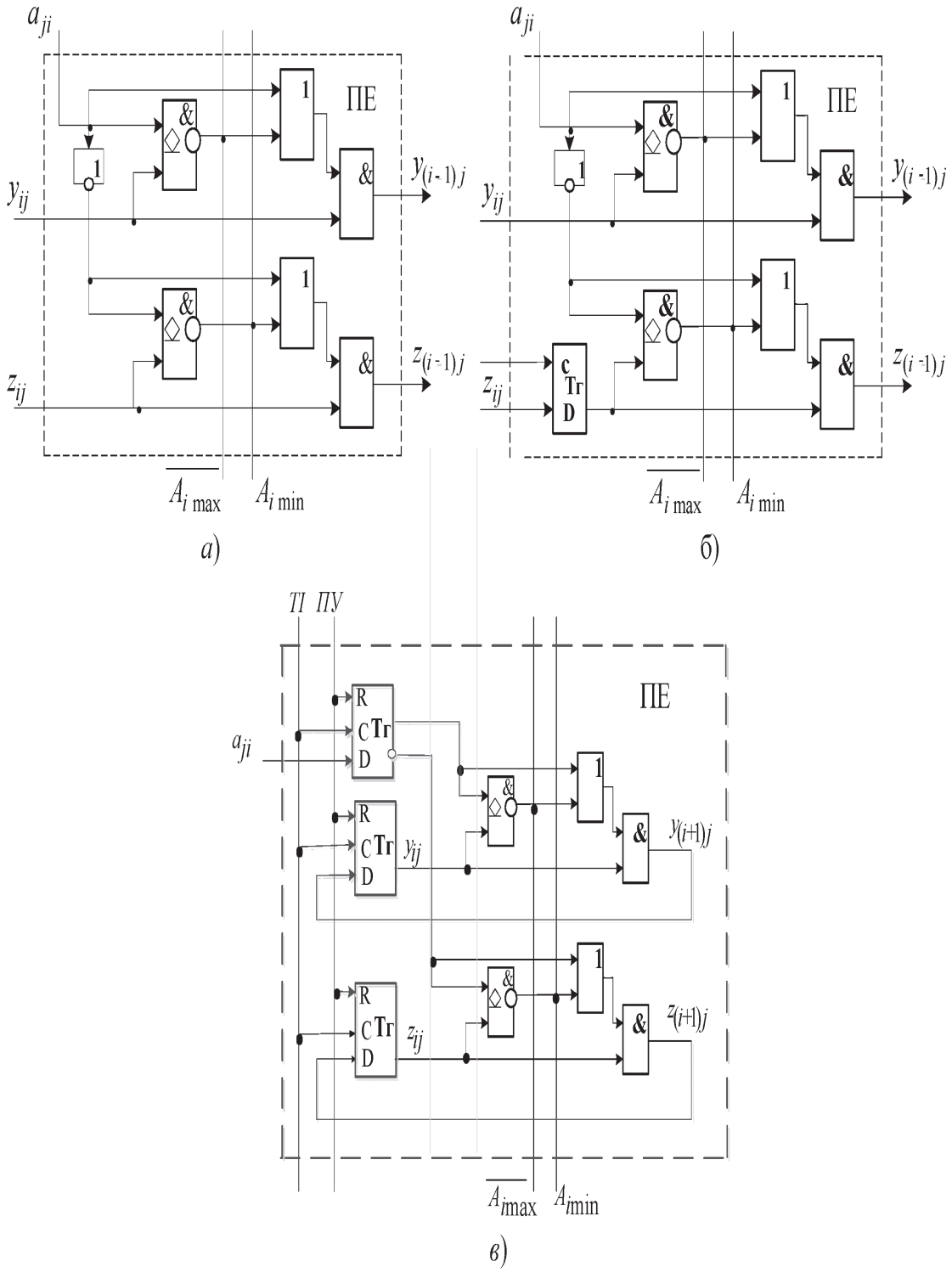

Рисунок 2.2 – Діаграми процесорного елемента

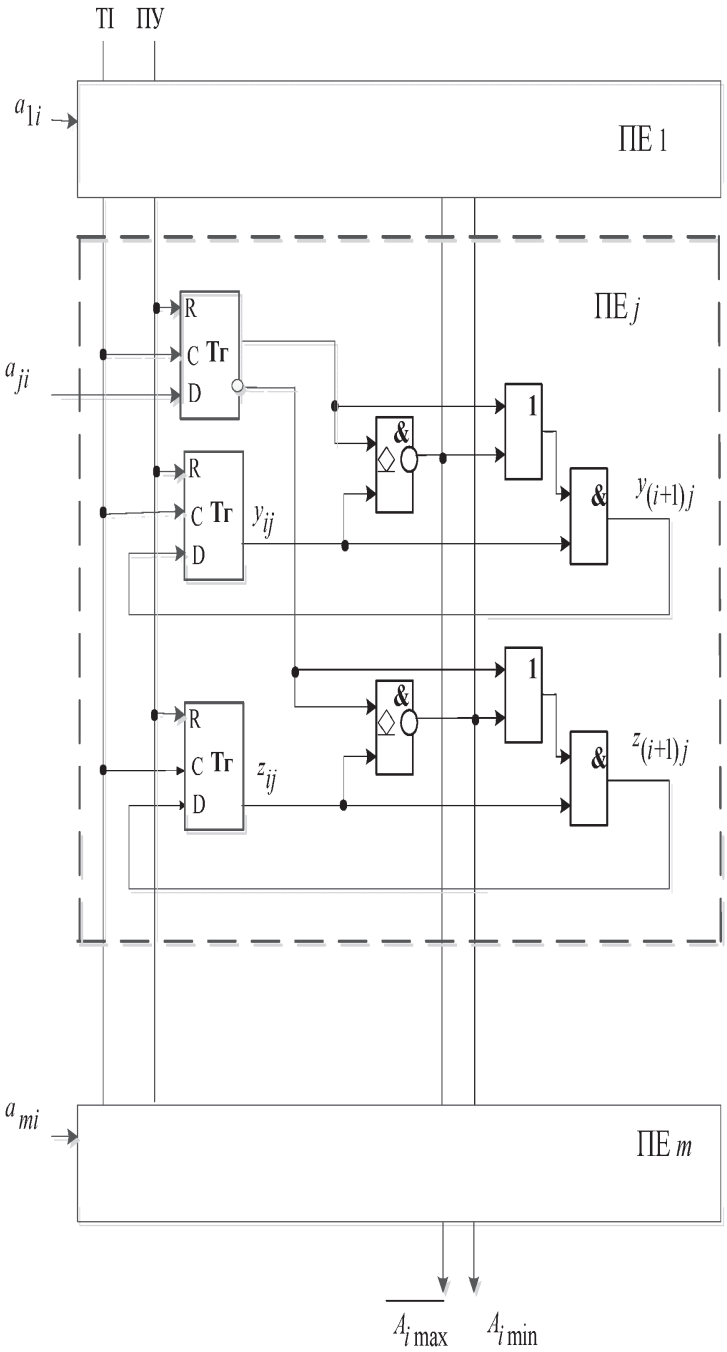

Рисунок 2.3 – Схема паралельно-вертикального НВІС-пристрою обчислення суми квадратних різниць

Пристрій для визначення максимальної кількості групи чисел працює наступним чином.

Перед тим, як імпульс першої установки починається з входу першої установки В, тригери Т1 і Т2 встановлюються в журналі 1 у кожному блоці

порівняння ВРj (j = 1, ..., m). Інформація з виходів тригерів Т1 і Т2 (log 1) у кожному блоці порівняння ВРj задає виходи елементів і журнал 1 сигналу.

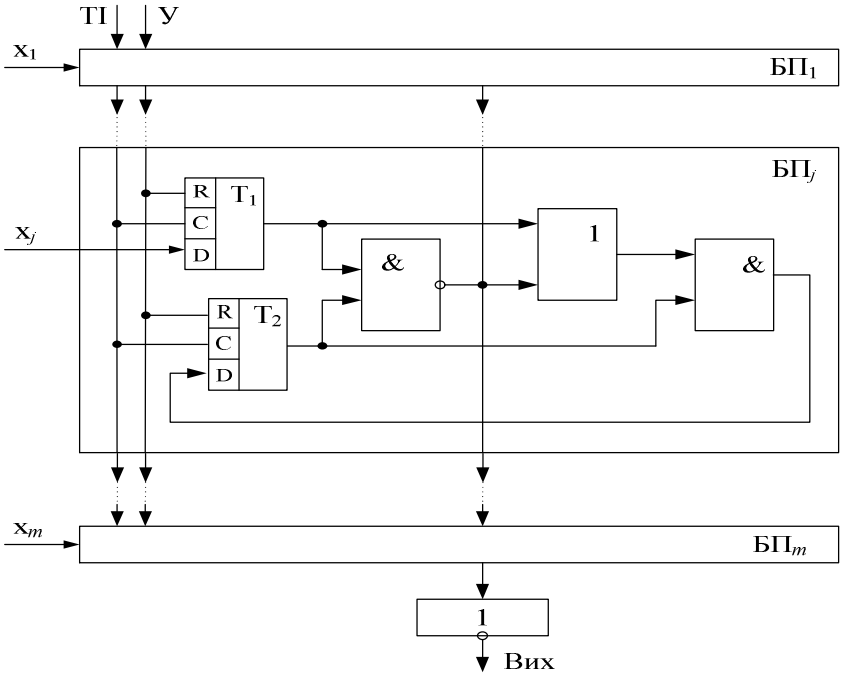

Рисунок. 2.4 – Структура пристрою визначення максимального числа з групи чисел

У першому циклі кожного блоку порівняння ВРj інформаційний вхід тригера Т2 від елемента І отримує значення log 1, а вхід інформації тригера Т1 від одноцифрового вводу інформації xj отримує значення найбільшого числа Jo числа. Перший тактовий імпульс у кожному блоці порівняння ВРj в тригері Т1 записує найбільшу цифру j-о, а тригер Т2 записує log 1, який дозволяє j-о брати участь у визначенні максимального значення. У цьому випадку у кожному блоці порівняння ВРj значення найбільшого розряду j-го числа з виходу тригера Т1 надходить на перший вхід елемента І-NOT з відкритим колектором та першим входом елемента АБО. Журнал 1 з виходу тригера Т2 (керуючий сигнал) подається на другий вхід елемента І-NOT з відкритим колектором і на другий вхід елемента І. І мова йде про другий вхід елементів АБО до цих одиниць порівняння, а вхід елемента НЕ. Якщо вищі цифри порівнюваних чисел дорівнюють нулю, на вході елемента утворюється NOT 1, а в інших випадках log

0. Інформація з виводу елемента NOT (найвища цифра максимальної кількості) подається у висновок результату. Якщо log 1 на другому вході елементів АБО блокує порівняння ВР1, ..., ВРm на їх виходах, сигнал log1 і на log.0, встановлюється інформація з виходів тригера Т1. Інформація з виходів елементів АБО з виходів елементів І генерує керуючі сигнали, які подаються на інформаційні входи Т2.

Інформація про другий тактовий імпульс з однозначного вводу інформації xj (наступна цифра) записується в тригер Т1 кожного блоку порівняння ВР, і тригер Т2 записує значення з виходу елемента І.

Це дає змогу (log.1) або забороняє (log.0) участь інформації з одноцифрового введення інформації xj у подальшому формуванні максимальної кількості.

Формування другої та наступних цифр результату і керуючих сигналів відбувається так само, як і в першому циклі.

Час визначення максимальної кількості групи чисел у цьому пристрої залежить від кількості цифр, а не від їх кількості. Для n циклів, де n - кількість цифр, отримуємо максимальне число з чисел групи m.

Час обчислення максимальної та мінімальної кількості з масиву рівнянь:

$$t_m = (t_{Tг} + 3t_I)n, \tag{2.16}$$

$t_{Tг}$ - час запису інформації у тригер;

t_I - час затримки інформації при проходженні через логічні елементи типу *АБО, І, І-НЕ.*

Затрати обладнання на реалізацію даного пристрою рівні:

$$W_m = (3W_{Tг} + 6W_I)m \tag{2.17}$$

W_{T2} - затрати обладнання на реалізацію D-тригера;

W_I - затрати обладнання на реалізацію логічних елементів типу *АБО, І, І-НЕ.*

Збільшення необхідної потужності обробки за рахунок збільшення швидкості передачі бітів каналу даних, яка може бути дві або більше, потрібно для обробки більш інтенсивних потоків даних. Максимальна продуктивність досягається, якщо швидкість передачі даних у каналах даних однакова, тобто якщо всі цифри обробляються одночасно.

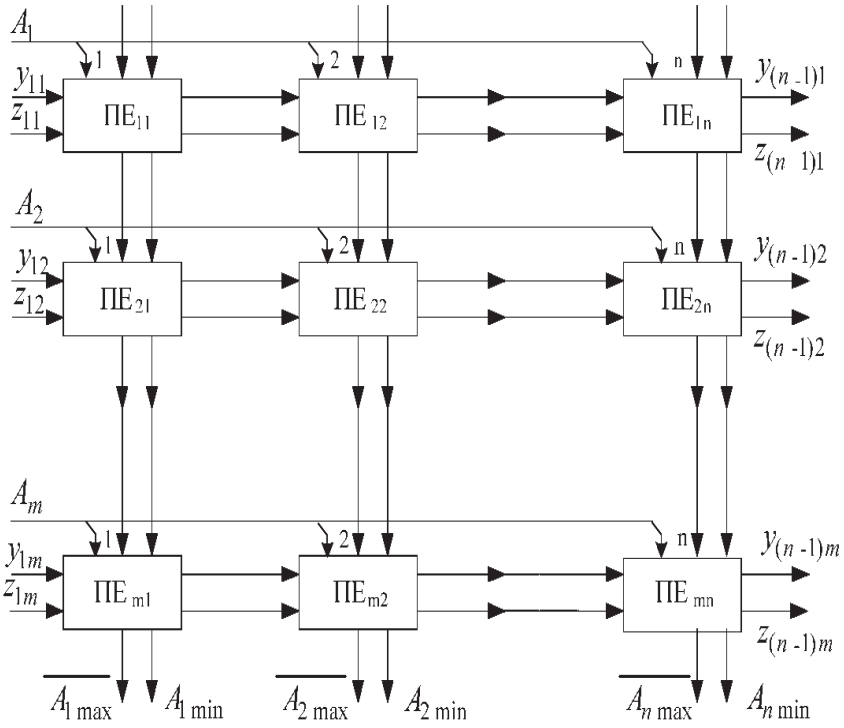

Рисунок 2.5 – Матрична однотактна НВІС-структура пристрою обчислення суми квадратних різниць m чисел

На основі однотактного і конвеєрного ПЕ синтезовано відповідно матричні однотактні (рис.2.3.4) та конвеєрні (рис.2.3.5) паралельні НВІС-структури пристрою обчислення суми квадратних різниць. Кожна із цих структур складається із матриці $(m \times n)$ ПЕ, в якій $ПЕ_{1i}, -ПЕ_{mi},$ - i-го стовпчика

обчислюють відповідно до формул $\overline{A_{i\max}} = \bigwedge_{j=1}^{m} \overline{a_{ji} \wedge y_{ij}}, y_{1j} = 1,$ і $A_{i\min} = \bigwedge_{j=1}^{m} \overline{\overline{a_{ji}} \wedge z_{ij}}, z_{1j} = 1,$ значення i-х розрядів максимального $\overline{A_{\max}}$ і мінімального $A_{\min}$ чисел та формують у відповідності з формулами $y_{(i+1)j} = (\overline{A}_{i\max} \vee x_{ji}) \wedge y_{ij}$, і $y_{(i+1)j} = (\overline{A}_{i\max} \vee x_{ji}) \wedge y_{ij}, z_{(i+1)j} = (A_{i\min} \vee x_{ji}) \wedge z_{ij}$, значення $(i+1) - x$ слів управління $y(i+1)$ і $z(i+1)$.

Час обчислення суми квадратних різниць ь у однотактному пристрої визначається за формулою:

$$T_O = 4nt_i, \tag{2.18}$$

Де t_i - час спрацювання логічного елемента "I", n - розрядність чисел.

Апаратні витрати на реалізацію однотактного пристрою дорівнюють:

$$W_O = 7NnW_i, \tag{2.19}$$

W_i - апаратні витрати на логічні елементи типу І, АБО, І-НЕ.

Особливістю конвеєрної НВІС-структури (рис.2.5) є введення у ПЕ тригерів та використання n блоків пам'яті типу FIFO. Кожний i-ий блок $FIFO_i$,- забезпечує затримку інформації на i тактів. Тривалість конвеєрного такту у такому пристрої визначається за наступною формулою:

$$T_K = t_{Tг} + 4nt_i, \tag{2.20}$$

де $t_{Tг}$ - час спрацювання тригера.

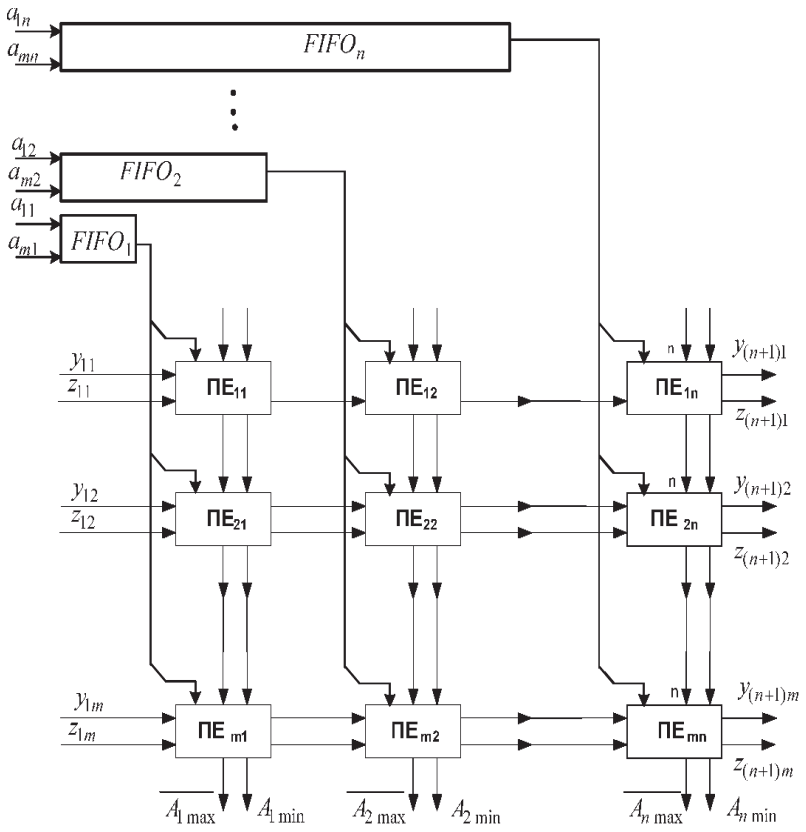

Рисунок 2.6 – Матрична конвеєрна НВІС-структура пристрою обчислення суми квадратних різниць m чисел

З точки зору процесорів Conveyor - це метод, який використовується в розробці для збільшення пропускної здатності інструкцій (кількість інструкцій, які можуть бути виконані за певний проміжок часу). Ідея полягає в тому, щоб розділити обробку комп’ютерної інструкції на ряд незалежних кроків, результати зберігаються в кінці кожного кроку. Таким чином ланцюг управління комп'ютером може отримувати інструкції з найменшою швидкістю обробки. Однак це рішення набагато швидше, ніж виконання всіх цих кроків лише для кожного оператора.

Обробка конвеєра також використовує суперскалярність, архітектуру, яка використовує кілька декодерів інструкцій, які можуть завантажувати багато блоків виконання. Тобто, якщо інструкції, оброблені трубопроводом, не суперечать один одному і не залежать від результату іншого, такий пристрій може виконувати інструкції паралельно.

Витрати на обладнання для реалізації конвеєрного пристрою для обчислення суми квадратних різниць складають:

$$W_K = W_{FIFO} + mn(W_{Tг} + 7w_i), \tag{2.21}$$

W_{FIFO} і $W_{Tг}$ - апаратні затрати відповідно на пам’ять типу FIFO і тригер.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА СИНТЕЗ АПАРАТНОГО ПРИСТРОЮ СОРТУВАННЯ ДАНИХ МЕТОДОМ ЗЛИТТЯ

3.1 Принципи розпаралелювання обчислень

Основний спосіб збільшення швидкості обробки зображень - паралельний процес прийому їжі та використання масово паралельних обчислень з великим обсягом пам'яті. Ці інструменти включають графічний процесорний блок (GPU), який є процесором класу SIMD (Single Instruction Multiple Data). Особливістю процесорів SIMD є те, що кожен з них використовує операцію для обробки набору незалежних даних. Рекомендується використовувати програмне забезпечення для компіляції та виконання міжплатформних CUDA (Compute Unified Device Architecture) для розробки великого програмного забезпечення для сортування, яке виконується частково на процесорі (CPU) і частково на GPU. Тому розробка паралельних алгоритмів та програмного забезпечення для обробки з графічними процесорами є нагальним завданням.

Розробка високопродуктивного програмного забезпечення для обробки GPU може бути забезпечена інтегрованим підходом, який включає:

Дослідження, розробка методів та алгоритмів паралельної обробки;

Адаптація алгоритмів обробки зображень до архітектури GPU-GPU;

автоматизоване програмне забезпечення, що скорочує час і підвищує якість програм [1-4].

Однією з найпоширеніших вимог до обробки зображень є забезпечення високої продуктивності.

Сучасні графічні процесори п'ятого покоління, такі як NVIDIA GeForce 8-GTX 200 та AMD (ATI) HD 2K-5K, містять ряд однакових обчислювальних пристроїв (потокових процесорів), які працюють із спільною пам'яттю GPU. Кількість потокових процесорів та розмір пам’яті GPU можуть змінюватись залежно від моделі GPU. Усі потокові процесори синхронно виконують одну і ту

ж команду, щоб GPU можна було призначити класу SIMD. Командна система потокового процесора містить 32-бітна точність арифметики, керування (гілка та петля) та інструкції з доступу до пам'яті. Через великі затримки інструкції доступу до пам'яті виконуються асинхронно. Тактова швидкість графічного процесора нижче, ніж у процесора комп'ютера. Однак через велику кількість потокових процесорів, що працюють паралельно, продуктивність GPU може досягати декількох TFlops.

Для ефективної реалізації алгоритмів обробки на основі GPU необхідний розподіл простору та часу на операційному рівні потокових процесорів. Просторово-часове призначення алгоритмів обробки повинно забезпечити ідентифікацію всіх форм паралелізму та знайти необхідні просторово-часові рішення для їх ефективної реалізації. Такі вимоги забезпечуються алгоритмом розподілу в ярусно-паралельній формі (PFD). Якщо YFT є поданням алгоритму, розподіл всіх його операторів функцій є FI за рівнем, так що на j-му рівні розміщуються оператори функцій, які залежать принаймні від одного функціонального оператора (j-1) -го рівня і не залежать від Оператори наступних рівнів. Всі функціональні оператори на одному рівні виконуються незалежно. Для ефективної реалізації алгоритмів обробки зображень пропонується представити такі алгоритми у вигляді конкретної потокової діаграми, яка враховує особливості архітектури GPU-GPU. Такий графік створюється в чотири етапи:

1) розкладання алгоритму обробки;

2) організація зв'язку (обмін даними) між операторами функцій;

3) консолідація функціонерів;

4) Планування розрахунків при реалізації алгоритму.

На фазі декомпозиції алгоритм обробки F розбивається на оператори функцій Fi, між якими здійснюються з'єднання, відповідні цьому алгоритму. Розкладання може бути здійснено згідно з функціональним методом розкладання. Використання функціонального методу декомпозиції дає

можливість отримати просторово-часове подання структури алгоритму обробки зображень на основі арифметичних операцій. Результат Перший етап розробки - це графічна схема алгоритму, в якій оператори функцій Fj мають приблизно однаковий час виконання, а їх складність визначається архітектурою GPU-GPU.

На етапі проектування зв'язку необхідно визначити структуру та біт каналів зв'язку між функціональними операторами FI. Що таке перехід від графічної діаграми алгоритму до діаграми потоку, в якій визначено просторово-часове положення та фіксація операторів функцій на рівнях. Структура взаємозв'язків на блок-схемі між операторами функцій сусідніх рівнів Fji визначається кількістю каналів припливу та бітом.

Схемова схема, створена в два етапи, забезпечує оцінку складності обчислювальної алгоритму обробки та визначає необхідну продуктивність універсальних процесорів та графічних процесорів при їх реалізації програмним забезпеченням. Третій етап розвитку - консолідація операцій за рахунок поєднання функціональних операторів FJK та каналів даних як у межах зсуву, так і між зрушеннями. На цьому кроці діаграма потоку адаптується до структури універсальних процесорів та графічних процесорів. Ця фаза тісно пов'язана з фазою планування процесу обробки.

Четвертий етап планових розрахунків - зберігання інформації про структуру заданого алгоритму обробки потокових графіків. На цьому етапі здійснюється планування процесу розрахунку, визначаються затримки та перестановки. Для відтворення процесу обробки даних у вказаних операторах графіку потоку вводяться управління, затримка та перестановка. Існує два основні варіанти консолідації операцій (групування операторів функцій) для отримання конкретної діаграми потокової обробки.

Перший спосіб отримати певний графік алгоритму - це лінійне проектування його на горизонтальну вісь *X*. У цьому випадку консолідація операцій відбувається шляхом інтеграції між шарами функціональних операторів та каналами даних.

Другий варіант - отримати вказану діаграму потоку

Алгоритм - це його лінійна проекція на вертикальну вісь *Y*. У цьому випадку операції консолідуються шляхом об'єднання функціональних операторів та каналів даних як на рівні, так і між рівнями.

В результаті такого об'єднання операцій ми отримуємо певну потокову діаграму, яка базується на архітектурі певного GPU. Наведена діаграма потоку характеризується складністю функціонерів *Fj*, шириною L (числом операторів функцій Fji на рівні діаграми) та висотою h (число рівнів на діаграмі). Слід зазначити, що параметри графіка взаємозалежні і одна зміна призводить до зміни іншої.

Для реалізації алгоритмів обробки GPU ми використовуємо модель програмування, що називається паралелізмом даних. Ця модель програмування заснована на застосуванні операції до кількох даних. Порядок таких операцій є особливістю програми, написаної за цією моделлю.

У розробці програмного забезпечення були використані Nvidia CUDA 5.5, MS Visual Studio 2010 та MS SQL Server Management Studio 2012. Організуйте на ньому складні паралельні обчислення. Графічний процесор, що підтримує CUDA, перетворюється на потужну, програмовану відкриту архітектуру, схожу на сьогоднішні процесори.

Мови, що використовуються для реалізації алгоритмів обробки зображень на GT-GPU Nvidia, були C / C ++, мовою програмування високого рівня, яка підтримує різні парадигми програмування: об'єктно-орієнтовані, узагальнені та процедурні. Особливостями C / C ++ є: продуктивність; масштабованість; мало пам'яті, адрес, портів; Створення узагальнених алгоритмів для різних типів даних, їх спеціалізація та обчислення на етапі компіляції за допомогою шаблонів; Підтримка різних стилів програмування та технологій.

Багато методів обчислення матриці характеризуються тим, що одні й ті ж дії обчислення повторюються для різних елементів матриці. доступний момент вказує на наявність паралелізму за даними, коли виконуються матричні

обчислення, і в результаті паралелізація матричних операцій зводиться в більшості випадків до поділу оброблених матриць між процесорами використовуваної комп'ютерної системи. Вибір методу ділення матриць призводить до визначення конкретного методу паралельних обчислень. Наявність різних схем розподілу даних створює ряд паралельних алгоритмів для матричного обчислення.

Найпоширеніші та найпоширеніші методи поділу матриць - це розділення даних на смужки (вертикальні чи горизонтальні) або на прямокутні фрагменти (блоки).

Діапазон поділу матриці. За допомогою розділення смужок блоків кожному процесору присвоюється підмножина рядків (горизонтальний або горизонтальний розділ) або стовпців (стовпчик за стовпцем або вертикальний розділ) матриці (рис. 3.1). У більшості випадків рядки та стовпці діляться на смуги безперервно (послідовно). Наприклад, для такого підходу, наприклад, для горизонтального макету по рядку, матриця A представлена таким чином (рис. 3.1):

$$A = (A_0, A_1, ..., A_{p=1})^T, A_i = (a_{i_0}, a_{i_1}, ..., a_{i_{k-1}}), \\ i_j = ik + j, 0 \le j < k, k = m / p \qquad (3.1)$$

де, $A_i = (a_{i_0}, a_{i_1}, ..., a_{i_{k-1}}), i_j = ik + j, 0 \le j < k, k = m / p$

- і-й ряд матриці A (передбачається, що число рядків t є кратним кількості процесорів p, тобто t (t = k * p). Дані постійно обмінюються.

Іншим можливим підходом до смуги є використання тієї чи іншої схеми чергування (циклічності) рядків чи стовпців. Як правило, для гніздування використовується кількість процесорів p - у цьому випадку матриця A має вигляд з горизонтальним перегородкою

Іншим можливим підходом до смуги є використання тієї чи іншої схеми чергування (циклічності) рядків чи стовпців. Як правило, для гніздування використовується кількість процесорів p - у цьому випадку матриця A має вигляд з горизонтальним перегородкою

$$A=(A_0, A_1,..., A_{p=1})^T, A_i=(a_{i_0}, a_{i_1},..., a_{i_{k-1}}), i_j = ik + jp, 0 \le j < k, k = m / p \qquad (3.2)$$

Циклічна смугова схема може бути корисною для кращого збалансування навантаження процесорів (наприклад, при розв'язуванні лінійної системи рівнянь методом Гаусса).

Коли розділяється блок-шашка, матриця ділиться на прямокутні набори елементів - поділ зазвичай використовується безперервно. Припустимо, кількість процесорів p = S · q, кількість рядків у матриці кратно s, а число стовпців кратне q, тобто m = k · s і n = l Q. Уявіть собі вихідна матриця A наступна як набір прямокутних блоків:

$$A=\begin{pmatrix} A_{00} & A_{02} & ...A_{0q-1} \\ & ... & \\ A_{s-11} & A_{s-12} & ...A_{s-1q-1} \end{pmatrix}, \qquad (3.3)$$

де A_{ij} – блок матриці, який складається з элементів:

$$A_{ij}=\begin{pmatrix} a_{i_0 j_0} & a_{i_0 j_s} & ...a_{i_0 j_{i-1}} \\ & ... & \\ a_{i_{k-1} j_0} & a_{i_{k-1} j_1} & a_{i_{k-1} j_{i-1}} \end{pmatrix}, \qquad (3.4)$$

$$i_v = ik + v, 0 \le v < k, k = m / s, j_m = jl + u, 0 \le u \le l, l = n / q.$$

При такому підході доцільно, щоб комп'ютерна система мала фізичну або принаймні логічну топологію масиву процесорів з 5 рядками та q стовпцями. У цьому випадку процесори, прилеглі до структури решітки, обробляють сусідні блоки вихідної матриці, коли дані постійно обмінюються. Однак слід зазначити, що рядки та стовпці також можна повертати по черзі на блок-схемі.

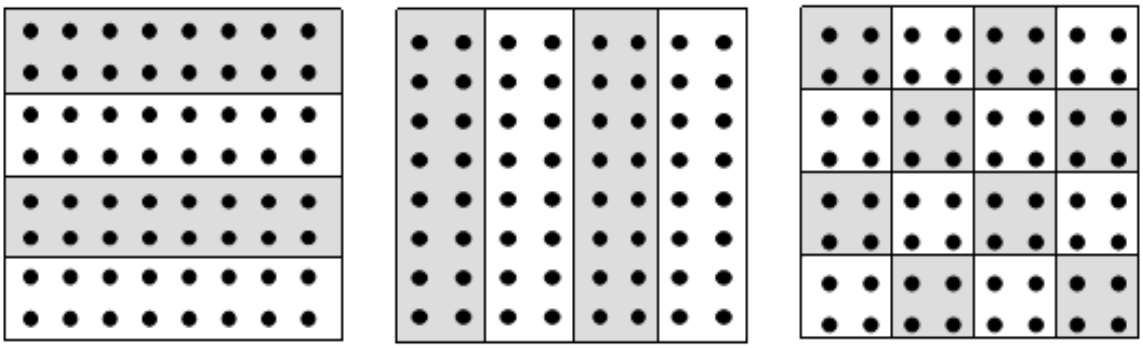

Рисунок 3.1 – Способи поділу елементів матриці між процесорами обчислювальної системи

Розглянуто три паралельні алгоритми множення квадратної матриці на вектор. Кожен підхід базується на різному типі виводу (матричні та векторні елементи) між процесорами. Поділ даних змінює взаємодію процесорів, так що кожен із представлених методів суттєво відрізняється від двох інших.

Послідовний алгоритм множення матриці на вектор може бути представлений наступним чином.

Множення матричного вектора - це послідовність для обчислення точкових добутків. Оскільки будь-яке обчислення крапкового добутку векторів довжини n вимагає n операцій множення і виконання n-1 операцій додавання, його складність знаходиться в порядку O (n). Для виконання множення матричного вектора необхідно виконати t операції для обчислення крапкового добутку, тому алгоритм має складність порядку O (tn).

Поширення даних

Коли виконуються паралельні алгоритми множення матриці на вектор, крім матриці A необхідно відокремити вектор b і векторний результат c. Елементи векторів можна дублювати, тобто всі елементи вектора можна скопіювати у всі процесори, що складають багатопроцесорну комп'ютерну систему, або їх можна розділити між процесорами. При блоковому поділі вектора від n елементів кожен процесор обробляє безперервну послідовність до векторних елементів (ми припускаємо, що розмірність вектора n залишається дільною на кількість процесорів, тобто N = kr).

Подвоєння векторів b і c між процесорами є коректним рішенням (у наступному передбачається, що m = n для спрощення ілюстрації). Вектори b і c складаються з n елементів, тобто вони містять стільки ж даних, скільки рядків або стовпців матриці. Коли процесор зберігає рядок або стовпчик матриці та окремі елементи векторів b і c, загальна кількість збережених елементів знаходиться в порядку O (n). Коли процесор зберігає рядок (стовпець) матриці та всі елементи векторів b і c, загальна кількість збережених елементів також знаходиться в порядку O (n). Таким чином, при дублюванні та діленні векторів вимоги до зберігання є класом складності.

3.2 Пристрій для обчислення суми квадратних різниць за допомогою GPU

CUDA - це паралельна апаратна та програмна архітектура, за допомогою якої обчислювальну потужність можна значно збільшити за допомогою графічних процесорів Nvidia. За допомогою програми CUDA SDK програмісти можуть реалізувати N алгоритмів, які можна виконати на графічних процесорах Nvidia на спеціальному спрощеному діалекті мови програмування C та інтегрувати спеціальні функції в текст програми C. Архітектура CUDA надає

розробникам можливість доступу та керування інструкціями для графічного прискорювача, встановленими на власний розсуд.

Оригінальна версія SDK CUDA була представлена 15 лютого 2007 року. Інтерфейс програмування програм CUDA заснований на мові C з деякими розширеннями. Для успішного перекладу коду на цю мову, CUDA SDK містить компілятор командного рядка Nvidia nvcc. Компілятор nvcc базується на компіляторі Open64 і призначений для перекладу файлів коду хоста (головного, керуючого коду) та файлів коду пристрою (апаратного коду) у файли об'єктів, придатні для кінцевої програми або бібліотеки, створеної в будь-якому середовищі програмування, z як NetBeans. Архітектура CUDA використовує модель пам'яті сітки, моделювання потоку кластерів та інструкції SIMD. Його можна використовувати не тільки для високопродуктивних графічних обчислень, але і для різних наукових розрахунків за допомогою відеокарт nVidia. Вчені та дослідники широко використовують CUDA у різних галузях, включаючи астрофізику, обчислювальну біологію та хімію, моделювання потоку, електромагнітні взаємодії, комп'ютерну томографію, сейсмічний аналіз тощо. CUDA може підключатися до програм, які використовують OpenGL та Direct3D. CUDA є багатоплатформою для операційних систем, таких як Linux, Mac OS X та Windows.22 У березні 2010 року nVidia випустила CUDA Toolkit 3.0 з підтримкою OpenCL.

Платформа CUDA була вперше запущена з виходом чіпа NVIDIA G80 восьмого покоління і стала доступною у всіх наступних серіях графічних чіпів для сімейств прискорювачів GeForce, Quadro та NVidia Tesla.

Перший пристрій, що підтримував CUDA-SDK, G8x, мав 32-розрядний одноточний векторний процесор, який використовує CUDA-SDK як API (CUDA підтримує двомовний тип C, але тепер точність 32 -Біта плаваюча точка зменшена)). Пізніше процесори GT200 підтримують 64-бітну точність (лише SFU), але продуктивність значно нижча, ніж 32-бітна точність (через те, що SFU - лише два для кожного потокового багатопроцесора, а скалярних процесорів -

вісім). GPU організовує багатопотокове обладнання, за допомогою якого можна використовувати всі ресурси GPU. Тож перспектива перевести функції фізичного прискорювача в графічний прискорювач (приклад реалізації - PhysX). Він також пропонує численні можливості використовувати пристрої комп'ютерної графіки для виконання складних, не графічних обчислень, наприклад, в області комп'ютерної біології та в інших областях науки.

У цій галузі архітектура CUDA пропонує такі переваги перед традиційним підходом до організації обчислень загального призначення за допомогою графічних API: ...

Інтерфейс програмування додатків CUDA (API CUDA) заснований на стандартній мові програмування С з деякими обмеженнями. На думку розробників, це повинно спростити і згладити процес вивчення архітектури CUDA

Спільна пам'ять об'ємом 16 Кб може використовуватися в організованому користувачем кеші з більш широкою пропускною здатністю, ніж при вилученні із звичайних текстур

- Більш ефективні транзакції між пам'яттю процесора та відеопам'яттю
- Повна апаратна підтримка цілих та бітових операцій

Підтримка GP для компіляції коду GPU через Open LLVM

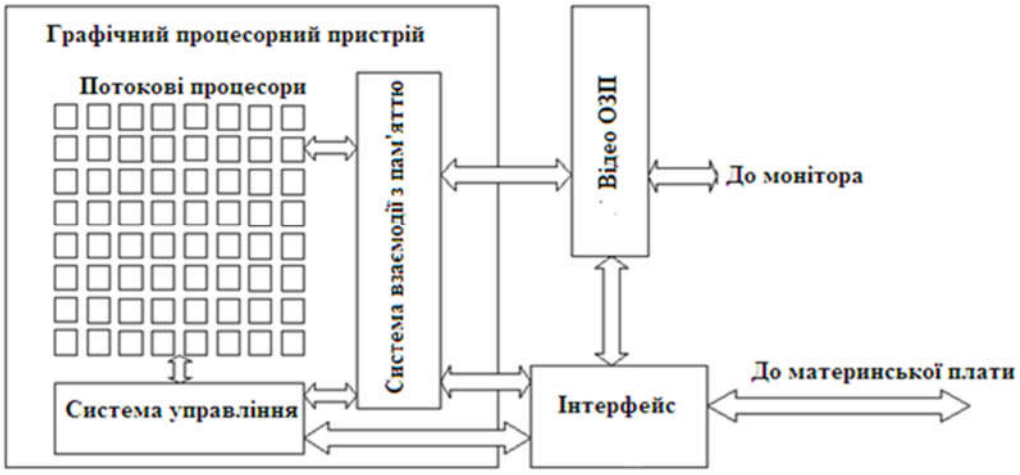

Рисунок 3.2 – Архітектура графічного процесора

Максимальний і мінімальний обчислювальний процесор NVIDIA GeForce GT 540M Це швидка відеокарта для ноутбуків середнього класу. Він заснований на тому ж мікросхемі, що і GeForce GT 435M, але з більшою тактовою частотою. Підтримуються два типи відеопам'яті: GDDR5 та DDR3 з 900 МГц. Однак через високу вартість та розумне енергоспоживання GDDR5 не використовується.

Як і попередник GT 435M, GT 540M базується на архітектурі GF108. GF108 заснований на архітектурі Fermi GF100 (DirectX 10). Однак він оптимізований для споживача (для ігор) та економить енергію.

Продуктивність NVIDIA GeForce GT 540M знаходиться між Radeon HD 5650 і Radeon HD 5730. Це означає, що цей графічний адаптер на 10% швидше, ніж GT435M (при використанні відеопам'яті DDR3). Під час використання GDDR5 продуктивність картки може бути однаковою, ніж у Radeon HD 5750 або HD 5770.

Разом з продуктивним процесором, ресурсоємні ігри 2010 року повинні вільно працювати в середніх (і високих) налаштуваннях і з роздільною здатністю SXGA.

Новинка мікросхем GF104 / 106/108 полягає в тому, що вони підтримують біткоїн HD аудіо (Blu-Ray) через HDMI. Як і Radeon HD 5730, GeForce GT 540M може передавати Dolby True HD і DTS-HD на приймач hi-fi без втрати якості.

GT 540M підтримує технологію PureVideo HD, яка використовується для декодування відео з графічним процесором. Вбудований відеопроцесор VP4 підтримує ряд функцій C. Тому GPU може декодувати MPEG-1, MPEG-2, MPEG-4, частина 2 (наприклад, MPEG-4 ASP, DivX або Xvid), VC-1, WMV9 і H. 264 (VLD, IDCT). Крім того, GPU може декодувати два потоки при 1080p одночасно (Blu-ray, picture-in-picture).

Підтримуючи CUDA, OpenCL та DirectCompute 2.1, GeForce GT 540M може допомогти у тривіальних обчисленнях. Наприклад, потоковий процесор може кодувати відео набагато швидше, ніж швидкий процесор. Крім того,

фізичні розрахунки можна проводити з графічним процесором за допомогою PhysX (підтримується Mafia 2 та Metro 2033). Однак GPU не настільки швидкий, що він обчислює PhysX і дозволяє грати в складні ігри з високими налаштуваннями.

Відповідно до тенденцій 3D, GeForce GT 540M підтримує технологію 3D бачення. Таким чином ви можете "завантажити" ноутбук 3D-вмістом (3D-ігри, 3D-фотографії), коли до ноутбука підключено 3D-дисплей або зовнішній 3D-екран.

Таблиця 3.1 – Технічна характеристика GEFORCE GT540m

Серія	GEFORCE GT540m
Архітектура	Fermi
Ядер CUDA	96
Тактова частота	672 МГц
Частота пам’яті	900 МГц
Тип пам’яті	DDR3
Розрядність шини пам’яті	128 біт
Максимум відеопам’яті	1536 Мб

Таблиця 3.2 – Результати експериментів обчислення суми квадратних різниць на GPU

Експеримент	Кількість елементів в матриці	Час виконання(мс)
1	1000	0,016
2	10000	0,031
3	100000	0,181
4	10000000	1,669

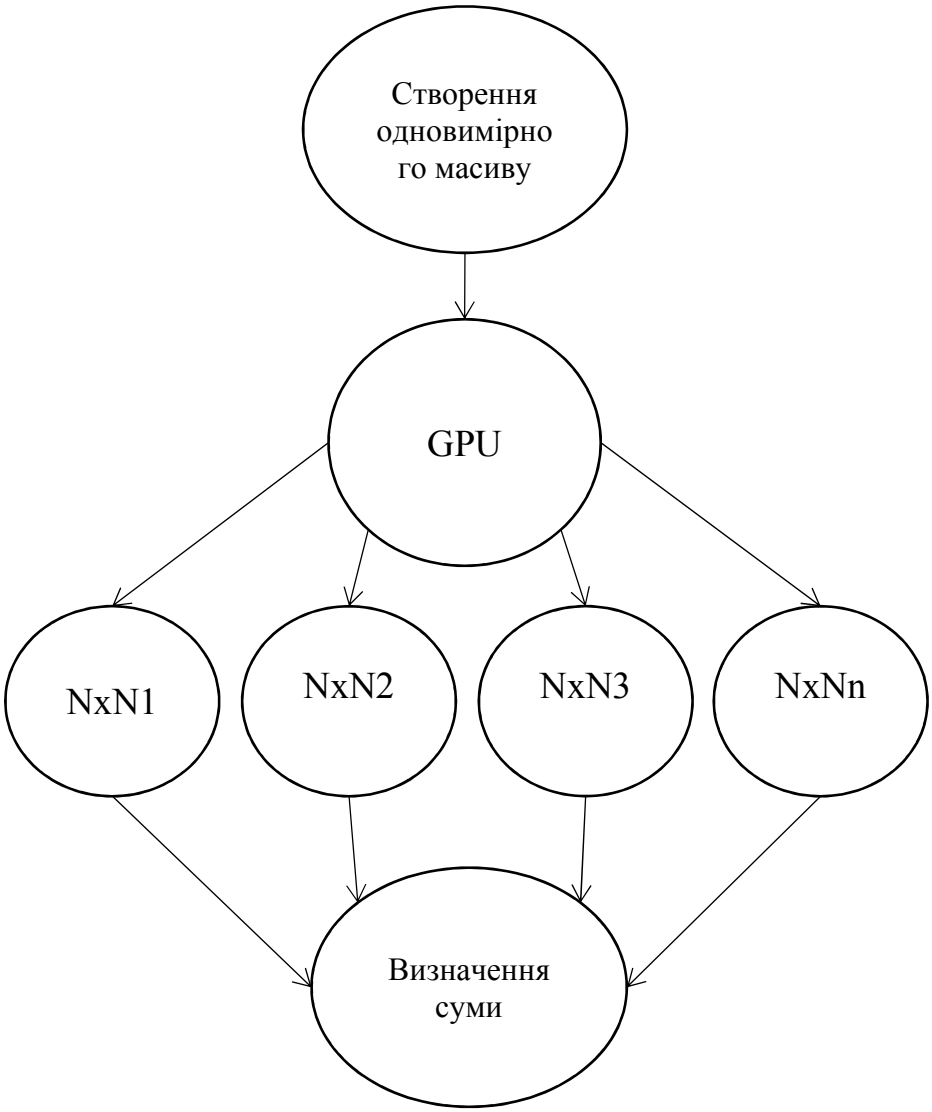

Рисунок 3.3 – Ярусно-параллельна форма обчислення суми квадратних різниць на GPU

3.3 Пристрій обчислення суми квадратних різниць процесором

Реалізація обчислення суми квадратних різниць на процесорі Intel Core I3-2330m. Intel Core i3-2330M - двоядерний процесор для ноутбуків. Він заснований на архітектурі Sandy Bridge і підтримує технологію гіперточення для одночасної обробки 4 потоків даних (для оптимального використання конвеєра). У порівнянні з більш швидким Core i5, i3 не підтримує Turbo Boost і тому постійно

працює на частоті 2,2 ГГц. Порівняно з Core i3-2310M, 2330M працює лише на частоті на 100 МГц вище, ніж основний дисплей.

Архітектуру Sandy Bridge було замінено Arrandale і підтримує нові 256-бітні інструкції AVX, вдосконалену технологію Turbo 2.0 та інтегровану 32-нм графічну карту.

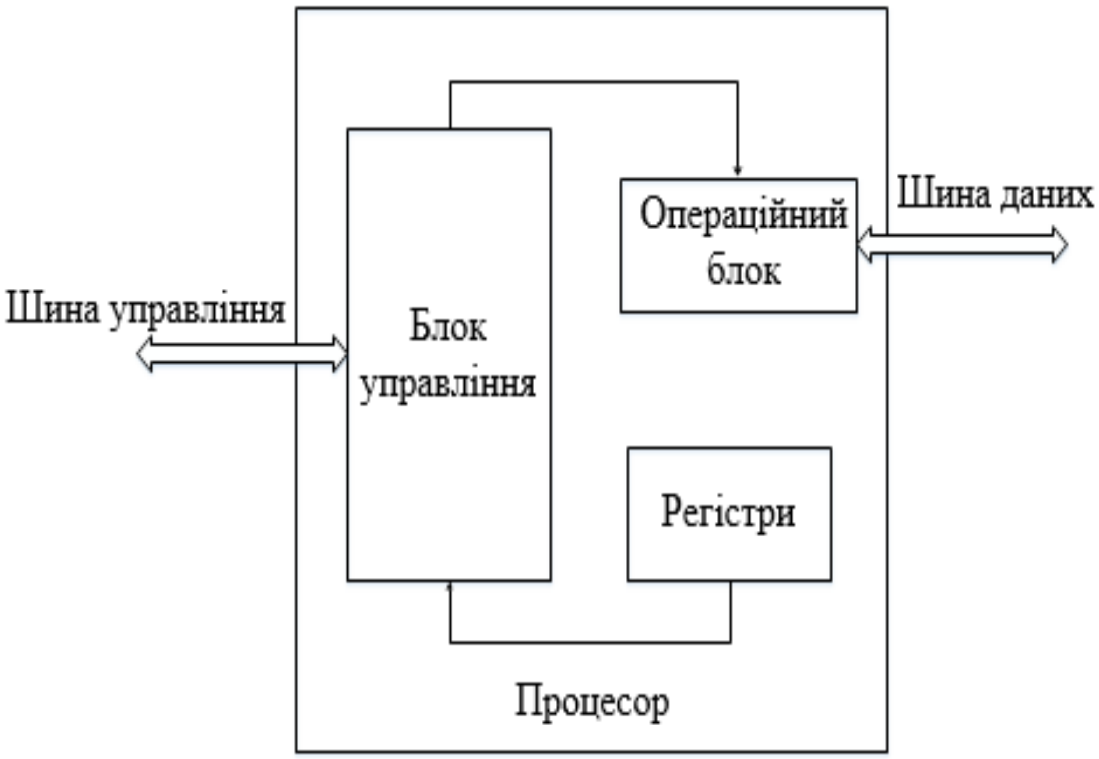

Рисунок 3.4 –Архітектура центрального процесора

Core i3-2330M оснащується вбудованою графічною картою Intel HD Graphics 3000, яка є помітно більш швидкої в порівнянні з Intel HD Graphics в процесорах сімейства Arrandale. Новий графічний чіп інтегрований в CPU і проводиться по 32 нм техпроцесу, може використовувати кеш-пам'ять третього рівня спільно з самим процесором (завдяки використанню нової кільцевої шини). У 2330M відеокарта працює на частоті від 650 до 1100 МГц (при використанні Turbo Boost). У більш швидких процесорах лінійки Sandy Bridge ця карта може підвищити частоту до 1300 МГц (наприклад, i5-2520M).

Крім того, покращений двоканальний контролер пам'яті для стандарту DDR3 розташовується на кристалі CPU, що наближає нас до реалізації повнофункціональної «системи в чіпі».

Завдяки покращеній архітектурі, середня продуктивність Core i3-2330M вище, ніж у Core i3 сімейства Arrandale з аналогічною частотою без Turbo Boost. У синтетичних бенчмарках, продуктивність даного процесора буде приблизно такою ж, як у Core i3-390M (з більш високою тактовою частотою - 2.5 ГГц). Отже, 2330M повинен без проблем впоратися з більшістю додатків.
Показник 35 Вт TDP включає енергоспоживання самого процесора, вбудованого GPU і контролера пам'яті.

Таблиця 3.3 – Технічна характеристика Intel Core i3-2330M

Серія	Intel Core i3-2330M
Архітектура	Sandy Bridge
Тактова частота	2200 МГц
Кеш 1-го рівня	128 Кб
Кеш 2-го рівня	512 Кб
Кеш 3-го рівня	3072 Кб
Кількість ядер	2
Кількість потоків	4
Максимальне енергоспоживання	35 Вт
Число транзисторів	624 Млн
Техпроцес	32 нм
Максимальна температура	85(PGA); 100(BGA) °C
Сокет	rPGA988B / BGA 1023
Додатково	HD Graphics 3000 (650- 1100MHz),
64 Bit	підтримка 64 Bit
Апаратна віртуалізація	VT-x

Опис бібліотеки Iostream - заголовки з класами, функціями та змінними для організації вводу / виводу на мові програмування C ++. Він включений у стандартну бібліотеку C ++. Назва походить від потоку введення / виводу. У C ++ та його попереднику, мові програмування C, немає інтегрованої підтримки вводу-виводу, натомість використовується бібліотека функцій. iostream керує введенням / виводом, як stdio.h у C. iostream використовує об'єкти cin, cout, cerr та clog для отримання інформації про та зі стандартного вводу, виводу, помилки (не буферизовано) чи помилки (буферизований)) Для передачі потоків. Як частина стандартної бібліотеки C ++, ці об'єкти також є частиною стандартного простору імен - std.

Обчислення суми квадратної різниці у процесорі Intel coreI3 -2330m:

```
#inc  lude <iostream>
using namespace std;
void randMatr (int *matr, int n, int k)
{
    for (int i = 0; i<n; ++i)
        for (int j = 0; j<n; ++j)
            matr[i * n + j] = rand()%k;
}
void outputMatr (int *matr, int n)
{
    for (int i=0; i<n; ++i, cout<<'\n')
        for (int j = 0; j<n; ++j)
            cout<<matr[i * n + j]<<'\t';
}
void min_elemnent (int *matr, int n)
{
    int min=matr[0];
    for (int i = 0; i<n; ++i)
        for (int j = 0; j<n; ++j)
        {
            if (matr[i*n+j]<min) min = matr[i*n+j];
        }
        cout<<"\n1 elenent = "<<min<<endl;
}
void sortMatr (int *matr, int n)
{
    for (int i = 0; i<n; ++i)
        for (int j = 0; j<n+1; ++j)
            if(matr[j*n+j]>matr[(j+1)*n+j+1])
            {
```

```
            int c=matr[j*n+j];
            matr[j*n+j]=matr[(j+1)*n+j+1];
            matr[(j+1)*n+j+1]=c;
        }
```

Таблиця 3.4 – Результати експериментів обчислення суми квадратних різниць на CPU

Експеримент	Кількість елементів в матриці	Час виконання(мс)
1	1000	2,4
2	10000	14,757
3	100000	26,123
4	10000000	36,123

3.4 Порівняння результатів виконання обчислення суми квадратних різниць на GPU і CPU

Для візуального представлення і порівняння результатів експериментів було прийнято рішення про побудову графіків. В графіках взято такі значення як кількість елементів в матриці і час виконання обчислення.

Таблиця 3.5 – Результати експериментів обчислення суми квадратних різниць на GPU

Експеримент	Кількість елементів в матриці	Час виконання(мс)
1	1000	0,016
2	10000	0,031
3	100000	0,181
4	10000000	1,669

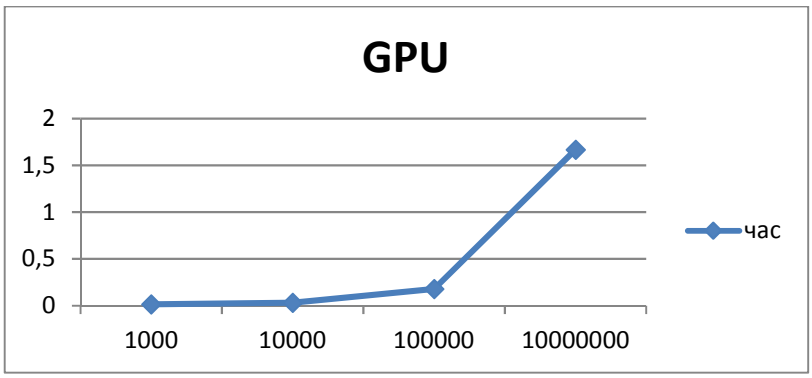

Рисунок 3.5 – Графік обчислення суми квадратних різниць на GPU

Таблиця 3.6 – Результати експериментів обчислення суми квадратних різниць на CPU

Експеримент	Кількість елементів в матриці	Час виконання(мс)
1	1000	2,4
2	10000	14,757
3	100000	26,123
4	10000000	36,123

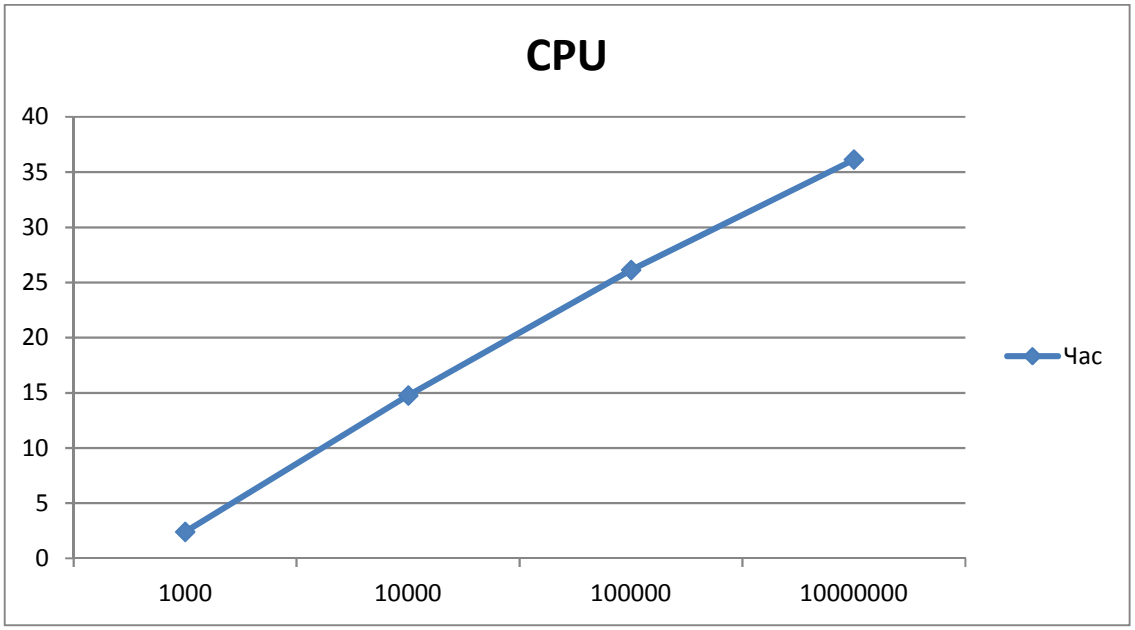

Рисунок 3.7 – Графік обчислення суми квадратних різниць на CPU

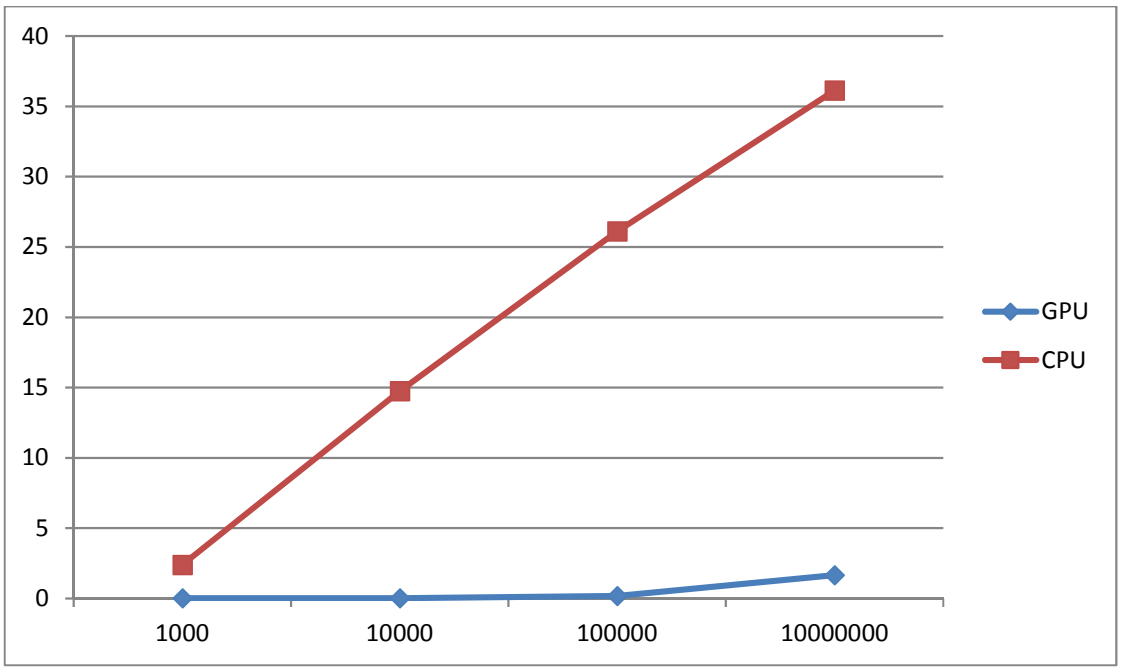

Рисунок 3.8 – Графік порівняння обчислення суми квадратних різниць на GPU і CPU

Порівнявши час виконання обчислення суми квадратних різниць на GPU і CPU, можна зробити висновки, що швидкість виконання обчислення на GPU значновища ніж на CPU/

ВИСНОВКИ

1. Обчислення суми квадратних різниць у наборі даних за допомогою GPU-GPU та програмної моделі CUDA можна забезпечити комплексним підходом, який охоплює наступне: вивчення методів та алгоритмів. Розрахунок обчислення суми квадратних різниць у полі даних; Архітектура графічного процесора та модель програмного забезпечення CUDA.

2. Обчислення суми квадратних різниць у наборі даних ґрунтується на методі порозрядного порівняння. Цей метод робить послідовне порівняння цифр усіх чисел від найбільшого числа.

3. Для розробки програмного забезпечення обчислення суми квадратних різниць за допомогою алгоритмів моделей програмного забезпечення GPU та CUDA для обчислення суми квадратних різниць повинен бути представлений у паралельному шарі.

4. Розробка високоефективних паралельних структур для обчислення суми квадратних різниць у полі даних шляхом розрядного порівняння найкраще проводиться за допомогою інтегрованого підходу, який охоплює методи, алгоритми, структури та технологію VLSI з урахуванням конкретного застосування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Березький О.М. Методичні рекомендації до виконання магістерської роботи з освітнього ступеня “Магістр”. Спеціальність: 123 - Комп’ютерна інженерія. Магістерська програма - Комп’ютерна інженерія" / О.М. Березький, Л.О. Дубчак, Г.М. Мельник /Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2018. 41 с.

2. Методичні вказівки до оформлення курсових проектів, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп’ютерна інженерія» / І.В. Гураль, Л.О. Дубчак / Під ред. О.М. Березького. Тернопіль: ТНЕУ, 2019. 33 с.

3. Бучковський Р.В. Використання нейромереж для обчислення потоків даних / II науково-практична конференція молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі» Частина 1. м. Тернопіль. 14 листопада 2019 р. с. 38

4. Бучковський Р.В. Алгоритм обчислення суми квадратних різниць з використанням технології CUDA / II науково-практична конференція молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі» Частина 1. м. Тернопіль. 14 листопада 2019 р. с. 39

5. Уоссермен Ф.Нейрокомпьютерная техника. – М.: Мир,1992. – 259с.

6. А.В. Палагин, В.Н. Опанасенко. Реконфигуриремые вычислительные системы. К.: Просвіта, 2006.- 280с.

7. Круглов В.В., Борисов В.В. Искусственные нейронные сети. Теория и практика. М.: Горячая Линия-Телеком, 2002. 382 с.

8. Николаев А.Б., Фоминых И.Б. Нейросетевые методы анализа и обработки данных. Учебное пособие. - М.: МАДИ (ГТУ), 2003, - 95с.

9. Цмоць І.Г. Інформаційні технології та спеціалізовані засоби обробки сигналів і зображень у реальному часі. Львів: УАД, 2005.- 227с.

10. Грибачев В. П. Элементная база аппаратных реализаций нейронных сетей // Компоненты и технологии. 2006. № 8

11. Круг П.Г. Нейронные сети и нейрокомпьютеры: Учебное пособие по курсу «Микропроцессоры». М.: Издательство МЭИ, 2002. 176 с.

12. Проблемы построения и обучения нейронных сетей / под ред. А.И.Галушкина и В.А.Шахнова. - М. Изд-во Машиностроение. Библиотечка журнала Информационные технологии №1. 1999. 105 с.

13. Галушкин А.И. Некоторые исторические аспекты развития элементной базы вычислительных систем с массовым параллелизмом (80- и 90- годы) // Нейрокомпьютер, №1. 2000. С.68-82

14. Аряшев С.И., Бобков С.Г., Сидоров Е.А. Параллельный перепрограммируемый вычислитель для систем обработки информационных сигналов // "Нейроинформатика 99". Москав, МИФИ. Часть 2. С.25-33.

15. Кирсанов Э.Ю. Цифровые нейрокомпьютеры: Архитектура и схемотехника. Казань: Казанский Гос. У-т. 1995. 131 с.

16. Власов А.И. Аппаратная реализация нейровычислительных управляющих систем //Приборы и системы управления 1999, №2, С.61-65.

17. Борисов В.Л., Капитанов В.Д. Методика быстрого создания нейроускорителей // Нейрокопьютеры: разработка и приенение, №1, 2000 год.- С.12-24.

18. Хехт-Нильсен Р. Нейрокомпьютинг: история, состояние, перспективы // Открытые системы. N4. 1998.

19. Власов А.И. Нейросетевая реализация микропроцессорных систем активной акусто- и виброзащиты// Нейрокомпьютеры:разработка и применение, №1, 2000. С.40-44.

20. Нейроприскорювачі на базі нейрочіпів - http://citforum.ru/hardware/neurocomp/neyrocomp_07.shtml

21. Елементна база нейрообчислювачів - http://opticstoday.com/katalog-statej/stati-na-ukrainskom/nejrokomputeri/elementna-baza-nejroobchislyuvachiv.html

22. Сучасні напрямки розвитку нейрокомп'ютерних технологій - http://www.victoria.lviv.ua/html/oio/html/theme9.htm

23. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы М.: Изд. Дом «Вильямс», 2001. 384с.

24. Браунси К. Основные концепции структур данных и реализация в С++. М.: Изд. Дом «Вильямс», 2002. 320с.

25. Проценко В.С. Техніка програмування мовою Сі: Навчальний посібник К.: Либідь, 1993. 224 с.

26. Шилдт Г. Теория и практика С++ СПб.: BHV Санкт-Питербург, 1996. 416 с.

27. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. М.: «Мир», 1979. 536с.

28. Вирт Н. Алгоритмы + структуры данных = программы. М.: "Мир", 1985. 44 с.

29. Гудман С. Хидетниеми С. Введение в разработку и анализ алгоритмов. М.: "Мир", 1981. 366 с.

30. Кнут Д. Искусство программирования для ЭВМ. т.3. Сортировка и поиск. М.:Мир, 1976. 678 с.

31. Мейер Б., Бодуэн К. Методы программирования: В 2-х томах М.: Мир, 1982. 356+368с.

32. Керниган, Б.,Мова програмування Сі. Завдання по мові Сі/Б.Керниган, Д.Ритчи. М.:ФиС, 1985. 280 с.

33. Страустрап Б. Мова програмування Сі ++ /Б.Страуструп. М.:Радіо та зв'язок, 1991.352 с.

34. Белецкий Я. Енциклопедія мови Сі /Я.Белецкий. М.:Мир, 1992. 687 с.

35. Белецкий Я. ТурбоСі++: Нова розробка: навч. посібник для студентів вищих навчальних закладів / Я.Белецкий. М.'.Машинобудівництво, 1994. 400с.

36. Пильщиков, В. Н. Збірник вправ по мові Паскаль: навч. Посібник для втузов /В. Н.Пильщиков. — М.:Висш. шк.,1990. —223 с.

37. Кармен, Томас Х., Лейзерон, Чарльз И., Ривест, Рональд Л., Штайн, Клиффорд. Алгоритми: построение и анализ, 2-е издание. :Пер. с англ. М.: Издательский дом “Вильямс”, 2005. 1296 с.

38. Кнут Д. Искусство программирования для ЗВМ: Сортировка и поиск. М., 1978. 844с.

39. Кухарев Г.А. и др. Техника параллельной обработки бинарних данньїх на СБИС. М.: Виш. Шк., 1991. 226 с

40. Пат. № 66138, Україна, МПК 006Б 7/38. Пристрій для обчислення сум парних добутків: Патент на корисну модель / І.Г. Цмоць, О.В. Скорохода; заявник і патентовласник Національний університет «Львівська політехніка». № 201106811; заявл. 30.05.2011; опубл. 26.12.2011, Бюл. № 24. 8 с..

41. Патент України на винахід №29700. Пристрій для визначення максимального числа з групи чисел. Бюл. №6-11. 2000. Рашкевич Ю.М., Зербіно Д.Д, Цмоць І.Г.

42. Кун С. Матричние процессорина СБИС. М.: Мир, 1991. 672

43. Галушкин А.И. Нейрокомпьютеры. Кн.3.-.М; ИПРЖР,2000.528с.

44. Круглов В.В., Борисов В.В. Искусственные нейронные сети. Теория и практика. 2-е изд., стереотип. М.: Горячая линия-Телеком, 2002. 382 с.

45. Круглов В.В., Борисов В.В. Искусственные нейронные сети. Теория и практика М.: Горячая Линия Телеком, 2002. 382 с.

46. Рутковская Д., Пилиньский Л., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы / Пер. с польского М.: Горячая линия.Телеком, 2007. 452 с.

47. Рассел С., Норвиг П. Искусственный интеллект: современный подход / Пер. с английского М.: Вильямс, 2007. 1408 с.

48. Рассел С., Норвиг П. Искусственный интеллект: современный подход / Пер. с английского. М.: Вильямс, 2007. 1408 с.

49. Цмоць І.Г. Принципи розробки і оцінка основних характеристик високопродуктивних процесорів на надвеликих інтегральних схемах/ Вісник ДУ “Львівська політехніка”, №349, Львів, 1998 с.5-11.

50. Батюк А.Є., Цмоць І.Г. Методи синтезу спеціалізованих обчислювальних систем для розв’язання задач у реальному часі / Інформаційні технології і системи. Т2, №1, Львів 1999 с.155-161.

Додаток А

Лістинг коду програми сортування на GPU

```
#inc  lude <iostream>
using namespace std;
void randMatr (int *matr, int n, int k)
{
    for (int i = 0; i<n; ++i)
        for (int j = 0; j<n; ++j)
            matr[i * n + j] = rand()%k;
}
void outputMatr (int *matr, int n)
{
    for (int i=0; i<n; ++i, cout<<'\n')
        for (int j = 0; j<n; ++j)
            cout<<matr[i * n + j]<<'\t';
}
void min_elemnent (int *matr, int n)
{
    int min=matr[0];
    for (int i = 0; i<n; ++i)
        for (int j = 0; j<n; ++j)
        {
            if (matr[i*n+j]<min) min = matr[i*n+j];
        }
        cout<<"\n1 elenent = "<<min<<endl;
}
void sortMatr (int *matr, int n)
{
    for (int i = 0; i<n; ++i)
        for (int j = 0; j<n+1; ++j)
            if(matr[j*n+j]>matr[(j+1)*n+j+1])
            {
                int c=matr[j*n+j];
                matr[j*n+j]=matr[(j+1)*n+j+1];
                matr[(j+1)*n+j+1]=c;
            }
```