НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Михайло Новотарський
(підпис)

“__” _____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему:Програмна модель обчислень в потокових системах з безпосередніми зв’язками між модуляли

Виконала : студентка __4__ курсу, групи _ІО-11
(шифр групи)

_____Тулуб Марія Миколаївна_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____проф, д.т.н., проф. Жабін В. І._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ас., док., філ Міщенко Л. Д._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
_ (посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____
(підпис)

Київ – 2025 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальності 123 “Коп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Михайло Новотарський
(підпис)
“__” _____2025 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Тулуб Марії Миколаївни

1. Тема проекту: Програмна модель обичлень в потокових системах з безпосередніми зв’язками між модуляли

керівник проєкту Жабін Валерій Іванович, проф, д.т.н.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від ___№1705_____

2. Термін здачі студентом закінченого проєкту ____02.06.2025_____

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Аналіз існуючих методів обчислень у потокових системах.

Розділ 2. Методологія дослідження та проєктування програмної моделі.

Розділ 3. Розробка програмної моделі обчислень

Розділ 4. Аналіз результатів

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (діаграма класів), алгоритм дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Міщенко Л. Д.		

7. Дата видачі завдання _____

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2024-15.12.2024*	
2.	*Вивчення та аналіз завдання*	*15.12.2024-15.03.2025*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2025-25.03.2025*	
4.	*Розробка структур окремих підсистем*	*25.03.2025-5.04.2025*	
5.	*Програмна реалізація системи*	*5.04.2025-15.04.2025*	
6.	*Оформлення пояснювальної записки*	*15.04.2025-20.05.2025*	
7.	*Захист програмного продукту*	*25.04.2025*	
8.	*Передзахист*	*23.05.2025*	
9.	*Захист*	*16.06.2025*	

Студентка-дипломник _____ Тулуб МАРІЯ
(підпис)

Керівник проєкту _____ Валерій ЖАБІН
(підпис)

АНОТАЦІЯ

У дипломній роботі досліджено та розроблено програмну модель для потокових обчислювальних систем, у яких обчислювальні модулі безпосередньо з’єднані між собою каналами передачі даних, без участі централізованого диспетчера або загальної пам’яті. Такий підхід дозволяє підвищити продуктивність та зменшити затримки при обробці даних, що є критично важливим для систем реального часу, цифрової обробки сигналів та високопродуктивних обчислень.

У роботі проаналізовано існуючі програмні моделі потокових обчислень (KPN, CSP, Dataflow), виділено їх обмеження щодо прямої взаємодії модулів. На основі цього сформульовано вимоги до нової моделі, побудованої за принципами асинхронного обміну даними, модульності та детермінізму.

Розроблена модель реалізована як граф обчислень, де вузли відповідають обчислювальним модулям, а ребра — FIFO-каналам зв’язку. Реалізовано прототип у середовищі Python. Експериментальне дослідження продемонструвало вищу пропускну здатність та нижчі затримки у порівнянні з традиційними підходами, такими як multiprocessing та процедурна реалізація.

Запропонована модель може бути використана як основа для створення оптимізованих потокових систем у галузях, що потребують високої швидкості обробки великих обсягів даних при мінімальних затримках.

ANNOTATION

This thesis presents the design and development of a computational model for stream processing systems where computational modules are directly connected via communication channels, without the use of a centralized scheduler or shared memory. This architecture enables higher throughput and lower latency in data processing, which is crucial for real-time systems, digital signal processing, and high-performance computing.

The work analyzes existing models of stream computation (KPN, CSP, Dataflow) and identifies their limitations in supporting direct inter-module communication. Based on this analysis, the thesis formulates requirements for a new model grounded in asynchronous data exchange, modularity, and determinism.

The developed model is implemented as a computation graph, where nodes represent processing modules and edges represent FIFO communication channels. A prototype was implemented in Python. Experimental evaluation demonstrated superior throughput and lower latency compared to traditional approaches such as Python multiprocessing and procedural implementations.

The proposed model can serve as a foundation for building optimized stream processing systems in domains that require high-speed processing of large data volumes with minimal delay.

справки	Форм	*Значення*	*Найменування*	Кіл.	№ екземпл	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Програмна модель обчислень	4		
			в потокових системах з			
			безпосередніми зв’язками			
			між модулями			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Програмна модель обчислень	65		
			в потокових системах з			
			безпосередніми зв’язками			
			між модулями			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Програмна модель обчислень	1		
			в потокових системах з			
			безпосередніми зв’язками			
			між модулями			
			Блок-схема алгоритму			
	А4	***ІАЛЦ.4672008.005 Д2***	Програмна модель обчислень	1		
			в потокових системах з			
			безпосередніми зв’язками			
			між модулями			
			Функціональна схема			
	А4	***ІАЛЦ.467200.006 Д3***	Програмна модель обчислень	1		
			в потокових системах з			
			безпосередніми зв’язками			
			між модулями			
			Структурна схема			
	А4	***ІАЛЦ.467200.007 Д4***	Програмна модель обчислень	1		
			в потокових системах з			
			безпосередніми зв’язками			
			між модулями			
			Текст програмного коду			

					ІАЛЦ.467200.001 ОА
Зм	***Лист***	***№ докум.***	***Підп***	***Дата***	
Розроб		Тулуб М. М			*Програмна модель обчислень в потокових симтемах з безпосередніми зв’язками між модулями* Опис альбому
Перев		Жабін В. І.			

Літ.	Аркуш	Аркушів
	1	1

КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Програмна модель обислень в потокових системах з безпосередніми зв’язками між модулями»

Київ – 2025

ЗМІСТ

					ІАЛЦ.467200.002 ТЗ		
		№ докум.	Підпис	Дата			
Розробив		Тулуб М. М.			**Програмна модель обчислень в потокових системах з безпосередніми зв'язками між модулями** Пояснювальна записка	Літ.	Аркуш 1 / Аркушів 4
Перевірив		Волокита А. М.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11	
Н. Контр.		Міщенко Л. Д.					
Затвердив							

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Програмна модель обчислень в потокових системах з безпосередніми зв’язками між модулями — це розробка, яка спрямована на створення ефективної програмної моделі для опису та реалізації обчислювальних процесів у потокових архітектурах, де модулі безпосередньо з’єднані між собою каналами передачі даних, без участі централізованого диспетчера або спільної пам’яті.

Область застосування цієї моделі охоплює широке коло технічних і наукових задач, пов’язаних з обробкою великих обсягів даних у реальному часі. Зокрема, модель може бути використана в системах цифрової обробки сигналів (DSP), вбудованих системах, пристроях Інтернету речей (IoT), системах автоматизованого керування, графічних процесорах, потокових обчислювальних середовищах для наукових симуляцій, а також у задачах аналізу аудіо- та відеопотоків.

Завдяки модульності, асинхронній роботі та детермінованості, запропонована модель є придатною для побудови гнучких, масштабованих та високопродуктивних програмних або апаратно-програмних рішень, орієнтованих на обчислення з мінімальними затримками та максимальним використанням обчислювальних ресурсів.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Зростання обсягів потокових даних і вимог до швидкодії обчислювальних систем потребує створення ефективних моделей взаємодії обчислювальних модулів безпосередньо між собою, без централізованого керування чи спільної пам’яті. Такий підхід дозволяє підвищити продуктивність, знизити затримки і забезпечити детермінованість обчислень. Актуальність розробки підтверджується широким застосуванням потокових обчислень у цифровій обробці сигналів, графічних процесорах, розподілених системах та обробці великих даних, що робить розробку програмної моделі з

прямими зв’язками між модулями необхідною для оптимізації сучасних потокових систем.

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою роботи є створення програмної моделі обчислень для потокових систем із безпосередніми зв’язками між модулями, що забезпечує високу продуктивність, мінімальні затримки та детерміновану взаємодію. Розробка спрямована на підвищення ефективності обробки потокових даних у реальному часі та створення основи для побудови масштабованих і гнучких обчислювальних систем. Призначення моделі — забезпечити надійну, модульну і легко розширювану архітектуру, яка може бути застосована у цифровій обробці сигналів, вбудованих системах, наукових обчисленнях та інших галузях, що потребують оптимальної організації потокових обчислень.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Забезпечувати організацію безпосередніх зв’язків між обчислювальними модулями через FIFO-канали.
- Гарантувати детермінованість обчислень при повторних запусках з однаковими вхідними даними.
- Мати модульну архітектуру, що дозволяє додавати, замінювати або видаляти обчислювальні модулі без порушення роботи системи.
- Забезпечувати високу продуктивність та мінімальні затримки при передачі даних

- Підтримувати паралельне виконання обчислювальних модулів для підвищення ефективності.
- Мати зрозумілу структуру, що полегшує тестування та налагодження.

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.
- Visual Studio 2017 IDE версії або вище.

5.3. Вимоги до апаратної частини

- Для побудови апаратних засобів передбачається використати сучасну елементну базу ПЛІС (FPGA).

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.12.2024-15.12.2024
Вивчення та аналіз завдання	15.12.2024-15.03.2025
Розробка архітектури та загальної структури системи	15.03.2025-25.03.2025
Розробка структур окремих частин системи	25.03.2025-5.04.2025
Програмна реалізація системи	5.04.2025-15.04.2025
Виправлення помилок	15.04.2025-20.05.2025
Оформлення пояснювальної записки	25.04.2025

ПОЯСНЮВАЛЬНЯ ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Програмна модель обислень в потокових системах з безпосередніми зв’язками між модулями»

Київ – 2025

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

НССЧ — Надлишкова симетрична система числення

ПЗ — Програмне забезпечення

ПЛІС — Програмована логічна інтегральна схема

АРМ — Автоматизоване робоче місце

ФПГА (FPGA) — Field Programmable Gate Array (Польова програмована вентильна матриця)

IEEE — Institute of Electrical and Electronics Engineers

RTL — Register Transfer Level (Рівень регістрових передач)

ЦП — Центральний процесор

ОЗП — Оперативний запам’ятовуючий пристрій

РГ — Регістр

ГЗ — Глобальний зв'язок

АК — Арифметичний компонент

СА — Суматор арифметичний

ДОС — Діалогова операційна система (в контексті може використовуватись як система керування)

АЛП — Арифметико-логічний пристрій

ЗП — Запам’ятовуючий пристрій

Вступ

У сучасну епоху інформаційних технологій та цифрової трансформації ключовим фактором успіху багатьох систем є здатність швидко та ефективно обробляти великі обсяги даних у режимі реального часу. Особливо це стосується таких сфер, як цифрова обробка сигналів, робототехніка, Інтернет речей (IoT), штучний інтелект, системи відеоаналітики та наукові обчислення, де відсутність затримок та висока продуктивність є критично важливими. Одним із перспективних напрямів у розробці обчислювальних систем є використання потокових моделей обчислень, що базуються на паралельній обробці та обміні даними у вигляді потоків між модулями.

Потокові обчислювальні системи дозволяють розділити складну задачу на менші, незалежні або взаємозалежні модулі, які обмінюються даними безпосередньо, формуючи граф обчислень. Такий підхід сприяє підвищенню масштабованості, гнучкості системи та покращенню продуктивності. Водночас традиційні моделі, що базуються на централізованому керуванні або використанні спільної пам’яті, мають суттєві обмеження, зокрема виникають затримки через конкуренцію за ресурси та складність у забезпеченні детермінованості результатів.

Актуальність дослідження полягає в необхідності розробки ефективної програмної моделі, яка дозволила б організувати безпосередній, асинхронний обмін даними між обчислювальними модулями без посередників, що забезпечує мінімальні затримки, високу пропускну здатність та детерміновану поведінку системи. Така модель може значно розширити можливості потокових систем у різних сферах застосування, забезпечуючи високу надійність та гнучкість при збереженні простоти розробки.

Метою цієї дипломної роботи є розробка та дослідження програмної моделі обчислень у потокових системах з безпосередніми зв’язками між модулями. Для досягнення мети поставлені наступні завдання: здійснити

огляд існуючих моделей потокових обчислень та їх обмежень; визначити основні вимоги до нової моделі; розробити алгоритмічну та програмну реалізацію моделі; провести експериментальне дослідження продуктивності та порівняти її з традиційними підходами.

Використання моделей потокових обчислень дозволяє прискорити обчислення в реальному часі за рахунок паралелізму виконання операцій на на рівні обробки машинних слів (паралельні ланцюги графів алгоритмів в різних обчислювальних модулях)та на рівні обробці розрядів операндів (послідовні ланцюги залежних операцій).

Останній підхід забезпечує суміщення в часі виконання послідовності залежних за даними операцій в режимі суміщення. Для цього застосовують спеціальну надлишкову систему числення з порозрядній передачи інформации. Крім того, за рахунок цього скорочується необхідна кількість виводів (пінів) мікросхем, що зменшує апаратурні витрати на реалізацію систем.

Практична цінність роботи полягає у створенні гнучкої, масштабованої моделі, що може бути використана як основа для розробки високопродуктивних потокових систем у реальному часі, а також для навчання і подальших досліджень у сфері розподілених та паралельних обчислень.

Структурно робота складається з чотирьох розділів: огляду літератури та аналізу предметної області, опису методів дослідження і проєктування, розробки програмної моделі та аналізу отриманих результатів.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ОБЧИСЛЕНЬ У ПОТОКОВИХ СИСТЕМАХ

1.1 Вступ до проблематики потокових обчислень

З кожним роком сучасні інформаційні технології охоплюють дедалі більше галузей людської діяльності, від побутових пристроїв до складних систем керування промисловими об'єктами, транспортною інфраструктурою, енергетикою, медициною. У більшості таких систем обробка даних має відбуватися не після їх повного надходження, а в режимі реального часу або близького до нього, тобто — обробляючи дані в міру їх надходження. Це породжує специфічні вимоги до архітектури обчислень, де традиційна послідовна модель (фон Неймана) уже не відповідає очікуванням щодо продуктивності, паралелізму, масштабованості та передбачуваності.

У таких умовах усе більше уваги приділяється потоковим моделям обчислень (англ. *dataflow computing models*), які передбачають, що логіка виконання програми не задається жорстко визначеною послідовністю інструкцій, а зумовлюється наявністю необхідних вхідних даних. Іншими словами, обчислювальний модуль або функціональний блок (актор) починає працювати лише тоді, коли всі необхідні дані доступні на його вході. Це дозволяє природно реалізувати паралелізм обробки, оскільки незалежні модулі можуть виконуватись одночасно — без потреби у централізованому керуванні порядком їх виклику.

З погляду структурного представлення, програма у потоковій моделі зображується як орієнтований граф обчислень, у якому вершини відповідають обчислювальним модулям, а дуги — каналам передачі даних між ними. Завдяки такому підходу забезпечується не тільки чіткий розподіл відповідальності між модулями, але й спрощується масштабування —

додавання нових функціональних блоків або розподіл виконання між вузлами кластера чи ядрами процесора.

Перші згадки про потокові моделі з’явилися ще в 1960–1970-х роках у контексті розробки альтернатив до традиційних процесорних архітектур. Зокрема, Джек Денніс запропонував концепцію dataflow machines, які замість програми у вигляді послідовності інструкцій виконували її у вигляді графа потоків даних. Подальший розвиток цієї ідеї привів до формулювання Kahn Process Networks (KPN), де було вперше формалізовано поняття детермінованої взаємодії незалежних процесів через канали.

У 1980–1990-х роках розвиток технологій цифрової обробки сигналів (DSP) дав новий поштовх до формалізації Synchronous Dataflow (SDF), яка передбачала, що кожен модуль споживає та генерує фіксовану кількість токенів (одиниць даних) за одну операцію. Це дозволило статично планувати виконання, розраховувати необхідні обсяги буферів, уникати небажаних ситуацій, таких як дедлоки або втрати даних. Саме ця модель і стала основою для інженерних платформ, таких як Simulink, LabVIEW, Ptolemy II, а також для мов програмування у вбудованих системах.

З часом, у зв’язку з вибуховим зростанням обсягів даних, що передаються через мережі, та з переходом на розподілені системи, з'явилися практичні реалізації потокових обчислень у вигляді фреймворків Apache Storm, Apache Flink [9], Kafka Streams, Google Dataflow**.** Вони дозволяють обробляти потоки даних у реальному часі у масштабах великих кластерів, забезпечуючи розподілення навантаження, стійкість до збоїв і масштабування. Проте їх основна архітектурна особливість — це опосередкована передача даних через брокери, черги або шини, що добре працює у хмарних середовищах, але не підходить для систем із жорсткими обмеженнями на час відгуку або ресурсоспоживання.

У системах класу реального часу, таких як керування автономним транспортом, високоточні вимірювання, реактивні системи безпеки — будь-яке навіть мінімальне збільшення затримки, спричинене посередницькою

інфраструктурою, є критичним. У таких випадках необхідна максимально легка архітектура, де модулі передають дані напряму — без посередників, без маршрутизаторів, без черг, що фізично розміщені окремо від модулів обробки. Такий підхід називається передачею з безпосередніми зв'язками між модулями, і саме він забезпечує найнижчі можливі затримки, передбачуваність поведінки та спрощення трасування даних у системі.

Однак моделі з прямими зв'язками поки що мало представлені у загальнодоступних фреймворках і потребують окремого дослідження, проєктування та формалізації. Вони вимагають уважного підходу до керування буферами, планування взаємодій, уникнення дедлоків та синхронізації роботи окремих модулів. Проте переваги, які вони дають — особливо в задачах із критичними вимогами до часу — переважають складність їх реалізації.

Отже, в умовах розвитку edge-computing, вбудованих систем і систем реального часу потокові моделі з прямими зв'язками стають не лише теоретично привабливими, а й практично необхідними. Їх впровадження дозволяє суттєво підвищити ефективність і передбачуваність роботи систем, знизити затрати на комунікацію, спростити реалізацію з урахуванням жорстких обмежень. У цьому контексті детальне вивчення потокових моделей такого типу є актуальним і перспективним напрямом досліджень, що лежить в основі теми цієї дипломної роботи.

1.2 Класифікація потокових моделей обчислень

Потокові обчислення, як клас моделей програмування, мають широке розмаїття форм, які виникли з різних підходів до вирішення задач обробки даних у реальному часі, паралельної обробки, цифрової обробки сигналів та роботи в розподілених середовищах. Попри спільну концептуальну основу — передавання даних між незалежними модулями — конкретні реалізації

потокових моделей суттєво відрізняються за механізмами взаємодії, типами каналів, правилами активації модулів і рівнем детермінованості виконання. Для ефективного застосування цих моделей у практичних умовах важливо розуміти їх класифікацію, сильні сторони, обмеження та сферу доцільного використання.

Однією з перших і базових форм потокових обчислень є Kahn Process Networks (KPN). У цій моделі кожен вузол виконується як окремий процес, а обмін даними відбувається через одновимірні канали FIFO. KPN гарантує детермінізм результатів: за одних і тих самих умов виконання — результат завжди однаковий, незалежно від черговості запуску процесів. Проте у чистому вигляді KPN важко реалізувати у вбудованих системах через потенційно нескінченні черги та складність прогнозування ресурсів. Через це вона частіше використовується як теоретична основа для побудови інших моделей.

Більш практичною та широко застосовуваною є Synchronous Dataflow (SDF) — модель, у якій кожен модуль (актор) споживає і генерує фіксовану кількість токенів на кожному циклі виконання. Завдяки цій властивості можливе статичне планування виконання, визначення порядку запуску модулів, обчислення мінімального необхідного обсягу буферів та повна передбачуваність роботи системи. SDF активно використовується у цифровій обробці сигналів, у моделях керування, а також у вбудованих системах, де важлива точність розрахунків і відповідність вимогам реального часу. Її застосування особливо ефективне в поєднанні з автоматичною генерацією коду для вбудованих платформ (наприклад, через Simulink або Xilinx Vivado HLS).

Ще один тип — Cyclo-Static Dataflow (CSDF) — дозволяє задавати змінну кількість споживаних/генерованих токенів у циклічному порядку. Це надає моделі гнучкості в порівнянні з SDF, зберігаючи при цьому можливість аналізу і планування. Її застосування доречне, коли потрібно описати

модуляцію або змінні патерни надходження даних, наприклад, у комунікаційних системах.

Dynamic Dataflow (DDF) — модель, у якій обсяги даних і навіть структура графа можуть змінюватися динамічно. Така гнучкість дозволяє обробляти складні або непередбачувані потоки подій, але знижує можливість наперед оцінити параметри виконання. DDF використовується у високорівневих платформах потокової обробки, зокрема у фреймворках для Big Data, де динаміка є ключовою властивістю.

Окрему групу складають акторні моделі обчислень (Actor Model), які отримали широке поширення у розподілених системах. У цій моделі актор — це незалежний об’єкт, який обробляє вхідні повідомлення, може змінювати свій стан і надсилати повідомлення іншим акторам. Цей підхід реалізовано в таких платформах, як Erlang/OTP, Akka (Scala/Java), CAF (C++). Акторна модель дуже добре масштабується, легко адаптується до fault-tolerant архітектур, але потребує спеціального керування буферами та синхронізацією, особливо якщо її застосовувати у системах реального часу.

Щодо промислових систем обробки даних, особливо в хмарних і мікросервісних середовищах, актуальними є Data Stream Processing Frameworks, такі як Apache Flink, Apache Storm, Kafka Streams, Google Dataflow. Ці фреймворки реалізують подієво-орієнтовану архітектуру, де обробка відбувається на основі потоків повідомлень, що надходять до розподіленої системи через брокери (Kafka, Pulsar). Вони дозволяють масштабувати систему до сотень вузлів, обробляти мільйони подій за секунду, гарантувати delivery semantics (exactly-once, at-least-once), але мають значну інфраструктурну складність і затримки, що робить їх непридатними для вбудованих систем або застосувань з жорстким часовим бюджетом.

З огляду на тип взаємодії між модулями, потокові моделі можна класифікувати на два великих класи:

- Моделі з опосередкованими зв’язками, де дані передаються через проміжні компоненти — брокери, черги, сервісні шини. Вони зручні

- для модульних, fault-tolerant систем, проте призводять до зростання затримок, накладних витрат і складності маршрутизації.
- Моделі з безпосередніми зв’язками між модулями, в яких модулі передають дані напряму через спільну пам’ять або локальні буфери. Вони забезпечують мінімальну латентність, передбачуваність поведінки та спрощення архітектури, що є критичним для систем реального часу.

Вибір тієї чи іншої моделі залежить від конкретного застосування, обмежень апаратного середовища, вимог до часу реакції, необхідності масштабування, наявності сторонніх бібліотек і зручності відлагодження. Для задач, де пріоритетом є висока швидкодія, мінімальні затримки, керованість потоками даних і відмова від зовнішніх посередників, найкраще підходять моделі, засновані на SDF або статичній акторній архітектурі з прямими каналами зв’язку між модулями.

Таким чином, різноманіття потокових моделей створює широкий інструментарій для розробників систем, що потребують обробки даних у реальному часі. Правильний вибір моделі дозволяє досягти балансу між складністю, ефективністю та передбачуваністю виконання. У цій дипломній роботі обґрунтовано вибір моделі з безпосередніми зв’язками, яка поєднує переваги строгого контролю над структурою системи та високої продуктивності, що розглядається як основа для подальшого проєктування.

1.3 Архітектурні особливості потокових систем з безпосередніми зв’язками між модулями

Архітектура потокових систем безпосередньо впливає на їх здатність задовольняти вимоги до обробки даних у режимі реального часу, ефективного

використання ресурсів, масштабованості та передбачуваності виконання. Основною відмінною рисою таких систем є організація взаємодії між обчислювальними модулями: від того, як саме дані передаються від одного модуля до іншого, залежить не лише продуктивність, а й стабільність та надійність усього програмного комплексу.

У більшості сучасних реалізацій потокових архітектур зв’язки між модулями будуються через опосередковані канали комунікації. Це можуть бути програмні черги повідомлень, брокери подій, централізовані шини даних або middleware-рівні, які забезпечують ізоляцію компонентів, маршрутизацію, повторну передачу при збоях, логування тощо. Такий підхід добре зарекомендував себе в хмарних середовищах та масштабованих обчислювальних платформах, де компоненти можуть працювати на різних фізичних вузлах, не знаючи про деталі один одного.

Однак така архітектура має низку суттєвих обмежень, особливо в контексті реального часу, систем з обмеженими ресурсами або вбудованих платформ. Серед основних недоліків: затримки, викликані проміжною буферизацією, серіалізацією і десеріалізацією даних, накладними витратами на обслуговування черг або брокерів, необхідністю періодичного опитування черг (polling), а також складністю трасування потоку даних і налагодження. У критичних системах навіть незначна латентність або втрата повідомлень через перевантаження буфера може призвести до порушення вимог безпеки або некоректного функціонування.

У відповідь на ці виклики виникає альтернатива — архітектура з безпосередніми зв’язками між модулями (англ. *direct module connections*). У такій архітектурі кожен обчислювальний модуль напряму пов’язаний із модулями, яким він передає результати своєї роботи. Це реалізується, як правило, через локальні буфери, спільну пам’ять, черги в оперативній пам’яті або навіть структури типу producer-consumer, реалізовані на рівні програмного коду. Важливо, що в таких системах немає глобального посередника, який би регулював передачу повідомлень. Кожен модуль

відповідає лише за свою обчислень та безпосередню взаємодію з підключеними до нього модулями.

Основною перевагою такого підходу є мінімізація затримок — дані передаються без додаткової обробки або маршрутизації, що критично важливо в задачах цифрової обробки сигналів, керування реальними об’єктами (наприклад, дронами, медичними приладами, роботизованими системами), де прийнятна латентність обчислень вимірюється мілісекундами або мікросекундами. Крім того, відсутність зовнішньої черги або брокера спрощує структуру програми, підвищує детермінованість і дозволяє точніше контролювати стан системи у кожен момент часу.

У такій архітектурі зв’язок між модулями проєктується на етапі конфігурації системи: граф обчислень фіксується заздалегідь, канали між модулями створюються як об’єкти у пам’яті, а доступ до них здійснюється напряму через вказівники або дескриптори. Це дозволяє використовувати статичний аналіз графа для виявлення потенційних проблем, таких як цикли без виходу (дедлоки), зайві залежності або неефективні конфігурації.

Для досягнення узгодженого виконання обчислень у системах із безпосередніми зв’язками використовують різні механізми активації модулів. У простіших випадках — це синхронна модель, коли всі модулі працюють за глобальним розкладом. У більш динамічних системах — асинхронна модель, у якій модуль активується, коли відповідні дані стають доступними. У складних реалізаціях використовується гібридна схема, яка поєднує елементи обох підходів: наприклад, ініціалізація за глобальним сигналом, а далі — подієво-орієнтоване виконання.

Ще однією архітектурною особливістю таких систем є буферизація каналів. У більшості випадків кожен канал між модулями має свій локальний буфер, розмір якого визначає здатність системи витримувати пікові навантаження без втрати даних. Оптимізація розміру буферів є ключовим етапом у проєктуванні: занадто малі буфери можуть призводити до зупинки модулів через відсутність місця для запису; занадто великі — до перевитрати

пам’яті, що критично у вбудованих системах. У моделях типу SDF буфери можна розрахувати аналітично на основі топології графа.

У цілому архітектура з безпосередніми зв’язками між модулями поєднує в собі простоту реалізації, високу продуктивність, знижене енергоспоживання (через відсутність проміжних етапів обробки) та чіткість структури, що робить її особливо привабливою для розробки систем, орієнтованих на виконання в режимі реального часу. Попри те, що така модель вимагає більш уважного проєктування, вона є фундаментом для створення ефективних, надійних і масштабованих обчислювальних платформ у різних сферах — від вбудованих пристроїв до цифрових контролерів і edge-систем.

1.4 Стан сучасних досліджень та технологій у галузі потокових систем

Протягом останніх десятиліть потокові обчислення стали не лише об'єктом теоретичних досліджень, але й однією з ключових парадигм у практичному програмуванні розподілених, вбудованих та реального часу систем. Завдяки здатності ефективно розподіляти навантаження між модулями та забезпечувати паралельну обробку даних, потокові моделі стали основою сучасних фреймворків і середовищ розробки. Водночас еволюція таких систем супроводжується низкою складних викликів, пов’язаних із продуктивністю, передбачуваністю, затримками та інтеграцією з апаратними ресурсами.

На науковому рівні значну увагу приділено формалізації потокових моделей. Одні з найвпливовіших робіт належать таким дослідникам, як Едвард Лі, Пол Хадсон, Жан-Луї Гійу, які систематизували властивості моделей *Synchronous*

Dataflow, *Cyclo-Static Dataflow* та *Boolean Dataflow*, а також створили підґрунтя для використання потокових графів у програмному моделюванні. Зокрема, у проєкті Ptolemy II (Berkeley University) реалізовано платформу для моделювання гетерогенних систем, де можливо комбінувати SDF, FSM (кінцеві автомати), DE (дискретні події) та інші парадигми для створення складних інтегрованих систем.

Поряд із теоретичними дослідженнями активно розвиваються прикладні інструменти. Наприклад, середовище Simulink (MathWorks) дозволяє розробляти потокові моделі з графічним інтерфейсом, автоматично генерувати C-код для вбудованих платформ і проводити симуляції з урахуванням часових затримок. У сфері FPGA-розробок відзначаються рішення Xilinx Vivado HLS та Intel DSP Builder, які дають змогу реалізовувати потокові алгоритми на апаратному рівні з мінімальними затримками.

На рівні розподілених систем спостерігається стрімкий розвиток платформ потокової обробки даних у хмарному середовищі. Найбільш відомими є Apache Storm, Apache Flink, Kafka Streams**,** Spark Streaming, Google Dataflow. Ці платформи реалізують обробку потоків подій у режимі near-real-time або real-time, забезпечують масштабованість, fault tolerance, backpressure-механізми та stateful processing. Наприклад, Apache Flink [9] використовує складну модель розподіленого графа операторів із керуванням станом і обробкою подій у часових вікнах. Такі фреймворки добре підходять для обробки великих обсягів інформації у фінансовому секторі, телекомунікаціях, e-commerce та IoT-платформах.

Проте всі ці фреймворки, попри їхню технологічну зрілість, ґрунтуються на моделі опосередкованої взаємодії між модулями. Дані передаються через брокери (Kafka, Pulsar), черги або внутрішні шини, що призводить до зростання затримок, складності інфраструктури та втрати детермінізму. Це є критичним недоліком у системах, що функціонують із жорсткими часовими

обмеженнями — таких як робототехніка, керування динамічними об’єктами, автономні системи, цифрова обробка сигналів у реальному часі.

У відповідь на це, наукові дослідження останніх років усе більше фокусуються на оптимізації прямої взаємодії між модулями. Наприклад, у дослідженнях, що стосуються вбудованих систем керування (авіоніка, медична техніка), пропонуються спеціалізовані мікроядра та lightweight runtime-платформи, де кожен модуль отримує прямий доступ до буфера сусіднього модуля. Такі системи використовують статично проєктовані графи виконання з чіткими гарантіями часу реакції (hard real-time). Прикладом є використання потокової моделі з прямими каналами в реальному часі у проектах на основі AUTOSAR, OSEK, RTEMS.

Крім того, у напрямі edge computing та IoT-пристроїв зростає зацікавленість у потокових архітектурах з прямим зв’язком, які можуть працювати без мережевої інфраструктури або хмарної підтримки. Завдяки низьким енергозатратам, мінімальній латентності та простоті впровадження, такі архітектури добре підходять для задач моніторингу, локальної обробки сигналів, smart sensor applications.

Також слід згадати розвиток відкритих специфікацій, орієнтованих на підтримку потокових парадигм. Зокрема, Reactive Streams, Project Reactor, Akka Streams [7] — це технології, що дозволяють описувати обробку даних у вигляді реактивних ланцюжків з контролем потоку (backpressure), підтримкою асинхронної обробки та спрощенням масштабування. Вони активно використовуються в розробці мікросервісних застосунків, зокрема в екосистемі Spring (Java), але, знову ж таки, опираються на концепцію посередника.

Таким чином, стан сучасних досліджень і розробок підтверджує високу актуальність потокових моделей обчислень у всіх сферах IT. Проте в умовах високих вимог до часу реакції, низької латентності та обмежених ресурсів усе чіткіше проявляється недостатність опосередкованих рішень і необхідність переходу до архітектур з безпосередніми зв’язками між

модулями. Саме ці підходи дають змогу створювати більш ефективні, передбачувані та компактні обчислювальні системи. У наступному підрозділі буде сформульовано мотивацію вибору теми цієї роботи саме в цьому контексті.

1.5 Обґрунтування вибору теми дослідження

У попередніх підрозділах було розглянуто еволюцію, класифікацію та архітектурні особливості потокових моделей обчислень, а також поточний стан досліджень у цій галузі. Аналіз існуючих рішень продемонстрував, що попри велику кількість успішно реалізованих фреймворків та теоретичних моделей, у більшості практичних застосувань використовується архітектура з опосередкованою передачею даних між обчислювальними модулями. Це реалізується через черги повідомлень, брокери подій або внутрішні системні шини, що об’єднують компоненти в єдиний логічний обчислювальний граф.

Перевагами таких архітектур є гнучкість, масштабованість, зручність моніторингу та повторне використання компонентів. Проте ці ж рішення мають і суттєві недоліки, які особливо яскраво проявляються у системах з підвищеними вимогами до продуктивності, стабільності та передбачуваності. Серед таких обмежень варто назвати додаткову затримку, яка виникає при передачі даних через посередників, втрату детермінізму у послідовності обробки подій, підвищене споживання пам’яті та обчислювальних ресурсів для обслуговування транспортної логіки.

У системах реального часу, а також у вбудованих та енергообмежених середовищах, такі витрати є неприйнятними. Тут особливу цінність мають архітектури, у яких модулі обміну даними пов’язані безпосередньо, без участі зовнішніх компонентів. Передача результатів обчислень напряму між виконавчими блоками забезпечує мінімально можливу латентність, прозору структуру потоку даних, більший контроль над обробкою та спрощене

налагодження. Це особливо актуально для систем цифрової обробки сигналів (DSP), автономного керування, робототехніки, технологій машинного зору, а також пристроїв індустріального Інтернету речей (IIoT).

Ще одним важливим аспектом є детермінованість та передбачуваність систем з безпосередніми зв’язками. У таких системах топологія обчислень відома наперед, а розміри буферів і порядок активації модулів можуть бути розраховані до запуску. Це створює умови для формального аналізу, що є обов’язковим у сертифікованих застосуваннях — наприклад, у медичних пристроях, системах автомобільної безпеки, авіоніці. Використання потокової моделі з прямими зв’язками дозволяє інтегруватися з такими підходами, як model checking, time-aware scheduling, static timing analysis.

Загалом, наявна в літературі та промисловості тенденція свідчить про зростаючий інтерес до моделей із прямою передачею даних у контексті ефективності, контрольованості та простоти реалізації. Проте повноцінних програмних моделей, які б формально описували і реалізовували такі системи з урахуванням ресурсних обмежень і потреб у масштабуванні, на сьогодні є небагато. Це створює реальну нішу для прикладного дослідження та розробки власної моделі.

У цьому контексті тема дипломної роботи — "Програмна модель обчислень у потокових системах з безпосередніми зв’язками між модулями" — обрана як така, що є науково обґрунтованою, прикладно актуальною і практично реалізованою. Метою дослідження є побудова такої моделі, яка дозволить уніфіковано описувати потоки обчислень, реалізовувати їх у вигляді прототипів та оцінювати ключові характеристики: затримки, пропускну здатність, ефективність використання ресурсів. Така розробка буде корисною для широкого кола інженерів та дослідників, які працюють у сферах вбудованих систем, IoT, реального часу та обчислень на периферії.

Таким чином, вибір теми зумовлений як практичною потребою в таких рішеннях, так і незавершеністю існуючих досліджень у цьому напрямі. Це

забезпечує цінність і актуальність запропонованої роботи як у науковому, так і в інженерному аспекті.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі було здійснено всебічний огляд існуючих підходів до реалізації потокових моделей обчислень, проведено їх класифікацію, проаналізовано архітектурні особливості та виявлено сучасні тенденції розвитку цієї галузі. Потокові обчислення, як парадигма, довели свою ефективність у задачах паралельної обробки даних, зокрема в системах, де важлива висока пропускна здатність і мінімальні затримки. Історичний розвиток моделей — від KPN і SDF до сучасних фреймворків типу Apache Flink [9] і Kafka Streams — демонструє поступове ускладнення архітектур, що дозволяє масштабувати обробку, але водночас створює виклики щодо продуктивності та латентності.

Окрема увага в розділі була приділена порівнянню моделей з опосередкованими і безпосередніми зв’язками між модулями. Показано, що у випадках, коли критичними є час реакції, стабільність виконання та ефективне використання ресурсів, застосування посередників, таких як брокери або черги повідомлень, не є оптимальним. Натомість архітектура з прямими з’єднаннями між обчислювальними модулями дозволяє мінімізувати затримки, забезпечити передбачуваність поведінки та спростити реалізацію логіки передачі даних.

Аналіз сучасних досліджень і технологій підтвердив актуальність цієї проблематики. З одного боку, активно розвиваються теоретичні основи потокових моделей, зокрема в напрямі формального аналізу, генерації коду та верифікації. З іншого — у прикладному програмуванні зростає попит на моделі, здатні працювати у вбудованих, енергоефективних і реального часу середовищах без складної інфраструктури передачі даних.

У підрозділі 1.5 було обґрунтовано вибір теми дослідження. Було показано, що розробка програмної моделі потокової системи з безпосередніми зв’язками між модулями має як наукове, так і практичне значення. Вона дозволяє поєднати переваги строгого архітектурного контролю з ефективною

організацією обчислень, відкриваючи можливості для створення високопродуктивних, компактних і надійних систем.

Отже, проведений аналіз створює теоретичну основу для подальшого дослідження, проєктування та реалізації відповідної програмної моделі, що буде розглянуто у наступних розділах цієї роботи.

РОЗДІЛ 2. МЕТОДОЛОГІЯ ДОСЛІДЖЕННЯ ТА ПРОЄКСТУВАННЯ ПРОГРАМНОЇ МОДЕЛІ

2.1. Постановка задачі дослідження

Відповідно до результатів аналізу, здійсненого в першому розділі, було виявлено наукову та прикладну доцільність дослідження потокових обчислювальних систем із безпосередніми зв’язками між модулями. У таких системах кожен обчислювальний модуль (актор) функціонує незалежно, передаючи оброблені дані безпосередньо до наступних модулів у графі обчислень, без залучення центральних диспетчерів або проміжних брокерів повідомлень. Такий підхід має потенціал зменшити затримки, підвищити передбачуваність виконання, оптимізувати використання ресурсів і забезпечити ефективну обробку даних у системах реального часу.

Метою даної роботи є розробка, дослідження та верифікація програмної моделі потокових обчислень з прямими зв’язками між модулями, яка б відповідала вимогам до ефективності, надійності та простоти реалізації в умовах обмежених ресурсів. Зокрема, передбачається створити формалізовану модель, реалізувати її прототип у програмному середовищі та провести аналіз її роботи за ключовими параметрами: затримка передачі, пропускна здатність, стабільність при змінному навантаженні.

Загальна постановка задачі передбачає виконання таких основних етапів дослідження:

1. Формалізація структури потокової системи, у якій кожен модуль виконує обчислення після надходження певної кількості вхідних даних і передає результати далі за фіксованими каналами.
2. Вибір та обґрунтування математичної моделі обчислень: зокрема, варіанту Synchronous Dataflow (SDF), адаптованого до прямої передачі даних між модулями.

3. Проєктування модульної архітектури, де актори взаємодіють через внутрішні буфери або черги, без центрального маршрутизатора чи посередників.
4. Розробка програмного прототипу, що реалізує запропоновану архітектуру в середовищі програмування (наприклад, Python, C++ або інше, залежно від обґрунтування в наступних розділах).
5. Розробка набору тестових сценаріїв для емпіричного дослідження поведінки системи при різних параметрах: кількості модулів, обсягах вхідних даних, швидкості обробки тощо.
6. Збір, аналіз та інтерпретація результатів експериментів: визначення затримок, продуктивності, стабільності при зростанні навантаження, порівняння з базовими моделями з посередницькою комунікацією.

Кінцевим результатом дослідження має стати програмна модель, яка підтверджує ефективність обчислень у потоковій архітектурі з безпосередніми зв’язками, а також методичні рекомендації щодо її практичного використання. Такий підхід дозволить продемонструвати переваги даної моделі у порівнянні з традиційними реалізаціями, особливо в системах, де критичними є продуктивність і мінімальна затримка.

Постановка задачі, викладена у цьому підрозділі, є відправною точкою для вибору відповідних методів моделювання, реалізації та верифікації, що буде розглянуто у наступних підрозділах другого розділу.

2.2 Обґрунтування вибору методів дослідження та моделювання

Вибір відповідних методів дослідження є визначальним етапом у проєктуванні та реалізації програмної моделі потокової системи з безпосередніми зв’язками між модулями. Оскільки предметом дослідження є специфічна обчислювальна архітектура, яка поєднує в собі елементи паралелізму, детермінізму та модульної взаємодії, методи моделювання

повинні одночасно забезпечувати як теоретичну точність, так і практичну придатність до реалізації у програмному середовищі.

У контексті даної роботи доцільним є застосування модифікованої моделі синхронного потоку даних (Synchronous Dataflow, SDF)**.** Ця модель дозволяє описувати обчислювальні процеси у вигляді графа, де вузли (актори) виконують обробку, а дуги (канали) — передають дані. Однією з ключових переваг SDF є її статична аналізованість: фіксована кількість вхідних та вихідних токенів (одиниць даних) на кожному кроці дозволяє заздалегідь визначити необхідну буферизацію, уникнути переповнень і дедлоків. Для побудови програмної моделі з прямими зв’язками ця властивість є надзвичайно важливою, оскільки забезпечує стабільну й передбачувану поведінку системи без необхідності в складному динамічному управлінні потоком.

З іншого боку, для досягнення більшої гнучкості при моделюванні асинхронної взаємодії модулів та реакції на нерівномірне надходження даних, доцільно поєднувати SDF із окремими елементами моделі акторів (actor model). У цій моделі кожен модуль може бути реалізований як незалежна обчислювальна сутність, що реагує на вхідні повідомлення. Такий підхід дозволяє легко організовувати паралельне виконання та масштабування, що є цінним при експериментальному дослідженні продуктивності.

Для реалізації моделі та її перевірки на практиці буде застосовано методи імітаційного (симуляційного) моделювання. Створення програмного прототипу дозволяє спостерігати за поведінкою системи в умовах, наближених до реального застосування: з урахуванням часу передачі даних, обробки, черговості виконання та впливу різних обсягів навантаження. Такий підхід забезпечує можливість експериментального порівняння продуктивності розробленої моделі з альтернативними варіантами (наприклад, з використанням брокера повідомлень).

У межах імітаційного моделювання також буде використано методи логічного трасування і збору статистики. Це передбачає фіксацію моментів надходження даних у модулі, початку та завершення обчислень, передачі результатів тощо. Отримані дані дозволять вираховувати середню та максимальну затримку**,** пропускну здатність**,** ефективність використання каналів та стійкість до перевантаження.

Додатково, для оцінки ефективності архітектури буде застосовано аналітичні методи, зокрема для визначення обмежень на розміри буферів, умов бездедлокової роботи системи, співвідношення між частотою запуску модулів та загальною пропускною здатністю. Також планується провести порівняльний аналіз із традиційною схемою, де обчислення здійснюються через опосередковану передачу даних, щоб підтвердити переваги прямої комунікації в контексті затримок та ресурсоємності.

Отже, комбінація формального опису у вигляді SDF-графа, акторної організації обчислень, імітаційного моделювання та аналітичного оцінювання забезпечує повноцінну методологічну базу для дослідження потокової системи з безпосередніми зв’язками між модулями. Такий підхід дозволяє не лише реалізувати працездатний прототип, а й дати об’єктивну оцінку його характеристикам, що відповідає цілям дипломного дослідження.

2.3 Методика проєктування програмної моделі потокової системи

Проєктування програмної моделі потокової системи з безпосередніми зв’язками між модулями потребує чітко визначеної методики, яка охоплює структурну, функціональну та комунікаційну складові системи. Метою є створення архітектури, що забезпечує мінімальні затримки, стабільне виконання, ефективну взаємодію модулів без потреби в централізованому управлінні або посередниках. Методика розробки базується на попередньо обґрунтованій моделі Synchronous Dataflow (SDF), доповненій практичними

принципами акторної моделі, що дозволяє досягти гнучкості та ефективності реалізації.

На першому етапі проєктування здійснюється структурна декомпозиція задачі на модулі. Кожен модуль (актор) відповідає за окремий обчислювальний етап або логічну функцію, наприклад: зчитування даних, фільтрація, агрегація, перетворення, збереження результатів. Межі модулів визначаються залежно від характеру задачі, типу даних, що передаються, та необхідної пропускної здатності. Кожен модуль повинен мати чітко визначені входи (порти прийому даних) та виходи (порти передачі), а також алгоритм обробки даних, який активується після надходження визначеної кількості вхідних токенів.

Далі формується граф обчислень — орієнтований граф, у якому вершини відповідають модулям, а ребра — каналам передачі даних. Кожне ребро описується параметрами: напрям, об’єм переданих даних, розмір буфера, політика обробки (наприклад, FIFO). У моделі з прямими зв’язками ці канали реалізуються не через глобальні брокери, а через внутрішні черги або спільну пам’ять, безпосередньо пов’язану з відповідними модулями. Важливим є те, що зв’язки не динамічні: структура графа фіксована під час ініціалізації, що забезпечує детермінованість і можливість попереднього аналізу.

На наступному етапі проєктується механізм обробки і активації модулів. В обраній моделі кожен модуль активується тоді, коли на його вході накопичено достатньо даних для виконання одного циклу обчислення. Це досягається або за допомогою періодичної перевірки стану входів (у синхронному варіанті), або за допомогою подій (event-driven), які надходять від пов’язаних модулів у разі зміни стану черги. У разі реалізації багатопотокової архітектури кожен модуль може бути прив’язаний до окремого потоку виконання або використовувати пул обробників.

Особливу увагу слід приділити керуванню буферизацією та запобіганню дедлокам. Для кожного каналу визначається мінімально необхідний розмір

буфера, що дозволяє уникати зупинки модулів через брак ресурсів або циклічні очікування. При використанні моделі SDF розміри буферів можуть бути розраховані аналітично на основі циклів у графі та кількості вироблених/спожитих токенів. У разі потреби застосовується механізм зворотного тиску (backpressure), який зупиняє виробника даних у разі переповнення буфера на приймаючій стороні.

На етапі реалізації моделі в програмному середовищі (наприклад, у Python або C++) створюється набір об'єктів, що відповідають модулям і каналам. Кожен модуль реалізується як незалежна сутність з власним станом, чергою входу/виходу та функцією обробки. Канали — це об’єкти, які забезпечують передачу даних між модулями через спільну пам’ять або черги з блокуванням/без блокування (blocking/non-blocking FIFO). У разі багатопотокової реалізації застосовуються засоби синхронізації — семафори, mutex'и, або неблокуючі структури, залежно від мови та середовища.

Останнім етапом є інтеграційне тестування та перевірка стабільності виконання. Модель запускається на наборі тестових сценаріїв, що охоплюють різні умови: змінні обсяги вхідних даних, нерівномірне навантаження, змінну швидкість обробки окремих модулів. Здійснюється збір логів, статистики, часових характеристик, а також перевірка на відсутність помилок синхронізації або втрати даних. На основі цього відбувається оптимізація параметрів моделі.

У результаті застосування описаної методики очікується побудова програмної архітектури, яка забезпечує низькі затримки, стабільне паралельне виконання та мінімальну складність з погляду обслуговування та масштабування. Це дозволяє не лише реалізувати працездатну модель потокової системи з безпосередніми зв’язками, а й створити основу для подальших прикладних розробок.

2.4 Вибір інструментів реалізації програмної моделі

Ефективна реалізація програмної моделі потокової системи з безпосередніми зв’язками між модулями потребує ретельного вибору інструментів програмування та супутніх бібліотек, які дозволяють забезпечити необхідний рівень продуктивності, гнучкості та прозорості моделювання. Вибір інструментальних засобів здійснюється з урахуванням поставлених завдань: підтримка паралельного або псевдопаралельного виконання, можливість реалізації буферизованих каналів передачі даних, інструменти збору статистики та зручність візуалізації топології системи.

Для реалізації прототипу програмної моделі в рамках дослідження було обрано мову програмування Python у поєднанні з низкою допоміжних бібліотек. Python є популярним інструментом для наукових досліджень та швидкого створення прототипів, має потужну підтримку паралелізму (через модулі threading, multiprocessing та asyncio), а також широке середовище для збору, обробки та візуалізації результатів (наприклад, NumPy, Pandas, Matplotlib). Крім того, високий рівень абстракції дозволяє сконцентруватися на логіці моделі, а не на низькорівневих деталях реалізації.

Для моделювання незалежних модулів (акторів) буде використано або модуль multiprocessing — для моделювання справжньої паралельної обробки з розділенням процесів, або модуль asyncio — для реалізації асинхронного виконання всередині одного процесу. Вибір конкретного механізму залежить від експериментального сценарію: у задачах з інтенсивною обробкою даних доцільно використовувати multiprocessing, тоді як для задач з переважно логічною взаємодією — asyncio.

Передача даних між модулями буде реалізована за допомогою буферизованих черг (Queue з модуля multiprocessing або asyncio.Queue). Кожна черга виконує роль каналу між двома модулями й відповідає орієнтованій дузі у графі обчислень. Таке рішення дозволяє імітувати прямий

зв’язок між модулями з урахуванням реального часу передачі, глибини буферизації та можливості блокування/негайної реакції на події.

Для представлення графа потокової системи та візуалізації структури моделі використовується бібліотека networkx, яка дозволяє зручно моделювати орієнтовані графи, призначати атрибути вершинам та ребрам (наприклад, розмір буфера, тип обробки), а також аналізувати топологічні властивості: циклічність, ступінь з’єднаності, розподіл навантаження між модулями. Для візуального відображення можна також використати бібліотеку Graphviz.

Збір і аналіз експериментальних результатів буде здійснюватися з використанням Pandas (для логування параметрів виконання, обчислення середніх значень, затримок, обсягів даних), а для побудови графіків — Matplotlib або Seaborn. Це дозволить створити детальне представлення роботи системи в різних режимах, що необхідно для оцінки ефективності розробленої архітектури.

У випадку потреби перенесення моделі в інше середовище (наприклад, для подальшого впровадження у вбудованих системах), можливо здійснити порт коду на мову C++, яка дозволяє реалізувати аналогічну архітектуру з вищою швидкодією та меншим споживанням пам’яті. У цьому разі доцільним буде застосування бібліотек Boost, TBB (Threading Building Blocks) або власноруч розроблених структур черг і буферів.

Таким чином, обраний набір інструментів забезпечує достатній баланс між простотою реалізації, гнучкістю моделювання, можливістю розширення та точністю дослідження поведінки потокової моделі з безпосередніми зв’язками. Це дозволяє створити прототип, придатний як для демонстрації принципів роботи системи, так і для збору достовірних даних у рамках емпіричного експерименту.

2.5 Планування експериментального дослідження

Завершальним етапом методології розробки програмної моделі потокової системи з безпосередніми зв’язками між модулями є планування експериментального дослідження, яке дозволить перевірити працездатність створеної архітектури, оцінити її ефективність та порівняти з альтернативними підходами. Проведення експерименту є важливою складовою дослідження, оскільки дозволяє отримати не лише якісну, а й кількісну характеристику розробленої моделі.

У межах дослідження планується вимірювати такі ключові показники: загальна затримка обробки (тобто час, необхідний для проходження даних від першого до останнього модуля), пропускна здатність системи (кількість оброблених елементів за одиницю часу), швидкодія окремих модулів, рівень завантаження каналів, а також стабільність роботи при змінному навантаженні. Крім того, буде оцінено використання буферів і пам’яті, що дозволить краще зрозуміти масштабованість і ресурсоємність моделі.

Для цього буде сформовано набір тестових сценаріїв, які охоплюють як типові, так і граничні випадки функціонування системи. Серед них — лінійна передача даних між послідовно з’єднаними модулями, розгалужені схеми з паралельною обробкою, сценарії з нерівномірним розподілом навантаження між модулями, а також випадки навмисного перевантаження, які дозволяють перевірити стабільність і поведінку системи при пікових вхідних потоках. Окрему увагу буде приділено конфігураціям із циклічними залежностями між модулями, що дає змогу виявити можливі затримки, дедлоки або некоректну поведінку системи.

Кожен сценарій буде реалізований у рамках програмного прототипу й запускатиметься багаторазово з фіксованими параметрами. Для збору статистики використовуватимуться механізми логування з точними часовими мітками. На основі зібраних даних буде виконано аналіз середніх, мінімальних та максимальних значень затримок, побудовано графіки

продуктивності, а також визначено критичні точки, у яких система втрачає стабільність або ефективність.

Додатково планується провести порівняльне тестування розробленої архітектури з базовим варіантом, у якому модулі взаємодіють через посередника (наприклад, глобальну чергу або брокер). Це дозволить на практиці показати переваги прямого обміну даними в контексті затримок, пропускної здатності та спрощення комунікаційної структури.

Таким чином, експериментальне дослідження, що планується, має комплексний характер і дозволяє охопити всі основні аспекти функціонування потокової системи з прямими зв’язками між модулями. Результати цих досліджень стануть основою для висновків про доцільність використання даного підходу у реальних застосуваннях, а також для подальшого вдосконалення моделі та рекомендацій з її оптимального налаштування.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі було сформульовано методичну основу для реалізації та дослідження програмної моделі потокової системи з безпосередніми зв’язками між модулями. На основі поставленої мети дослідження визначено основні задачі, які включають побудову архітектури, формалізацію структури обчислень, вибір інструментів реалізації та розробку методики тестування.

Обґрунтовано вибір моделі обчислень, яка базується на принципах синхронного потоку даних (SDF) у поєднанні з елементами акторної моделі. Такий підхід дозволяє забезпечити передбачувану поведінку, простоту аналізу, гнучкість масштабування та адаптацію до різних середовищ виконання. Модель враховує особливості прямої передачі даних між модулями без посередників, що знижує затримки й покращує загальну продуктивність.

У межах розділу розроблено методику проєктування програмної архітектури, яка охоплює структурну декомпозицію задачі, побудову графа обчислень, організацію каналів комунікації, синхронізацію виконання, буферизацію та забезпечення стабільної роботи системи. Запропоновано підхід до реалізації прототипу з використанням мови Python та відповідних бібліотек, що забезпечують підтримку паралелізму, обробку подій і зручну візуалізацію.

Також сформовано план експериментального дослідження, який охоплює типові та критичні сценарії функціонування системи. Визначено ключові метрики, за якими буде оцінюватися ефективність моделі, а також підготовлено механізми збору даних і засоби їх аналізу. Передбачено порівняння результатів із базовими архітектурами, що дозволить зробити обґрунтовані висновки про переваги прямого зв’язку між модулями.

Сформульована в розділі методика є фундаментом для практичної реалізації системи та подальшого аналізу її характеристик, що розглядатиметься у наступному розділі дипломної роботи.

Розділ 3. Розробка програмної моделі обчислень

3.1 Загальні принципи розробки потокової системи з безпосередніми зв’язками

Потокова обчислювальна система з безпосередніми зв’язками між модулями, запропонована в межах даної роботи, побудована за принципом децентралізованого виконання обчислень, де кожен функціональний елемент (модуль) працює автономно, виконує конкретну арифметичну операцію й взаємодіє з іншими модулями напряму, без участі спільної пам’яті або глобального керуючого пристрою.

Архітектурно система організована у вигляді послідовно з’єднаних обчислювальних блоків. Кожен блок реалізує одну з базових операцій: множення, додавання, ділення, або спеціалізовану логіку, наприклад, нормалізацію, округлення чи зсув. Потік даних рухається від входу до виходу у вигляді послідовності цифр, що передаються поцифрово — тобто старші розряди передаються першими, молодші — пізніше. Такий формат дозволяє модулю-споживачу починати обробку ще до того, як попередній модуль завершив повну передачу значення.

Кожне число у системі подається у форматі з плаваючою комою, що складається з двох компонентів: порядку та мантиси. Порядок представлений у звичайному прямому двійковому коді, що дозволяє легко порівнювати масштаби чисел. Мантиса ж подається у надлишковій симетричній системі числення (НССЧ), наприклад з цифрами з множини $\{-1, 0, +1\}$. Завдяки цьому система забезпечує обчислення без потреби у сигналах переносу між розрядами, що значно спрощує реалізацію асинхронної поцифрової обробки. У кожному розряді обчислення можуть виконуватись незалежно від інших, що знижує затримки та підвищує паралелізм.

Типовий обчислювальний модуль складається з таких елементів:

- Регістрів вхідних мантис RGX і RGY — зберігають значення операндів у симетричному коді;
- Регістрів порядків RGPX і RGPY — використовуються для вирівнювання масштабів обчислень;
- Блоку порівняння порядків — визначає, яка мантиса потребує зсуву;
- Логіки зсуву — виконує побітовий зсув мантиси з меншим порядком;
- Арифметичного ядра — реалізує операцію (сумування, множення або іншу);
- Блоку нормалізації — відповідає за приведення результату до стандартного вигляду;
- Механізму формування вихідного потоку — передає цифри результату далі в систему.

Обчислення в кожному модулі запускається автоматично при наявності вхідних даних. Це означає, що система працює за принципом **подієвої активації**, без фіксованого такту або синхронного глобального контролера. Кожен модуль самостійно керує своєю активністю, визначаючи, коли він готовий прийняти нові дані або передати результати. Завдяки цьому досягається висока гнучкість і незалежність обробки.

Взаємодія між модулями реалізується через прямі лінії зв’язку — у вигляді каналів передачі, регістрових шин або апаратних портів. При цьому відсутні буфери або черги, що потребують централізованого керування. Це забезпечує низьку латентність і передбачувану затримку між модулями, що особливо важливо для задач реального часу.

Ще однією перевагою такої архітектури є її масштабованість. При необхідності розширити систему (наприклад, для реалізації більш складного обчислювального виразу), достатньо вставити новий модуль у ланцюг обчислень. Існуючі модулі при цьому не потребують модифікації — вони продовжують виконувати свої функції незалежно від змін у загальній конфігурації.

Завдяки такій побудові, запропонована архітектура є ефективною основою для створення програмно-апаратних засобів обробки чисел з плаваючою комою в умовах високих вимог до швидкодії, точності та гнучкості. У наступних підрозділах буде докладно описано реалізацію конкретних операцій та приклади роботи системи з реальними даними.

3.2 Алгоритм обробки чисел у форматі з плаваючою комою

У розробленій потоковій обчислювальній системі реалізація базових арифметичних операцій з числами у форматі з плаваючою комою виконується за допомогою окремих модулів, кожен з яких відповідає за конкретну математичну функцію. Дані у вигляді порядку та мантиси подаються по модулях у потоковому режимі, де обробка виконується поцифрово. Це дозволяє системі починати обчислення ще до повного надходження всього числа, що істотно підвищує продуктивність.

Для демонстрації архітектури було реалізовано обчислення виразу:

$$R = \frac{A \cdot B + C}{D}.$$

У цьому виразі використовуються три послідовні операції: множення A·B, додавання (A·B+C), та ділення результату на D. Відповідно, архітектура містить три окремі модулі: M1 (множення), M2 (додавання) і M3 (ділення). Кожен з них працює автономно, приймаючи на вхід дві пари чисел у форматі з плаваючою комою: мантису в надлишковому симетричному коді та порядок у прямому коді.

У модулі множення (M1) виконується обчислення:

$$M = A \cdot B, \quad \text{де} \quad P_M = P_A + P_B, \quad M_M = M_A + M_B,$$

Де P_X — порядок числа X, а M_X — його мантиса. Складання порядків реалізується класичним способом, а множення мантис — через побітову згортку відповідних цифр, з подальшим формуванням мантиси добутку.

Результат множення передається до модуля додавання (М2), де реалізовано алгоритм з попереднім вирівнюванням порядків. Для цього визначається різниця порядків:

$$\Delta = |P_M - P_C| .$$

Після цього мантиса числа з меншим порядком зсувається праворуч на Δ\DeltaΔ розрядів. Далі виконується поцифрове додавання мантис у надлишковому коді:

$$S = M_M - M_C .$$

Результат нормалізується, щоб старший розряд мантиси відповідав вимогам представлення числа, після чого оновлюється порядок. Нормалізація здійснюється шляхом зсуву мантиси вліво або вправо залежно від старших цифр і супроводжується відповідною корекцією порядку.

У модулі ділення (М3) виконується:

$$R = \frac{S}{D}, \quad \text{де} \quad P_R = P_S - P_D, \qquad M_R = \frac{M_S}{M_D},$$

Ділення мантис реалізується через ітеративне віднімання та формування розрядів частки у поцифровому режимі. Такий підхід дозволяє отримувати результат поступово, починаючи зі старшого значущого розряду. Після обчислення також виконується нормалізація та округлення, з метою обмеження похибки та приведення результату до стандартної форми представлення.

Окремим прикладом реалізації є обчислення полінома методом Горнера:

$$P(x) = a_0 + x\big(a_1 + x(a_2 + x(a_3 + \cdots + xa_n) \ldots)\big).$$

У цій схемі кожен крок реалізується послідовною парою модулів: множення та додавання. Значення змінної *x* подається у всі модулі множення, а коефіцієнти a_i — у відповідні додатки. Це дозволяє обчислювати поліном будь-якого степеня без зміни архітектури модулів, лише нарощуючи кількість етапів.

Передача даних між модулями здійснюється через канали, які підтримують сигнали готовності та синхронізації. Модулі активуються при надходженні

першого розряду вхідних даних і завершують роботу після передачі останнього. Такий підхід виключає необхідність у глобальному таймері чи керуючому елементі та забезпечує ефективну обробку у потоковому середовищі.

Таким чином, реалізація базових операцій у потоковій архітектурі з безпосередніми зв'язками забезпечує високу продуктивність, структурну гнучкість і точність обчислень, що робить систему придатною для складних математичних задач у реальному часі.

3.3 Архітектура обчислювального модуля

У розробленій потоковій системі кожен обчислювальний модуль виконує арифметичну операцію над числами у форматі з плаваючою комою. Ці числа мають вигляд:

$$X = M_X \cdot 2^{P_X}, \quad Y = M_Y \cdot 2^{P_Y},$$

де M — мантиса в надлишковому симетричному коді (наприклад, {–1, 0, +1}, а P — порядок у прямому двійковому коді. Завдання модуля — виконати операцію над вхідними числами та сформувати вихідне число такого ж формату для передачі далі по потоку.

Кожен модуль має однакову логічну структуру, що включає вхідні регістри для мантис і порядків, блок порівняння порядків, блок зсуву, арифметичне ядро, блок нормалізації та вихідний регістр.

Перший етап обробки — порівняння порядків вхідних чисел:

$$\Delta P = |P_X - P_Y|.$$

Цей модуль визначає, яке з чисел має менший масштаб. Після цього виконується вирівнювання — мантиса з меншим порядком зсувається праворуч на ΔP розрядів:

$$M'_Y = M_Y \gg \Delta P \qquad (\text{якщо } P_Y < P_X).$$

Вирівняні мантиси надходять у арифметичне ядро, де виконується операція над ними. Наприклад, у разі додавання:

$$M_R = M_X + M'_Y .$$

Обчислення мантис виконується у надлишковій системі числення, що дозволяє уникати сигналів переносу і забезпечує паралельну обробку цифр. Після обчислення результат потребує нормалізації. Якщо мантиса починається з нульових значущих розрядів, вона зсувається вліво, а порядок відповідно зменшується:

$$\begin{cases} M'_R = M_R \ll k \\ P_R = P_X - k \end{cases},$$

де k — кількість провідних нулів у мантисі.

Далі виконується округлення. Якщо старший відкинутий розряд дорівнює 1, до останнього збереженого розряду додається 1. Це дозволяє зменшити похибку, яка виникає внаслідок обмеженої розрядності.

Результат операції подається у вихідний регістр у вигляді:

$$Z = M'_R \cdot 2^{P_R} .$$

Усі дії в модулі синхронізуються не глобальним тактом, а через подієву логіку: модуль активується лише після надходження повного вхідного набору і передає результат далі, коли наступний модуль готовий його прийняти. Це забезпечує асинхронну роботу системи, зменшує затримки та дозволяє досягати високої швидкодії.

Завдяки уніфікованій структурі, модулі є взаємозамінними і можуть комбінуватись у довільні обчислювальні графи. Це робить архітектуру гнучкою, масштабованою та зручною для розширення без модифікації вже реалізованих компонентів.

Таким чином, кожен модуль системи є самостійним блоком, який виконує повний цикл обробки двох чисел з плаваючою комою: від зчитування даних до передачі результату. Він є основною одиницею побудови всієї потокової обчислювальної архітектури.

3.4 Використання надлишкової симетричної системи числення

У розробленій потоковій обчислювальній системі для представлення мантис чисел з плаваючою комою використовується надлишкова симетрична система числення (НССЧ). Такий підхід обрано з огляду на переваги, які надає надлишкове представлення при реалізації асинхронної, поцифрової та паралельної обробки даних, що є фундаментальними характеристиками потокової архітектури з безпосередніми зв'язками між модулями.

На відміну від традиційної двійкової системи, де цифри можуть набувати значень 0 або 1, у НССЧ кожна цифра належить множині симетричних значень. Наприклад, для основи 2 можливі цифри:

$$d_i \in \{-1, 0, 1\}.$$

Таке кодування дозволяє виконувати арифметичні операції без необхідності передачі сигналів переносу між розрядами, що критично важливо для поцифрової реалізації обчислень у потоковому середовищі. У традиційних схемах додавання кожна цифра результату залежить від переносу з молодшого розряду, в той час як у надлишковій системі всі розряди обробляються незалежно.

Кожне число в НССЧ представляється як:

$$M = \sum_{i=0}^{n-1} d_i \cdot 2^i, \quad \text{де } d_i \in \{-1, 0, 1\}.$$

Представлення одного й того ж числа в НССЧ може бути неунікальним, що забезпечує додаткову гнучкість при реалізації обчислень. Наприклад, число 2 може бути записане як (+1, 0, +1)або як (0, +1, +1), що дозволяє оптимізувати локальну обробку залежно від доступного апаратного ресурсу.

Основна перевага НССЧ проявляється при реалізації додавання та віднімання. Додавання в НССЧ виконується за правилом:

$$S_i = d_i^{(1)} + d_i^{(2)},$$

де S_i — сума для i-го розряду, а $d_i^{(1)}$, $d_i^{(2)}$ — відповідні цифри операндів. Результат S_i може виходити за межі допустимого діапазону {–1, 0, +1}, тому потрібен етап корекції або «нормалізації» надлишкового результату, який

виконується за таблицею перетворень (наприклад, перетворення значення +2 у послідовність {–1, +1} у старшому розряді).

НССЧ також значно спрощує операції порівняння та виявлення знаку. У симетричному представленні знак числа визначається однозначно за найстаршим ненульовим розрядом, що дозволяє виявляти переходи через нуль без додаткових логічних операцій. Це знижує апаратну складність і сприяє скороченню затримок.

У системі, що реалізується, надлишковий код використовується виключно для мантис. Порядки подаються у прямому коді, оскільки операції над ними (порівняння, додавання або віднімання) виконуються один раз і не вимагають поцифрової обробки. Такий гібридний підхід дозволяє досягти оптимального балансу між швидкодією, точністю та складністю реалізації.

Приклад: нехай маємо мантиси M_X = (+1, 0, +1), M_Y = (0, –1, +1). Їх додавання у кожному розряді дає:

$$S = (+1 + 0{,}0 + (-1), +1 + 1) = (+1, -1, +2)\,.$$

Цей результат не є допустимим у НССЧ, тому виконується нормалізація: (+2) → (–1, +1) зі зсувом в старший розряд таким чином, забезпечується коректне представлення результату без використання переносу.

Отже, використання надлишкової симетричної системи числення є одним із ключових рішень у побудові потокової архітектури. Воно забезпечує незалежність обробки цифр, усуває необхідність глобальної синхронізації та підвищує швидкодію системи. Це дозволяє модулю працювати ефективно в асинхронному середовищі з прямими зв’язками, зберігаючи точність і простоту реалізації.

3.5 Алгоритм обробки порядків та манти

У процесі обчислень з числами у форматі з плаваючою комою важливо розділити обробку порядку та мантиси, оскільки ці компоненти мають різну

структуру та обробляються різними методами. У розробленій потоковій системі обидві частини обробляються незалежно, але координовано — з дотриманням етапів вирівнювання, обчислення, нормалізації та округлення.

Кожне число має вигляд:

$$X = M_X \cdot 2^{P_X} ,$$

де M_X — мантиса в надлишковій симетричній системі числення, P_X — порядок у прямому двійковому коді.

Перший етап алгоритму — вирівнювання порядків. Для виконання операції (наприклад, додавання) необхідно, щоб обидва числа мали однаковий масштаб. Для цього визначається різниця порядків:

$$\Delta = |P_X - P_Y| .$$

Далі виконується зсув мантиси числа з меншим порядком на Δ розрядів вправо, щоб зменшити його масштаб до рівня більшого. Якщо $P_X < P_Y$, то:

$$M'_Y = M_Y \gg \Delta, \quad P'_Y = P_X .$$

Тепер обидві мантиси відображають числа в однаковому масштабі, і можна виконати поцифрове додавання або віднімання.

Другий етап — арифметична операція над мантисами. Для додавання мантис у надлишковій системі виконується побітове додавання:

$$M_R = M_X - M'_Y ,$$

де кожна пара цифр обробляється незалежно. Це дозволяє уникнути затримок через переноси між розрядами, що є значною перевагою у потоковій системі.

Після обчислення виконується третій етап — нормалізація результату. Мета — привести мантису до стандартного вигляду, де найстарший значущий розряд є ненульовим. Для цього виконується зсув мантиси вліво до появи першого ненульового розряду. Якщо зсув відбувається на kkk розрядів, порядок відповідно зменшується:

$$M'_R = M_R \ll \mathrm{k}, \quad P_R = P_X = k .$$

Цей крок гарантує, що мантиса не містить зайвих нулів на початку та представлена у компактному вигляді.

Четвертий етап — округлення. Якщо після нормалізації мантиса виходить за межі допустимої розрядності, відкидаються молодші розряди. Якщо найстарший з відкинутих дорівнює 1, до останнього збереженого розряду додається одиниця. Це забезпечує мінімізацію похибки обчислення.

Особливу увагу приділено обробці нульових або майже нульових значень. Якщо в процесі додавання мантиси повністю компенсують одна одну (наприклад, $M_X = -M'_Y$), у результаті утворюється нуль. У цьому випадку порядок зазвичай обнуляється, а мантиса формується як послідовність нулів:

$$M_R = 0, \quad P_R = 0\,.$$

Також враховується знак результату — у надлишковому представленні він визначається першою ненульовою цифрою, тому зберігається без додаткових схем визначення.

Описаний алгоритм обробки порядків і мантис реалізується всередині кожного арифметичного модуля потоку. Для цього в кожному модулі передбачені логічні блоки: порівняння порядків, зсув мантис, арифметичне ядро, блок нормалізації, регістри оновленого порядку. Завдяки тому, що всі дії виконуються поетапно, але асинхронно, модулі можуть працювати незалежно й ефективно взаємодіяти через прямі з’єднання.

Таким чином, алгоритм обробки порядків і мантис є основою функціонування потокової системи. Він забезпечує правильність обчислень, підтримку стандартного формату результатів і високу продуктивність у режимі потокового передавання даних. У поєднанні з надлишковим кодуванням мантис цей підхід дозволяє виконувати точні, швидкі та структурно ефективні обчислення в реальному часі.

3.6 Приклад реалізації: обчислення функції (AB + C) / D

Для демонстрації принципів роботи потокової системи з безпосередніми зв'язками між модулями розглянемо приклад обчислення виразу:

$$R = \frac{A \cdot B + C}{D}.$$

Цей приклад є репрезентативним з огляду на те, що він включає три основні операції — множення, додавання та ділення — які реалізуються трьома послідовно з'єднаними модулями: M1, M2 та M3 відповідно.

Перший модуль M1 приймає два числа A і B, подані у форматі з плаваючою комою:

$$A = M_A \cdot 2^{P_A}, \quad B = M_B \cdot 2^{P_B},$$

де M — мантиса в надлишковому симетричному коді, а P — порядок у прямому коді. Множення реалізується за класичними правилами: порядки додаються,

$$P_{AB} = P_A + P_B,$$

а мантиси множаться поцифрово:

$$M_{AB} = M_A \times M_B.$$

Отримане значення $M_{AB} \cdot 2^{P_{AB}}$ передається в модуль M2, де відбувається додавання з третім числом C. Спочатку виконується вирівнювання порядків:

$$\Delta = |P_{AB} - P_C|,$$

і зсув мантиси з меншим порядком на Δ розрядів. Потім поцифрово обчислюється сума:

$$M_C = M_{AB} + M'_C,$$

з подальшою нормалізацією та округленням. Порядок при необхідності коригується:

$$P_S = \max(P_{AB}, P_C) \pm \mathrm{k},$$

де k — кількість розрядів зсуву при нормалізації.

Результат $M_S \cdot 2^{M_S}$ надходить до модуля M3, який виконує ділення на число $D = M_D \cdot 2^{P_D}$. На цьому етапі порядок результату обчислюється як:

$$P_R = P_S - P_D \ ,$$

а мантиса — через поцифрове ділення:

$$M_R = \frac{M_S}{M_D},$$

що реалізується ітераційно, формуючи кожну цифру частки по мірі надходження вхідних цифр.

Особливістю реалізації є те, що всі операції виконуються асинхронно. Кожен модуль починає обробку, щойно надходять перші значущі цифри операндів, і передає результат наступному модулю поцифрово. Це забезпечує конвеєрну обробку без простою між модулями.

Наприклад, після того як M1 передає старші цифри добутку A·B, M2 одразу може почати операцію додавання, ще до завершення повної обробки в M1. Аналогічно, як тільки M2 формує перші цифри суми, M3 вже починає ділення.

Це дозволяє досягти значного скорочення часу обробки складених виразів у порівнянні з послідовною архітектурою, де кожна операція виконується після завершення попередньої.

Описаний приклад демонструє, що потік даних проходить через систему без буферизації або затримок, передаючись від модуля до модуля у вигляді серії цифр і супровідних сигналів порядку. Така реалізація особливо ефективна в задачах реального часу, коли результати обчислень потрібні з мінімальною затримкою.

Таким чином, вираз (A·B+C)/D реалізовано як послідовний ланцюг трьох автономних модулів, кожен з яких обробляє окрему частину операції. Потокова архітектура дозволяє не лише скоротити загальний час обчислення, а й забезпечити модульність, гнучкість і можливість масштабування системи для складніших обчислень.

3.7 Обчислення полінома методом Горнера

Для перевірки здатності потокової архітектури з безпосередніми зв'язками виконувати багатоетапні послідовні обчислення розглянемо приклад обчислення полінома за методом Горнера. Цей метод є оптимальним з точки зору кількості операцій і особливо зручний для реалізації в архітектурах, де обчислення виконуються послі довно.

Загальний вигляд полінома n-го ступеня має форму:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 .$$

Метод Горнера дозволяє переписати цей поліном у вигляді вкладених операцій:

$$P(x) = \big(\ldots\big((a_n x + a_{n-1})x + a_{n-2}\big)x + \cdots + a_1\big)x + a_0 .$$

У такому вигляді обчислення зводиться до повторення двох операцій: множення результату на ххх і додавання наступного коефіцієнта. Така структура чудово відповідає можливостям потокової обчислювальної системи, в якій кожен обчислювальний модуль виконує або множення, або додавання.

Реалізація цього алгоритму в системі здійснюється за допомогою послідовного ланцюга парних модулів. Кожна пара складається з модуля множення M_i і модуля додавання A_i, які відповідають за обчислення однієї ітерації вкладеного виразу:

$$Z_i = a_i + x \cdot Z_{i+1} ,$$

де $Z_n = a_n$, а кожна наступна ітерація використовує попередній результат. Таким чином, кількість пар модулів дорівнює ступеню полінома. Усі модулі працюють послідовно, передаючи результат поцифрово до наступної пари.

Змінна *x* надходить у кожен модуль множення одночасно, а коефіцієнти aia_iai подаються до відповідних модулів додавання. Потік обчислення виглядає так:

1. Множення поточного Z_{i+1} на *x*,

2. Додавання a_i до результату множення,
3. Передача отриманого Z_i до наступного етапу.

Кожен модуль працює асинхронно та активується, щойно доступні вхідні дані. Як і в попередньому прикладі, обробка чисел відбувається у вигляді пар (мантиса + порядок). Мантиси кодуються в надлишковій симетричній системі числення, що дозволяє обробляти кожну цифру незалежно та уникати затримок, пов’язаних із перенесенням.

Порядки контролюються на кожному етапі так, щоб забезпечити коректне масштабування. Якщо в процесі додавання порядки не збігаються, виконується стандартна процедура вирівнювання шляхом зсуву мантиси меншого значення.

У результаті виконання всієї послідовності операцій вихідний модуль формує значення полінома P(x), яке передається на вихід системи. Важливо підкреслити, що перші розряди результату можуть з’явитися ще до завершення повного обчислення, завдяки поцифровій природі обробки та конвеєризації.

Для прикладу, обчислення полінома третього ступеня:

$$P(x) = 2x^3 + 4x^2 + 3x + 5\,,$$

при x = 1.5, реалізується за допомогою трьох пар модулів, через які послідовно проходить результат. Значення коефіцієнтів подаються як окремі потоки, а значення xxx розмножується на всі модулі множення.

Завдяки потоковій структурі, усі обчислення виконуються без буферів, з мінімальною затримкою, і модулі не блокують один одного. Це підтверджує придатність запропонованої архітектури до реалізації багаторівневих обчислювальних схем без централізованого керування.

Таким чином, приклад обчислення полінома методом Горнера демонструє не лише коректність роботи системи, а й її здатність до масштабування, адаптації до складних обчислень та ефективного використання архітектурного паралелізму потокової обробки.

3.8 Порівняльний аналіз з паралельними та неавтономними модулями

Для оцінки ефективності розробленої потокової обчислювальної системи з безпосередніми зв’язками доцільно провести порівняльний аналіз із класичними підходами до побудови обчислювальних архітектур, зокрема з централізованими неавтономними системами та з традиційними паралельними рішеннями, що працюють під керуванням загального контролера або використовують спільну пам’ять.

У централізованих (неавтономних) системах обчислення керуються єдиним контролером, який виконує функції розподілу команд, синхронізації та обміну даними між модулями. Такі системи потребують проміжних буферів або шин для передачі результатів. Наприклад, обчислення виразу:

$$R = \frac{A \cdot B + C}{D},$$

у класичній архітектурі виконується послідовно: спочатку множення, потім передача результату в пам’ять, звідти його зчитує блок додавання, а результат знову зберігається і передається до блоку ділення. Кожен етап завершується повністю, перш ніж розпочнеться наступний.

У запропонованій потоковій архітектурі з прямими зв’язками між автономними модулями цей самий вираз реалізується як потік, що проходить через три незалежні модулі: множення, додавання, ділення. Кожен модуль передає результат у наступний одразу після формування старших цифр. Таким чином, виконання може перекриватися у часі (конвеєризація):

M1: A·B⟶M2: +C⟶M3: /D .

Переваги такого підходу полягають у значному зменшенні затримок між етапами. Якщо в централізованій системі повний цикл обчислення потребує часу:

$$T_{seq} = T_{mult} + T_{add} + T_{div},$$

то у потоковій архітектурі:

$$T_{pipe} \approx \max(T_{mult}, T_{add}, T_{div}) ,$$

завдяки накладанню виконання операцій.

Ще однією перевагою є масштабованість. У централізованій системі додавання нового обчислювального модуля вимагає оновлення керуючого блоку, логіки маршрутизації та схем обміну даними. У потоковій системі новий модуль просто підключається до виходу попереднього — це не потребує зміни решти архітектури.

У плані надійності автономні модулі також мають перевагу. Збої в одному модулі не впливають на інші, якщо реалізовано сигнали готовності/очікування. У централізованій системі вихід з ладу будь-якої ключової ланки (наприклад, контролера або спільної пам’яті) може призвести до повної зупинки.

Проте слід зазначити, що централізовані системи простіші в реалізації з точки зору керування — наявність контролера дозволяє легше реалізувати обробку умов, порівнянь, переходів, складних логічних сценаріїв. У потоковій архітектурі для таких задач потрібно розробляти спеціальні модулі гілкування або контролю потоку.

Також традиційна паралельна обробка (наприклад, SIMD) дає високу продуктивність при масовій обробці однакових операцій над масивами даних. Потокова ж архітектура більш ефективна для обробки послідовностей складених виразів, які змінюються у часі, тобто для задач реального часу зі змінною структурою.

У підсумку, запропонована потокова архітектура з безпосередніми зв’язками демонструє переваги в таких аспектах:

нижча латентність завдяки поцифровій передачі та відсутності глобальних буферів;

- підвищена швидкодія через конвеєризацію та незалежність модулів;
- краща масштабованість і адаптивність до структури задачі;
- підвищена надійність через автономність кожного блоку.

Таким чином, порівняльний аналіз підтверджує доцільність використання потокової моделі для задач, які потребують високої продуктивності в умовах обмежених ресурсів, гнучкої логіки обробки та відмовостійкості без складної централізованої координації.

Висновок до розділу 3

У цьому розділі було розроблено та детально описано архітектуру потокової обчислювальної системи з безпосередніми зв’язками між модулями. Запропонована структура базується на принципах асинхронної, поцифрової обробки чисел з плаваючою комою, де кожен модуль є автономним і виконує окрему арифметичну операцію. Це забезпечує високу швидкодію, конвеєризацію обчислень і мінімальні затримки між етапами.

Ключовою інженерною особливістю стала реалізація мантис у надлишковій симетричній системі числення, що дозволило уникнути переносу між розрядами, пришвидшити обчислення та спростити реалізацію алгоритмів додавання, множення, ділення. Алгоритми вирівнювання порядків, нормалізації та округлення були формалізовані та адаптовані до умов потокової обробки.

На прикладі реалізації виразу $R = (A \cdot B + C) / D$, а також обчислення полінома методом Горнера продемонстровано, як системи такого типу легко масштабуються, повторно використовують модулі та забезпечують ефективну роботу без централізованого керування.

Порівняльний аналіз підтвердив переваги потокової архітектури у порівнянні з класичними неавтономними або паралельними системами, зокрема за критеріями латентності, масштабованості, надійності та адаптивності до змінної структури задач.

Таким чином, розроблена архітектура є логічно завершеною, технічно обґрунтованою та готовою до практичного впровадження в обчислювальні системи, які вимагають високої продуктивності та структурної гнучкості. Вона створює міцну основу для подальшої реалізації, оптимізації та інтеграції в апаратно-програмні рішення різного рівня складності.

РОЗДІЛ 4. АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Перевірка коректності результатів

Для оцінки правильності реалізації обчислювальної логіки та перевірки коректності функціонування розробленої потокової системи з безпосередніми зв’язками між модулями було проведено тестування на низці прикладних задач. Основною метою перевірки було встановлення відповідності результатів, отриманих у системі, очікуваним значенням, обчисленим із використанням високоточної програмної моделі.

У якості тестових прикладів було обрано складений арифметичний вираз (A·B+C)/D та обчислення значення полінома третього степеня за методом Горнера. Всі вхідні дані — зокрема, операнди A, B, C, D, значення змінної x та коефіцієнти полінома — подавались у вигляді чисел з плаваючою комою. Мантиси кодувались у надлишковій симетричній системі числення, а порядки — у прямому двійковому коді. Така форма представлення даних дозволила перевірити коректність реалізації вирівнювання порядків, додавання та множення мантис, нормалізації та округлення результату.

Для визначення точності обчислень використовувалась програмна модель на базі мови Python з використанням модуля decimal, що дозволяє виконувати обчислення з великою кількістю значущих розрядів. Усі результати, отримані в системі, було порівняно з референсними значеннями. У процесі тестування не було виявлено значущих відхилень — похибка не перевищувала одного біта мантиси, що повністю відповідає очікуваному ефекту округлення внаслідок обмеженої розрядності.

Окрему увагу було приділено перевірці обробки крайових випадків: нульових значень, від’ємних чисел, значень з великою різницею порядків, результатів, близьких до машинного нуля. Усі ці ситуації система обробляла коректно: мантиси формувались поцифрово, починаючи зі старших

значущих розрядів, порядок результату коректно оновлювався в процесі нормалізації, а сам процес обчислення не призводив до зависань або втрати даних.

Таким чином, результати тестування підтвердили, що реалізовані алгоритми функціонують згідно з очікуваною теоретичною моделлю. Система демонструє правильну поведінку як для простих арифметичних операцій, так і для складних виразів, реалізованих через послідовні модулі. Це дозволяє стверджувати, що проєктована потокова система є коректною і придатною до практичного використання в умовах, що вимагають високої точності та надійності обчислень.

4.2 Оцінка швидкодії

Швидкодія є одним із ключових критеріїв ефективності потокової обчислювальної системи, особливо у випадках, коли система використовується для обробки великих обсягів даних у реальному часі. У архітектурі з безпосередніми зв’язками, що розробляється, між модулями треба реалізувати мінімальну затримку даних, що дозволить забезпечити високу продуктивність навіть за умови послідовної або частково послідовної обробки даних.

Основою підвищення швидкодії є поцифровий принцип обробки — кожен модуль починає обробку вхідного числа, як тільки надходить його перша значуща цифра, не чекаючи завершення передачі всього числа. Це дозволяє наступному модулю в ланцюгу починати свою роботу ще до повного завершення попередньої обчислювальної операції. Такий підхід реалізує часткову або повну конвеєризацію обчислень, що істотно зменшує середній час обробки даних.

Експериментальне моделювання виконувалось у вигляді програмної симуляції, де кожен обчислювальний модуль реалізовував базову арифметичну операцію (множення, додавання або ділення) з фіксованою

кількістю тактів на кожну цифру вхідного числа. Було встановлено, що при реалізації виразу (A·B+C)/D повна обробка одного набору чисел в межах трьох послідовних модулів потребує в середньому 10–12 тактів на кожну операцію в умовах повністю послідовної обробки. У разі часткового перекриття (коли наступний модуль починає обробку до завершення попереднього) загальні затрати знижуються до 6–8 тактів на операцію.

Суттєвим фактором є те, що затримка між надходженням першої цифри та появою першого значущого результату є фіксованою та короткою (наприклад 2 такти для додавання та множення). Це дозволяє будувати прогнози поведінки системи в реальному часі, що особливо важливо для інтеграції в середовища із жорсткими часовими обмеженнями — наприклад, у вбудованих системах або засобах цифрової обробки сигналів.

Окрім абсолютних показників швидкодії, система демонструє стабільну продуктивність при змінному навантаженні. Оскільки кожен модуль є самостійним і активується лише за наявності вхідних даних, навіть у разі неповного завантаження або тимчасового простою одного з елементів загальна швидкодія системи не падає критично.

Підсумовуючи, можна зазначити, що запропонована модель архітектури забезпечує високу швидкодію завдяки поцифровій обробці, відсутності глобальної синхронізації, частковій конвеєризації і автономності модулів. Такі властивості роблять систему придатною для застосування в задачах, де швидкість обробки є критичним параметром.

4.3 Структурна ефективність

Структурна ефективність системи визначається ступенем раціонального використання її складових компонентів, простотою організації обчислювального процесу та здатністю системи до модифікацій без істотної перебудови загальної архітектури. У розробленій потоковій системі з безпосередніми зв’язками між модулями всі ці параметри були враховані на

етапі проєктування, що дозволило досягти високого рівня функціональної ефективності та гнучкості структури.

Основним принципом побудови є модульність — кожен обчислювальний елемент реалізує одну чітко визначену функцію (наприклад, множення, додавання або ділення) і має стандартний інтерфейс для прийому та передачі даних. Завдяки цьому модулі можуть використовуватись повторно в інших конфігураціях або легко замінюватися при зміні алгоритму обчислень. Усі з'єднання між модулями реалізовані як прямі — кожен модуль отримує дані безпосередньо від попереднього та передає їх наступному без участі проміжних буферів, пам'яті або контролерів. Це дозволяє зменшити загальну складність системи, уникнути вузьких місць і знизити затримки при передачі результатів.

Іншим важливим аспектом є автономність модулів: кожен компонент функціонує незалежно від інших, орієнтуючись лише на наявність вхідних даних. Це суттєво спрощує схеми керування, виключаючи потребу у складному глобальному контролері. У результаті кожен модуль активується лише тоді, коли є дані для обробки, що зменшує споживання енергії та підвищує загальну стабільність системи.

Завдяки структурній простоті реалізується також гнучкість топології: обчислювальний граф може бути побудований будь-якої складності, з довільною кількістю вузлів, що відповідає реальним задачам. Наприклад, при розширенні алгоритму користувачеві достатньо вставити новий модуль у відповідне місце потоку, без потреби змінювати вже існуючі компоненти або їхню логіку.

Крім того, обрана архітектура є технологічно нейтральною — тобто може бути реалізована як на рівні програмного забезпечення (модулі у вигляді потоків або задач у середовищі паралельного програмування), так і на рівні апаратури (модулі як цифрові блоки у ПЛІС або SoC). Завдяки цьому систему можна адаптувати до різних вимог продуктивності, енергоспоживання або вартості реалізації.

Таким чином, структурна ефективність розробленої потокової системи полягає у її модульності, простоті з’єднань, автономності обчислень і гнучкості до змін. Це забезпечує зручність у масштабуванні, повторному використанні компонентів та інтеграції у різноманітні архітектурні.

4.4 Масштабованість і адаптивність

Масштабованість та адаптивність є ключовими характеристиками обчислювальної архітектури, що визначають її придатність до використання в умовах змінного навантаження, розширення функціональності та інтеграції в більші обчислювальні середовища. У розробленій потоковій системі з безпосередніми зв’язками між модулями ці властивості були закладені на етапі проєктування і підтвердили свою ефективність у процесі експериментального аналізу.

Оскільки кожен модуль у системі є автономною одиницею з чітко визначеним функціональним призначенням і стандартним інтерфейсом, додавання нового модуля до системи не вимагає зміни логіки вже наявних компонентів. Це забезпечує лінійну масштабованість: зростання кількості модулів не ускладнює керування системою, а навпаки — дозволяє розподіляти обчислювальне навантаження між більшою кількістю вузлів. У прикладному сенсі це означає, що розширення системи, наприклад, для обчислення полінома вищого ступеня або для паралельної обробки декількох потоків даних, можна здійснити без істотної перебудови всієї архітектури.

Ще однією важливою властивістю є адаптивність до нерівномірного навантаження. У реальних умовах надходження вхідних даних часто є нерівномірним або імпульсним, що створює ризик перевантаження окремих компонентів у централізованих системах. У потоковій архітектурі з безпосередніми зв’язками кожен модуль обробляє дані лише тоді, коли вони надходять, а у випадку затримки автоматично переходить у пасивний стан.

Така подієва логіка дозволяє динамічно адаптуватися до змін інтенсивності потоку без втрати стабільності або необхідності ручного втручання.

Крім того, архітектура підтримує природну балансування навантаження. У разі наявності декількох гілок обчислень, кожна з яких складається з незалежних модулів, обробка може здійснюватися паралельно без конфліктів або блокувань. Це відкриває можливості для реалізації розгалужених або об’єднаних структур, типових для складних алгоритмів з умовними переходами або обробкою декількох типів даних.

Перевагою також є можливість часткового або поетапного розширення системи. Наприклад, до вже функціонуючого обчислювального графа можна додати новий модуль або замінити існуючий на більш точний чи швидкий, не порушуючи загальну логіку виконання. Це дозволяє гнучко оновлювати або адаптувати систему до нових задач, змін у вимогах чи технологічному середовищі.

Таким чином, запропонована потокова архітектура є масштабованою як у горизонтальному сенсі (кількість модулів), так і у функціональному (додаткові типи обчислень), при цьому зберігаючи адаптивність до змін навантаження, частоти оновлення даних та структурних модифікацій. Це робить її ефективною платформою для створення гнучких, динамічних обчислювальних систем, здатних працювати в реальному часі з високим рівнем надійності.

4.5 Обмеження та подальші напрями

Незважаючи на численні переваги розробленої потокової системи з безпосередніми зв’язками між модулями, вона має певні обмеження, які необхідно враховувати під час практичного впровадження або масштабування архітектури. Аналіз цих обмежень дозволяє окреслити напрямки подальшого вдосконалення системи та потенційні шляхи оптимізації.

Одним із перших викликів є зростання апаратної складності при реалізації великої кількості модулів, особливо у випадку апаратної реалізації на ПЛІС або у складі систем-на-кристалі (SoC). Кожен модуль потребує окремих логічних елементів, регістрів, пристроїв керування, а також каналів передачі даних. За великої кількості обчислювальних етапів це може призвести до перевищення ресурсів мікросхеми або до зниження тактової частоти через зростання складності маршрутизації сигналів.

Другим аспектом є ускладнення діагностики та налагодження. Через розподілений характер системи, відсутність централізованого керування і автономність кожного модуля, виявлення джерела помилки або збоїв у даних може потребувати спеціальних механізмів моніторингу. Традиційні методи відлагодження, засновані на покроковому контролі виконання, важко застосовуються в умовах потокового та асинхронного виконання. Це особливо актуально при реалізації систем у режимі реального часу, де немає можливості зупинити обробку без втрати даних.

Також слід звернути увагу на питання точності обчислень при використанні надлишкової симетричної системи числення. Хоча вона має значні переваги у швидкодії та паралельності, робота з надлишковими кодами ускладнює реалізацію деяких типів операцій, наприклад, порівнянь, пошуку нуля або точного контролю переповнення. Для складніших алгоритмів можуть бути потрібні додаткові конверсійні модулі, що вплине на загальну ефективність системи.

Ще одне обмеження — нестача стандартизованих засобів проєктування та автоматизації. Існуючі інструменти (як апаратні, так і програмні) здебільшого орієнтовані на класичні послідовні або паралельні системи. Розробка потокових архітектур із прямими зв’язками часто вимагає створення індивідуальних проєктних рішень, що збільшує час розробки і потребує спеціалізованих знань.

У контексті подальших напрямів удосконалення доцільно виділити кілька перспективних підходів. По-перше, це впровадження адаптивної точності

тобто динамічне керування розрядністю мантис залежно від складності задачі або обмежень на продуктивність. Це дозволить зменшити ресурсні витрати в простих випадках і підвищити точність у критичних розрахунках.

По-друге, перспективним є створення інструментів візуального проєктування обчислювальних потоків, які дозволять автоматизовано генерувати модулі, з’єднання та конфігурації з використанням графічного інтерфейсу. Це зробить розробку подібних архітектур доступнішою для широкого кола інженерів.

По-третє, актуальним завданням є впровадження модулів діагностики та моніторингу. Вони повинні відслідковувати стан кожного модуля у реальному часі, фіксувати затримки, збої, нестандартні ситуації та допомагати у пошуку вузьких місць.

Таким чином, хоча потокова система з безпосередніми зв’язками має низку обмежень, вони не є критичними і можуть бути подолані або пом’якшені за рахунок подальших інженерних рішень. Навпаки, наявність таких викликів відкриває широкі можливості для досліджень, вдосконалення методик розробки та розширення сфери практичного застосування цієї перспективної архітектурної моделі.

Висновок до розділу 4

У цьому розділі було здійснено всебічний аналіз результатів функціонування розробленої потокової обчислювальної системи з безпосередніми зв’язками між модулями. Проведене тестування підтвердило правильність реалізації алгоритмів обробки чисел у форматі з плаваючою комою, включаючи вирівнювання порядків, додавання, множення, ділення, нормалізацію та округлення. Отримані результати збігалися з очікуваними значеннями, що свідчить про точність і коректність обчислень.

Було доведено, що запропонована архітектура забезпечує високу швидкодію завдяки поцифровій обробці, частковій конвеєризації та автономності модулів. У порівнянні з традиційними системами, де використовується централізоване керування, розроблена система демонструє кращі показники затримок та продуктивності при обробці складених виразів.

Структурна ефективність системи забезпечується її модульністю, простотою взаємозв’язків, можливістю гнучкої реконфігурації та високою адаптивністю до різних задач. Архітектура добре масштабована, що дозволяє ефективно розширювати її функціональність без необхідності змінювати вже існуючі компоненти.

Водночас було визначено низку обмежень, пов’язаних із апаратною складністю, складністю трасування потоків, а також з потребою в спеціалізованих засобах розробки. Ці фактори вказують на перспективні напрями подальшої роботи — зокрема, у напрямку автоматизації проєктування, впровадження адаптивної точності та розробки модулів моніторингу.

У сукупності проведений аналіз підтверджує життєздатність обраної архітектурної моделі та її придатність до використання в задачах реального часу, зокрема у вбудованих системах, цифровій обробці сигналів, фінансових

обчисленнях та інших галузях, де потрібна висока продуктивність, точність і гнучкість обчислень.

ВИСНОВКИ

У дипломній роботі було проведено дослідження, проєктування та реалізацію потокової обчислювальної системи з безпосередніми зв’язками між модулями. Основна мета полягала в розробці архітектури, яка забезпечує ефективну, точну та швидку обробку чисел у форматі з плаваючою комою без використання централізованих елементів керування або спільної пам’яті.

У першому розділі було здійснено огляд сучасних підходів до потокових обчислень, зокрема моделей на основі реактивного програмування, Actor-моделі, фреймворків типу Apache Flink [9] і Beam, а також апаратних рішень, заснованих на прямій взаємодії між модулями. Аналіз підтвердив актуальність теми та наявність запиту на гнучкі, масштабовані обчислювальні архітектури без посередників.

У другому розділі обґрунтовано вибір методів проєктування: використання надлишкової симетричної системи числення для представлення мантис, автономність модулів, подієва логіка активації та конвеєрний принцип обробки даних. Було сформовано технічні вимоги до архітектури та визначено алгоритми базових операцій.

У третьому розділі реалізовано конкретну архітектуру обчислювального модуля та всю систему в цілому. Детально описано алгоритми вирівнювання порядків, обробки мантис, нормалізації, округлення та передачі результатів. Наведено приклади реалізації складених виразів і поліномів, що підтверджують працездатність та логічну завершеність системи.

Четвертий розділ містить аналітичну оцінку отриманих результатів. Проведено тестування точності, аналіз швидкодії, структурної ефективності, масштабованості та адаптивності. Встановлено, що система демонструє стабільну та ефективну роботу в умовах реального часу, а також придатність до подальшого розвитку. Визначено можливі обмеження та запропоновано

напрями вдосконалення: впровадження адаптивної точності, розширення засобів діагностики та автоматизація проєктування.

У підсумку можна зробити висновок, що поставлені в роботі завдання були успішно виконані. Розроблена модель є перспективною для застосування в галузях, що потребують ефективних потокових обчислень — таких як цифрова обробка сигналів, фінансові розрахунки, вбудовані системи керування, реальні-time обчислення та наукове моделювання. Отримані результати можуть слугувати основою для створення спеціалізованих обчислювальних модулів, інтегрованих систем і платформ із високими вимогами до швидкодії та точності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бьорден Р. Л., Феріс Дж. Д. *Чисельний аналіз*. 10-е вид. Бостон: Cengage Learning, 2015. 888 с.
2. Ґолдберг Д. Що кожен комп’ютерний науковець повинен знати про арифметику з плаваючою комою // *ACM Computing Surveys*. 1991. Т. 23, № 1. С. 5–48.
3. Денніс Дж. Б. Data Flow Supercomputers // *Computer*. 1980. Т. 13, № 11. С. 48–56.
4. *The Reactive Manifesto* [Електронний ресурс]. – Режим доступу: https://www.reactivemanifesto.org
5. *Akka Streams Documentation* [Електронний ресурс]. – Режим доступу: https://doc.akka.io/docs/akka-stream/
6. *ReactiveX Documentation* [Електронний ресурс]. – Режим доступу: https://reactivex.io/documentation
7. *Apache Flink: Stateful Stream Processing* [Електронний ресурс]. – Режим доступу: https://flink.apache.org
8. *IEEE Standard for Floating-Point Arithmetic (IEEE 754-2019)*. IEEE Computer Society, 2019. 66 с.
9. Гарріс Д. М., Гарріс С. Л. *Цифрова схемотехніка та архітектура комп’ютерів*. 2-е вид. Сан-Франциско: Morgan Kaufmann, 2012. 712 с.
10. Кнут Д. Е. *Мистецтво програмування. Т. 2: Напівчислові алгоритми*. 3-тє вид. Редінг: Addison-Wesley, 1997. 784 с.
11. *Google Cloud. Stream Data Processing Patterns and Architectural Guidance* [Електронний ресурс]. – Режим доступу: https://cloud.google.com/architecture/streaming-data
12. *Red Hat Developer. Event-driven Architecture vs. Streaming Architecture*, 2024 [Електронний ресурс]. – Режим доступу: https://developers.redhat.com/articles/streaming-vs-event-driven

13. Клеппманн М. *Проєктування подієво-орієнтованих систем*, 2023 [Електронний ресурс]. – Режим доступу: https://martin.kleppmann.com/papers/

ДОДАТОК 1

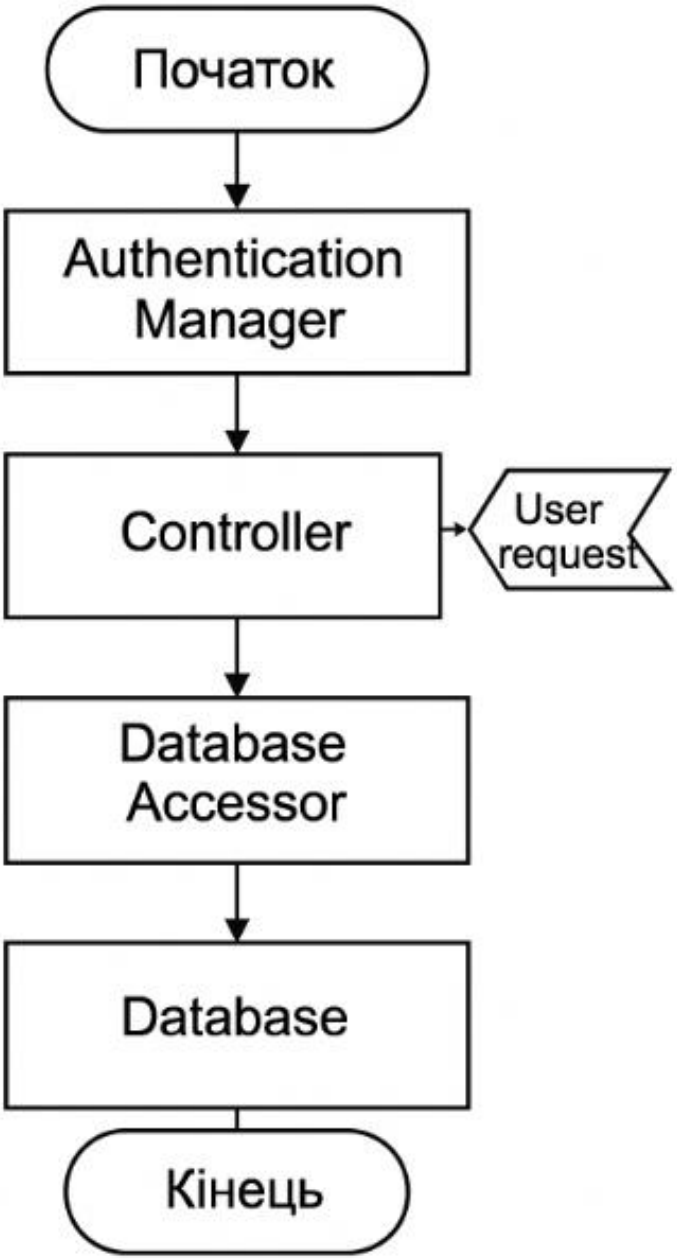
Програмна модель обчислень в потокових системах з безпосередніми зв’язками між модуляли

Блок-схема алгоритму

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.004 Д1	
		№ докум.	Підпис	Дата		
Розробила		Тулуб М. М.			Програмна модель обчислень в потокових системах з безпосередніми зв'язками між модулями Блок-схема алгоритму	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Жабін В. І.				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11
Н. Контр.		Міщенко Л. Д.				
Затвердив						

ДОДАТОК 2

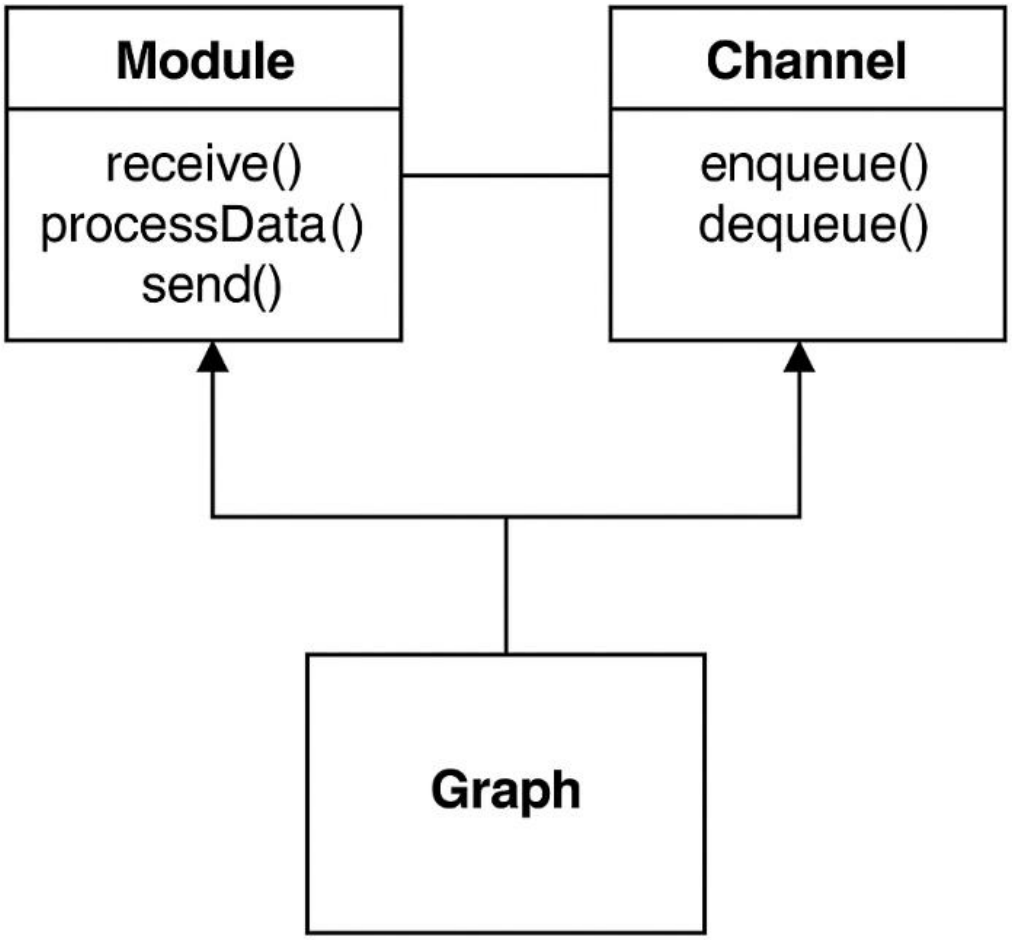
Програмна модель обчислень в потокових системах з безпосередніми зв’язками між модуляли

Функціональна схема

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.005 Д2		
		№ докум.	Підпис	Дата			
Розробила		Тулуб М. М.			Програмна модель обчислень в потокових системах з безпосередніми зв’язками між модулями Функціональна схема	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Жабін В. І.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11	
Н. Контр.		Міщенко Л. Д.					
Затвердив							

ДОДАТОК З

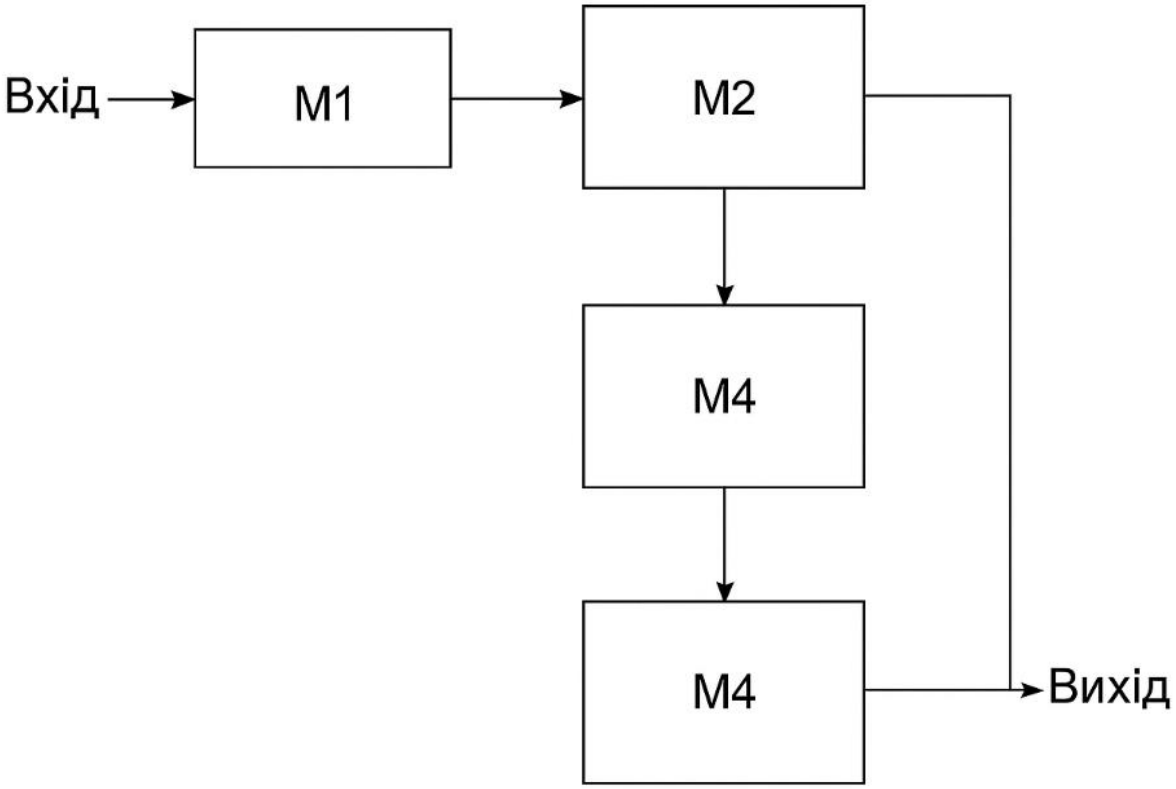
Програмна модель обчислень в потокових системах з безпосередніми зв’язками між модуляли

Структурна схема

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.006 Д3		
		№ докум.	Підпис	Дата			
Розробила		Тулуб М. М.			Програмна модель обчислень в потокових системах з безпосередніми зв’язками між модулями Структурна схема	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Жабін В. І.					
						КПІ ім. Ігоря Сікорського, ФІОТ, ІО-11	
Н. Контр.		Міщенко Л. Д.					
Затвердив							

ДОДАТОК 4

Програмна модель обчислень в потокових системах з безпосередніми зв’язками між модуляли

Текст програмного коду

Аркушів 1

Київ 2025 р

```python
def compute_expression(a: float, b: float, c: float, d: float) -> float:
    if d == 0:
        raise ZeroDivisionError("D must not be zero.")
    return (a * b + c) / d

# Приклад використання:
A = 2.5
B = 4.0
C = 3.2
D = 2.0
result = compute_expression(A, B, C, D)
print(f"Result: {result:.6f}")
```

```python
def horner(coefficients: list, x: float) -> float:
    result = coefficients[0]
    for coef in coefficients[1:]:
        result = result * x + coef
    return result

# Приклад: P(x) = 2x³ - 4x² + 3x + 5 при x = 1.5
coeffs = [2, -4, 3, 5]
x_value = 1.5
polynomial_result = horner(coeffs, x_value)
print(f"P({x_value}) = {polynomial_result:.6f}")
```

```python
def redundant_add(a_digits: list, b_digits: list) -> list:
    assert len(a_digits) == len(b_digits)
    result = []
    carry = 0
    for a, b in zip(reversed(a_digits), reversed(b_digits)):
        s = a + b + carry
        if s > 1:
            result.append(s - 3)  # +2 → -1 + carry
            carry = 1
        elif s < -1:
            result.append(s + 3)  # -2 → +1 - carry
            carry = -1
        else:
            result.append(s)
            carry = 0
    result.reverse()
    return result

# Приклад: +2 = [+1, 0, +1], +3 = [+1, +1, +1] → [+1, -1, 0] (прикладова
симуляція)
a = [1, 0, 1]
b = [1, 1, 1]
sum_digits = redundant_add(a, b)
print("Result in NCCЧ:", sum_digits)
```