Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

Метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA
Назва теми

Рівень вищої освіти *другий (магістерський)*

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Шифр КвРКІП 240107.24.01.06 ПЗ

Виконав здобувач II курсу, група КІ2м-24-1 — Підпис — Назарій ВЕЛИЧКО (Ініціали, прізвище)

Керівник доктор техн. наук, професор — Науковий ступінь, учене звання — Підпис — Сергій ЛИСЕНКО (Ініціали, прізвище)

Нормоконтролер д. техн. наук, професор — Науковий ступінь, учене звання — Підпис — Сергій ЛИСЕНКО (Ініціали, прізвище)

До захисту допускаю:
завідувач кафедри КІІС
«1» травня 2026 р.
дата

підпис — Ольга ПАВЛОВА (Ініціали, прізвище)

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Рівень вищої освіти ДРУГИЙ (МАГІСТЕРСЬКИЙ)

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ
Завідувачка кафедри КІІС
_____ Ольга ПАВЛОВА

" 12 " 01 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Величку Назарію Вікторовичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA
Керівник проекту (роботи) Лисенко Сергій Миколайович, д.т.н., проф.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 12.01.2026 р. № 6

2. Термін подання здобувачем роботи на кафедру 01.05.2026 р.

3. Вихідні дані до роботи Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Аналіз відомих методів планування та розподілу задач у гетерогенних обчислювальних системах базі FPGA

Побудова моделі системи

Метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA

Програмно-технічний засіб реалізації методу планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання « 12 » 01 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проєкту (роботи)	Термін виконання етапів проєкту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	12.01.2026	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	15.01.2026	виконано
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.02.2026	виконано
4	Робота над розділом 2 – вибір компонентів для проєктування системи адаптивного застосування моніторингових елементів розвідувального БПЛА	01.03.2026	виконано
5	Робота над розділом 3 – проєктування системи адаптивного застосування моніторингових елементів розвідувального БПЛА	29.03.2026	виконано
6	Оформлення пояснювальної записки згідно вимог	25.04.2026	виконано
7	Попередній захист ВКР	26.04.2025	виконано
8	Захист ВКР на засіданні ЕК	травень 2026 року	

Здобувач _____ Назарій ВЕЛИЧКО
Підпис Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Сергій ЛИСЕНКО
Підпис Ім'я, ПРІЗВИЩЕ

РЕФЕРАТ

Тема кваліфікаційної роботи магістра: Метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA

Автор роботи: Величко Назарій Вікторович.

Керівник роботи: Лисенко Сергій Миколайович.

Пояснювальна записка: 96 с., 13 рис., 2 табл., 3 дод., 112 джерел.

ПЛАНУВАННЯ, РОЗПОДІЛУ ЗАДАЧ, ГЕТЕРОГЕННІ ОБЧИСЛЮВАЛЬНІ СИСТЕМИ, FPGA, ЕНЕГРОЕФЕКТИВНІСТЬ.

Об’єктом дослідження є процес оптимізації продуктивності обчислювальних систем в частині планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA.

Предметом дослідження є метод та система планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA.

Метою кваліфікаційної роботи магістра є оптимізації продуктивності обчислювальних систем в частині зменшення енергоспоживання та підвищення швидкодії системи за рахунок динамічної надлишковості.

Для розв’язання поставлених задач використовувалися методи аналізу, синтезу, теорії забезпечення функціонування складних систем, методи математичного моделювання.

Наукова новизна отриманих результатів:

- удосконалено метод енергоефективного планування задач у гетерогенних паралельних системах, який на відміну від відомих поєднує евристичну оптимізацію та навчання з підкріпленням із урахуванням залежностей задач і неоднорідності ресурсів, і який дає змогу зменшити енергоспоживання та підвищити швидкодію системи за рахунок динамічної надлишковості;
- набула подальшого розвитку система планування задач у гетерогенних паралельних системах на основі FPGA.

Практична значимість отриманих результатів полягає у розробленні системи, яка здійснює оптимізацію планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA.

У першому розділі здійснено аналіз предметної області планування та розподілу задач у гетерогенних обчислювальних системах встановлено, що ефективність функціонування таких систем визначається не лише обчислювальною продуктивністю окремих компонентів, а й організацією пам’яті, характеристиками міжкомпонентного обміну даними, політиками диспетчеризації та здатністю системи адаптуватися до різних типів навантаження.

У другому розділі здійснено формалізацію обчислювального процесу та побудовано комплексну модель системи планування задач у гетерогенних обчислювальних середовищах.

У третьому розділі розроблено та детально обґрунтовано метод планування та розподілу задач у гетерогенних обчислювальних системах на базі програмованих логічних інтегральних схем, що поєднує математичну строгість оптимізаційного моделювання з гнучкістю адаптивного навчання.

У четвертому розділі розроблено та описано програмно-технічний засіб реалізації методу планування та розподілу задач у гетерогенних обчислювальних системах на базі програмованих логічних інтегральних схем, проведено його експериментальну перевірку та порівняльний аналіз ефективності.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

АПЗ - антивірусне програмне забезпечення

БД - база даних

БПР - блок прийняття рішень

ГА - генетичний алгоритм

ОС - операційна система

ПЗ - програмне забезпечення

СВВ - система виявлення вторгнень

ЕС - експертна система

DDoS - Distributed Denial of Service (розподілена відмова в обслуговуванні)

IDS - система виявлення вторгнень

ВСТУП

В умовах зростання обсягів даних, ускладнення алгоритмів обробки та посилення вимог до швидкодії й енергоефективності, обчислювальні системи дедалі частіше будуються як гетерогенні платформи, що поєднують процесори загального призначення та спеціалізовані прискорювачі [1-4]. Такий підхід дає змогу адаптувати виконання різних фрагментів задачі до різних моделей обчислень і до різних ресурсних обмежень, однак водночас ускладнює керування обчислювальним процесом [5-9]. Ключовою проблемою стає планування та розподіл задач між різнорідними виконавцями з урахуванням обмежень пропускної здатності введення-виведення, доступності пам’яті, особливостей залежностей між задачами, а також сервісних вимог щодо затримки та пропускної здатності [10-13].

Особливе місце серед прискорювачів займають програмовані логічні інтегральні схеми (FPGA), які завдяки реконфігурованій апаратурі дозволяють реалізовувати обчислювальні ядра, орієнтовані на потокову обробку, конвеєризацію та паралелізм на рівні даних. Водночас застосування FPGA у складі гетерогенних систем породжує низку специфічних факторів, що впливають на рішення планувальника. До них належать ресурсна місткість кристала (LUT, тригери, вбудована пам’ять, блоки арифметики), залежність характеристик виконання від конфігурації, час конфігурації або часткової реконфігурації, а також комунікаційні витрати під час переміщення даних між доменами пам’яті. У результаті класичні алгоритми планування, розроблені для однорідних процесорних систем або для прискорювачів із відносно стабільною моделлю виконання, не забезпечують достатньої адекватності для FPGA-орієнтованих гетерогенних середовищ, особливо за умов динамічних навантажень і багатокритеріальних вимог [14-17].

Актуальність теми зумовлена потребою у методах планування, які поєднують прийняття рішень щодо призначення задач на різні типи виконавців із явним урахуванням витрат на підготовку та передавання даних, обмежень внутрішніх ресурсів FPGA, а також можливих конфігураційних переходів [18-21]. Практичне

значення проблеми проявляється у системах, де одночасно присутні вимоги до високої пропускної здатності та низької затримки, а навантаження має змішану структуру: потокову, запитну або графову. У таких умовах метод планування має не лише забезпечити ефективне завантаження доступних ресурсів, але й мінімізувати непродуктивні витрати, що виникають через неузгодженість обчислень і комунікацій або через невдалу послідовність використання конфігурацій [22-25].

Для досягнення мети необхідно проаналізувати сучасні методи планування у гетерогенних системах та виділити обмеження їх застосування для FPGA-середовищ; сформувати формальну модель задачі планування, визначити критерії ефективності та обмеження; запропонувати алгоритмічний механізм прийняття рішень про розподіл задач, який враховує стан ресурсів, параметри задач і витрати передавання даних; розробити структуру програмно-апаратної підтримки методу та виконати експериментальне оцінювання його результативності на модельних і/або реальних сценаріях.

Метою кваліфікаційної роботи магістра є оптимізації продуктивності обчислювальних систем в частині зменшення енергоспоживання та підвищення швидкодії системи за рахунок динамічної надлишковості.

Поставлена мета досягається розв’язанням таких основних завдань:

- проаналізувати відомі методи оптимізації продуктивності обчислювальних систем;

- розробити метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA;

- здійснити дослідження методу планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA;

- реалізувати метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA у вигляді системи.

Об’єктом дослідження є процес оптимізації продуктивності обчислювальних систем в частині планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA.

Предметом дослідження є метод та система планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA.

Наукова новизна отриманих результатів:

- удосконалено метод енергоефективного планування задач у гетерогенних паралельних системах, який на відміну від відомих поєднує евристичну оптимізацію та навчання з підкріпленням із урахуванням залежностей задач і неоднорідності ресурсів, і який дає змогу зменшити енергоспоживання та підвищити швидкодію системи за рахунок динамічної надлишковості;
- набула подальшого розвитку система планування задач у гетерогенних паралельних системах на основі FPGA.

Практична значимість отриманих результатів полягає у розробленні системи, яка здійснює оптимізацію планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA.

Для розв’язання поставлених задач використовувалися методи аналізу, синтезу, теорії забезпечення функціонування складних систем, методи математичного моделювання.

За темою кваліфікаційної роботи опубліковано одну публікацію [112] у Збірнику наукових праць за матеріалами XIX Всеукраїнська науково-практична web конференція аспірантів, студентів та молодих вчених комп’ютерні інтелектуальні системи та мережі (25-27 березня 2026 р.)

1 АНАЛІЗ ВІДОМИХ МЕТОДІВ ПЛАНУВАННЯ ТА РОЗПОДІЛУ ЗАДАЧ У ГЕТЕРОГЕННИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ НА БАЗІ FPGA

1.1 Гетерогенні обчислювальні системи: поняття, архітектури, сфери застосування

Гетерогенна обчислювальна система розглядається як обчислювальне середовище, у якому одночасно присутні та взаємодіють кілька типів обчислювальних ресурсів, що відрізняються мікроархітектурою, моделлю паралелізму, організацією пам’яті та механізмами доступу до даних [26-29]. У такій системі виконання прикладних обчислень розподіляється між ресурсами, які можуть бути інтегрованими в межах одного кристала, представленими окремими пристроями в межах вузла або складати кластерну інфраструктуру. Поняття гетерогенності у контексті магістерського дослідження доцільно пов’язувати не лише з фізичною різнорідністю обчислювачів, а й із різними моделями виконання задач, різними режимами взаємодії з пам’яттю та різними способами керування виконанням через системне та прикладне програмне забезпечення [30-33].

Архітектурно гетерогенні системи можуть реалізовуватися як вузол із центральним процесором загального призначення та одним або кількома прискорювачами, як система-на-кристалі з інтегрованими ядрами різних типів, або як багатовузлова інфраструктура, у якій ресурси з різною природою доступні через мережеві інтерфейси та механізми оркестрації [34-37]. Для опису таких систем у науковій роботі важливо фіксувати, які саме ресурси беруть участь у виконанні обчислень, який механізм розмежування відповідальності між ними застосовано, як організовано обмін даними, а також який рівень абстракції використовується для постановки та керування задачами, оскільки ці аспекти безпосередньо визначають формалізацію задачі планування та розподілу [38-40].

Сфери застосування гетерогенних систем охоплюють задачі, де обчислювальне навантаження природно поділяється на компоненти з різними вимогами до моделі паралелізму, структури доступу до пам’яті та характеристик

виконання. У межах теми, пов’язаної з FPGA, контекст застосувань доцільно описувати через типи обробки даних і форму представлення задач: пакетну обробку незалежних завдань, виконання графів залежностей, потокову обробку та сценарії з часовими обмеженнями. У кожному випадку під “сферою застосування” у рамках даного розділу розуміється не конкретна галузь, а клас навантажень, для якого характерна участь кількох різнорідних обчислювальних ресурсів і наявність рішень про розміщення обчислень та переміщення даних [41].

Гетерогенні обчислення визначаються як спосіб організації виконання прикладних задач, за якого обчислювальний процес розподіляється між ресурсами різних типів, що відрізняються за принципами реалізації обчислень і доступу до пам’яті, а також за механізмами керування виконанням [42]. У такому підході прикладна задача розглядається як композиція обчислювальних компонентів, кожен з яких може бути співвіднесений із певним класом виконавчих пристроїв. В межах формалізації, яка використовується для планування, “роль” конкретного ресурсу означає його функцію в загальному конвеєрі обробки: ініціацію та оркестрацію виконання, виконання масово-паралельних фрагментів, реалізацію спеціалізованих апаратних ядер або інкапсуляцію обчислювальних та комунікаційних функцій в межах інтегрованої платформи [43].

Центральний процесор загального призначення (CPU) у гетерогенній системі трактується як керуючий та універсальний виконавець, який забезпечує виконання послідовних або слабо-паралельних ділянок, прийняття рішень щодо розподілу, підготовку даних, запуск прискорювачів та синхронізацію результатів. У контексті планування CPU виступає джерелом постановки задач, координатором черг і місцем виконання тих фрагментів, які не відображаються на прискорювачі або потребують операцій, тісно пов’язаних із системними викликами, керуванням пам’яттю та взаємодією з ОС. При моделюванні часу виконання CPU-частина часто розглядається як базовий опорний маршрут виконання, з яким порівнюється доцільність альтернативного виконання на інших ресурсах [44].

Графічний процесор (GPU) у межах гетерогенних обчислень визначається як пристрій, орієнтований на виконання великої кількості однотипних операцій над

масивами даних у паралельному режимі з використанням багатопоточних виконавчих блоків [45-48]. У постановці задачі планування роль GPU описується через здатність приймати обчислювальні ядра у вигляді кернелів, виконуваних над множиною елементів, та через специфіку керування даними між пам’яттю хоста і пам’яттю пристрою. Для формального опису важливо трактувати GPU як ресурс із власним контекстом виконання, який приймає задачі у вигляді запусків кернелів і потребує явного керування буферами та синхронізації [49-51].

Програмована логічна інтегральна схема (FPGA) у гетерогенних системах визначається як реконфігурований апаратний ресурс, здатний реалізувати обчислювальні функції шляхом синтезу спеціалізованої апаратної структури під конкретний алгоритм або клас алгоритмів. Роль FPGA у рамках теми магістерської роботи доцільно описувати через поняття апаратного ядра, яке реалізує задану функцію обробки, а також через керовані механізми конфігурації, включно з варіантами часткової реконфігурації, коли впродовж роботи змінюється підмножина логіки. Для задач планування FPGA розглядається як ресурс, для якого рішення про розміщення задач включає не лише вибір виконавця, а й вибір конфігураційного стану, а також узгодження з обмеженнями на апаратні ресурси та пропускні характеристики підсистеми введення-виведення. [52-55]

Спеціалізована інтегральна схема (ASIC) в гетерогенних обчисленнях визначається як апаратний модуль фіксованої функціональності, що реалізує наперед заданий набір операцій або алгоритмів. У термінах ролей ASIC виступає як детермінований виконавець вузького класу задач, до якого звертаються через стандартизований інтерфейс керування та передачі даних. У моделі планування ASIC-ресурс описується через доступність, обмежену номенклатуру підтримуваних операцій та параметри обробки, що визначаються конструктивно й не змінюються в часі, а отже рішення про розподіл для нього зводиться до вибору допустимого відображення задачі на доступний апаратний модуль і планування доступу до нього [56-58].

Система-на-кристалі (SoC) у гетерогенних обчисленнях визначається як інтегрована платформа, в якій в межах одного кристала співіснують різні

обчислювальні ядра та спеціалізовані модулі, поєднані спільною інтерконектною мережею та підсистемою пам’яті [59-61]. Роль SoC у рамках теми полягає у тому, що гетерогенність реалізується не як зовнішнє підключення прискорювачів, а як внутрішня кооперація компонентів із визначеними шляхами доступу до пам’яті та апаратними механізмами взаємодії. У контексті планування SoC доцільно описувати як середовище, де рішення про розподіл задач додатково залежить від топології міжмодульних з’єднань, спільного доступу до пам’яті, можливостей паралельного виконання в межах одного домену керування та політик арбітражу ресурсів.

1.1.1 Класи гетерогенних платформ

Класифікація гетерогенних платформ у контексті планування та розподілу задач доцільна як спосіб описати середовище виконання, оскільки тип платформи визначає характер доступних ресурсів, допустимі режими взаємодії між ними, організацію пам’яті та канали передачі даних, а також типові режими навантаження. У межах даної роботи під “класом платформи” розуміється не бренд чи конкретна реалізація, а архітектурно-експлуатаційний профіль системи, що задає рамкові припущення для моделювання часу виконання, формалізації обмежень і вибору політики планування [62].

Вбудовані гетерогенні платформи, які часто позначаються як edge-системи, характеризуються інтеграцією різнорідних обчислювальних компонентів у межах одного пристрою, що функціонує поблизу джерела даних або безпосередньо в контурі керування [63]. Типовою є конфігурація SoC, де поєднуються ядра загального призначення та апаратні прискорювальні блоки, включно з реконфігурованою логікою [64]. Для планування в таких системах суттєвими стають часові обмеження на реакцію, детермінізм виконання та обмеженість локальних ресурсів пам’яті та пропускної здатності інтерконекту. Розподіл задач у цьому класі платформ часто прив’язаний до топології внутрішнього інтерконекту та до режимів спільного доступу компонентів до пам’яті, оскільки значна частина

комунікації відбувається всередині кристала або плати й задається апаратною архітектурою [65].

Серверні гетерогенні платформи розглядаються як вузли дата-центрового класу, де центральні процесори доповнюються дискретними прискорювачами, підключеними через високошвидкісні шини, і працюють під керуванням повнофункціональної операційної системи та систем керування ресурсами. У цьому класі платформи гетерогенність зазвичай проявляється як співіснування CPU з GPU та/або FPGA, де прискорювачі мають власні контексти виконання і окремі домени пам’яті [66]. Для задач планування ключовими стають механізми постановки робіт у черги прискорювачів, моделювання передачі даних між доменами пам’яті, а також координація багатьох паралельних потоків запитів, що надходять від сервісів чи пакетних обчислень. Серверний контекст також передбачає мультиорендність або конкуренцію за ресурси між кількома робочими навантаженнями, що впливає на допустимі припущення щодо стабільності продуктивності [67].

Платформи класу HPC трактуються як високопродуктивні обчислювальні системи, побудовані з множини вузлів, об’єднаних низьколатентною високошвидкісною мережею, де в кожному вузлі можуть бути присутні різнорідні прискорювачі. У цьому класі гетерогенність проявляється одночасно на рівні вузла та на рівні кластера, а планування задач включає не лише вибір типу виконавця в межах вузла, але й розміщення робіт по вузлах із урахуванням мережевої взаємодії. Формалізація планування в HPC зазвичай враховує структуру паралельних програм, зокрема залежності між підзадачами та комунікаційні фази, оскільки міжвузловий обмін може бути суттєвою складовою загального часу виконання. Для FPGA-орієнтованих сценаріїв у HPC додатково важливо фіксувати, що доступ до прискорювачів може бути організований через політики виділення ресурсів, і це впливає на модель доступності пристроїв у часі [68].

Хмарні гетерогенні платформи з прискорювачами описуються як інфраструктури, у яких різнорідні ресурси надаються як сервіс із механізмами віртуалізації, ізоляції та оркестрації. У межах такого класу платформи

прискорювачі можуть бути доступні через абстракції на рівні віртуальних машин, контейнерів або керованих сервісів, а планування задач реалізується як комбінація локального диспетчеризаційного механізму та глобального оркестратора, що керує розміщенням робіт і виділенням ресурсів [69-71]. Для формальної постановки задачі планування в хмарі суттєвими стають поняття пулу ресурсів, черг запитів, політик розміщення та умов доступу до прискорювачів, а також можливість змінюваності доступної продуктивності через спільне використання інфраструктури. У випадку FPGA в хмарному середовищі важливим елементом моделі є представлення конфігураційних артефактів як керованих об’єктів, що визначають готовність ресурсу до виконання конкретного класу задач [72-75].

1.1.2 Типові сценарії використання FPGA-прискорення

FPGA-прискорення в гетерогенних системах зазвичай розглядають у контексті задач, де обчислення природно формуються як послідовність операцій над потоком даних або як повторювані ядра з фіксованою структурою обробки. У таких сценаріях виконання прикладної функції подається як апаратне ядро, яке приймає вхідні дані через інтерфейси передавання, виконує перетворення за заданою схемою та формує вихідний потік або блок результатів. У межах дослідження з планування та розподілу задач принципово важливо фіксувати, що для FPGA-орієнтованих сценаріїв задача часто включає не лише обчислювальну складову, а й вимоги до режиму подачі даних, до буферизації, до синхронізації з іншими компонентами системи та до керування конфігураційним станом прискорювача [76-78].

Потокова обробка даних є характерним випадком, коли вхідні дані надходять неперервно або квазинеперервно, а обробка має бути організована як конвеєр із фіксованими стадіями. У таких системах задачі часто формалізуються як ланцюжки операторів, де вихід однієї стадії є входом наступної, а ключовим аспектом стає узгодження швидкостей обробки між стадіями та режимів буферизації. Для планування в гетерогенному середовищі це означає необхідність

опису не тільки “часу виконання” окремого фрагмента, а й пропускної здатності обробки та умов стабільної роботи без переповнення буферів і без втрати даних [79-81].

Сценарії виконання висновку машинного навчання у гетерогенних системах з FPGA зазвичай описуються як виконання послідовності шарів моделі або як обробка фіксованих операторів над тензорами, де значущими стають структура графа обчислень, розміри проміжних активацій та схеми доступу до пам’яті. У межах задачі планування важливо врахувати, що inference може виконуватися пакетно або у режимі обробки запитів, а модель може бути подана як набір підзадач із залежностями. У FPGA-контексті такі підзадачі можуть відображатися на апаратні ядра, що реалізують окремі оператори або групи операторів, і тоді розподіл задачі перетворюється на задачу призначення фрагментів графа обчислень різним виконавцям із урахуванням обсягів переданих тензорів та послідовності виконання.

Цифрова обробка сигналів (DSP) часто подається як набір алгоритмів фільтрації, перетворень та кореляційних обчислень, де типово присутні регулярні структури, повторювані операції та передбачувані шаблони доступу до даних. У гетерогенній системі DSP-навантаження може бути представлено як цикл обробки кадрів або блоків сигналу, а також як потік семплів із фіксованими часовими параметрами. З точки зору планування це означає, що опис задачі має включати параметри розміру блоків, частоту надходження даних, допустимі межі затримок і механізми синхронізації з іншими компонентами, які генерують або споживають сигнал.

Криптографічні задачі у гетерогенних системах з FPGA зазвичай формалізуються як виконання визначених криптографічних примітивів над потоками або блоками даних, де важливою є коректність режимів обробки, порядку операцій і узгодження ключового матеріалу з сесіями обробки. У термінах планування такі задачі часто мають чітко визначені межі “одиниці роботи” (пакет, блок, сесія), а також вимоги щодо ізоляції контекстів та керування ключами. Для моделі планування це означає необхідність опису того, як контекст криптообробки

передається між CPU та FPGA, як організовується черга запитів, і які існують правила конкурентного доступу до криптографічного ядра. Мережеві функції в гетерогенних системах із FPGA описуються як обробка пакетів або потоків трафіку з реалізацією функцій комутації, фільтрації, інспекції, маршрутизації, балансування або інкапсуляції. У таких сценаріях задача часто представлена як набір операцій над заголовками та корисним навантаженням пакетів із вимогами до порядку обробки, до збереження стану потоків та до дотримання політик. У плануванні важливими стають моделі черг, механізми пріоритезації, правила віднесення потоків до контекстів обробки та узгодження з системними механізмами QoS, оскільки мережеві навантаження є типовим прикладом конкурентного доступу багатьох потоків запитів до одного прискорювального ресурсу.

Узагальнюючи, перелічені сценарії зручні для подальшої постановки задачі планування тим, що вони дозволяють описати типові форми задач: потокові конвеєри, графи залежностей, циклічну блокову обробку та пакетно-запитні режими. Саме форма задачі визначає, яким чином у наступних підрозділах буде побудовано модель вартості виконання, модель передачі даних та правила диспетчеризації на FPGA в рамках гетерогенної системи.

1.1.3 Критерії ефективності гетерогенних систем: продуктивність, енергоефективність, затримка, вартість, масштабованість, надійність

Оцінювання ефективності гетерогенних систем у контексті планування та розподілу задач спирається на набір критеріїв, які мають бути формалізовані як метрики або як цільові функції, що оптимізуються. У практичних постановках ці критерії часто перебувають у відношенні компромісу, тому для наукової роботи важливо описати їх у термінах вимірюваних величин, а також визначити, на якому рівні системи вони оцінюються: для окремої задачі, для потоку задач, для вузла чи для всієї інфраструктури [82].

Продуктивність у гетерогенній системі доцільно трактувати як характеристику швидкості виконання обчислень або обробки запитів за заданого режиму навантаження. Вона може бути подана як час завершення задачі, як пропускна здатність системи для потоку даних або як кількість завершених робіт за одиницю часу. У контексті планування продуктивність прямо пов’язана з тим, як задачі призначаються виконавцям, як формується черга на прискорювачах і як враховуються затримки на передачу даних та синхронізацію [83].

Енергоефективність використовується як критерій, що описує співвідношення між обсягом виконаної роботи та витратою енергії під час виконання. Для коректної формалізації необхідно фіксувати, що енергетична оцінка може виконуватися на рівні пристрою, на рівні вузла або як інтегральна оцінка для системи, а також що модель енерговитрат включає як обчислювальну фазу, так і фази передачі даних та очікування. У задачах планування це приводить до необхідності розглядати енергію як функцію маршруту виконання задачі та як параметр, який залежить від стану ресурсів і режиму завантаження [84].

Затримка як критерій у гетерогенних системах описує інтервал між надходженням задачі (або елементів потоку) і отриманням результату. У потокових або запитних сценаріях затримка розглядається разом із варіативністю затримки, оскільки послідовність рішень планувальника може призводити до різних чергових та комунікаційних ефектів. Для формальної постановки задачі планування затримка є сумою часу очікування в чергах, часу конфігураційних переходів (за наявності), часу передачі даних, часу обчислень та часу синхронізації між компонентами, і тому потребує явного представлення в моделі вартості [85-88].

Вартість у гетерогенних системах може трактуватися як сукупність витрат, пов’язаних із використанням обчислювальної інфраструктури для виконання заданого обсягу робіт. У хмарних та сервісних контекстах це часто подається як облік ресурсо-часу, зайнятості прискорювачів або інших квотних показників, тоді як для локальних систем вартість може бути представлена через нормативи використання ресурсів або через обмеження на доступність певних компонентів [89-91]. У задачах планування цей критерій фактично задає обмеження на

допустимі рішення з точки зору зайнятості прискорювачів і режимів їх використання в часі [92].

Масштабованість як критерій у рамках дослідження планування означає здатність системи та її механізмів керування підтримувати збільшення обсягу задач, кількості потоків запитів або кількості доступних ресурсів без зміни базових принципів роботи та без втрати керованості. Для формалізації в магістерській роботі масштабованість доцільно пов’язувати з моделлю розширення платформи: збільшенням кількості вузлів, збільшенням кількості прискорювачів на вузол, введенням нових типів ресурсів або зміною топології комунікацій. У цьому контексті критерій масштабованості відноситься і до алгоритму планування, оскільки він визначає, як зростає складність прийняття рішень і як змінюється якість планів при розширенні системи.

Надійність у гетерогенних системах трактується як здатність забезпечувати коректне виконання задач і стабільність сервісу за наявності відмов компонентів, деградації продуктивності або некоректних станів, що виникають в процесі експлуатації. У контексті планування надійність пов’язана з тим, як система обробляє недоступність ресурсів, як відбувається повторне призначення задач, як підтримується консистентність стану при часткових збоях та як узгоджується завершення обчислень у розподіленому середовищі. Для магістерської постановки задачі достатньо зафіксувати, що надійність проявляється як вимога до механізму планування враховувати доступність ресурсів у часі та можливість відновлення або міграції виконання [93].

Інтеграція різнорідних компонентів у межах гетерогенної системи створює набір системних проблем, які безпосередньо впливають на формалізацію та якість планування. Ці проблеми виникають на межі між різними моделями виконання, різними доменами пам’яті та різними механізмами керування роботою пристроїв, тому в межах першого розділу їх доцільно описувати як джерела додаткових обмежень і як фактори, що вводять невизначеність у моделі оцінювання часу виконання.

Узгодження моделей виконання стосується того, що CPU, GPU, FPGA та інші прискорювачі працюють за різними принципами диспетчеризації, мають різні одиниці планування та використовують різні механізми запуску. На рівні програмної моделі це проявляється як необхідність перетворення прикладної задачі у форму, придатну для конкретного виконавця, а на рівні планування як необхідність узгоджувати порядок запусків, семантику асинхронних операцій і точки синхронізації. Для FPGA додатковим аспектом є наявність конфігураційного стану, який фактично є частиною моделі виконання і визначає, які обчислювальні ядра доступні в конкретний момент. У підсумку, планувальник має працювати з неоднорідними поданнями роботи і зі змішаними механізмами завершення, що ускладнює побудову єдиної часової моделі [94].

Проблема балансування навантаження в гетерогенних системах пов’язана з тим, що різні компоненти мають різні профілі обробки задач, а також що реальна завантаженість залежить від поточних черг, контенції за пам’ять та каналами введення-виведення і від політик доступу. Балансування в цьому контексті означає узгодження призначення задач так, щоб уникати ситуацій, коли одні ресурси формують довгі черги, а інші простоюють, або коли розподіл задач створює непропорційне навантаження на спільні компоненти, зокрема на пам’ять чи інтерконект. Для FPGA-орієнтованих систем балансування ускладнюється тим, що доступність виконання певних задач може залежати від конфігурації, а отже ефективний баланс визначається не тільки кількістю задач, але й їхньою “конфігураційною сумісністю” у часі.

Вузькі місця введення-виведення (I/O bottlenecks) формуються через необхідність переміщення даних між доменами пам’яті та між пристроями через шини та інтерконекти, що мають обмежену пропускну здатність і обслуговують кілька потоків запитів. У гетерогенних системах значна частина часу може витрачатися на передачу буферів і на синхронізацію завершення цих передач, а також на конкуренцію за пропускну здатність при одночасних запусках задач. Для постановки задачі планування це означає, що модель часу виконання має включати складові передачі даних та можливе накладання передачі на обчислення, а також

має враховувати контенцію, коли кілька задач одночасно використовують спільний канал. Для FPGA в дискретному виконанні важливо також враховувати режим доступу до пам’яті прискорювача та реальну ефективну пропускну здатність, яка може залежати від шаблонів доступу [95].

QoS у гетерогенних системах трактують як сукупність вимог до якості обслуговування задач або потоків запитів, що визначають допустимі межі затримки, варіативності, пропускної здатності або пріоритетності. У багатокористувацьких або багатосервісних сценаріях QoS визначає правила співіснування різних класів робіт та механізми пріоритезації, а на рівні планування це перетворюється на необхідність враховувати пріоритети, резервування ресурсів, обмеження на максимальний час очікування та політики розподілу пропускної здатності. Для FPGA-орієнтованих систем QoS також пов’язаний із керуванням чергою на прискорювачі та з правилами доступу до конфігурацій, оскільки зміна конфігурації може впливати на часові гарантії для різних потоків задач. У цьому контексті планувальник має працювати не лише з метою оптимізації агрегованих метрик, а й з вимогами до ізоляції та пріоритетного обслуговування в межах спільної інфраструктури.

Описані проблеми інтеграції задають основу для подальшої формалізації в розділі про постановку задачі планування: вони визначають, які додаткові складові повинні бути включені в модель вартості виконання, які обмеження на доступність ресурсів і пропускну здатність необхідно врахувати, а також які параметри мають бути керованими змінними у запропонованому методі розподілу задач у гетерогенних системах на базі FPGA [96].

1.2 FPGA як обчислювальний ресурс у гетерогенних системах

У гетерогенних обчислювальних системах FPGA доцільно розглядати як ресурс, що реалізує виконання задач не через послідовність інструкцій, а через апаратну конфігурацію, яка задає структуру обробки даних. Для задач планування та розподілу це означає, що “стан” виконавця визначається не лише поточним

завантаженням і чергою, а й активною конфігурацією, набором розгорнутих апаратних ядер, режимом доступу до пам’яті та конфігураційними артефактами, що можуть змінюватися в часі. У цьому підрозділі FPGA трактується як елемент обчислювального ансамблю, який має власну модель виконання, власні обмеження ресурсів і специфічні складові часу виконання, що мають бути враховані в методі планування [97].

Реконфігурованість FPGA полягає у можливості змінювати функціональність апаратної реалізації шляхом завантаження конфігурації, яка визначає зв’язки та логіку всередині кристала. У термінах виконання задач конфігурація задає, які апаратні ядра існують на даному етапі, які інтерфейси вони експонують, які буфери та маршрути передачі даних використовуються і яким є розклад використання внутрішніх ресурсів. Для задач планування реконфігурованість вводить додатковий вимір у прийняття рішень: задача може бути призначена на FPGA лише за умови сумісності з активною конфігурацією або після виконання конфігураційного переходу, який сам по собі є керованою операцією з власними часовими та ресурсними наслідками.

Конвеєризація в FPGA-орієнтованих реалізаціях описує організацію обробки як послідовності стадій, що працюють одночасно над різними елементами потоку. У такому поданні обчислення характеризується не тільки загальною тривалістю обробки одного елемента, а й режимом сталого виробництва результатів після заповнення конвеєра. Для планування це важливо, оскільки вартість виконання задачі на FPGA часто не зводиться до простого “часу на операцію”, а залежить від того, чи обробка виконується як потік, яка довжина потоку, як організовано ініціацію та завершення, і чи присутні бар’єри синхронізації між стадіями конвеєра та іншими компонентами системи [98].

Паралелізм на рівні даних у FPGA описується як можливість одночасного виконання багатьох однотипних операцій над різними даними через дублювання обчислювальних блоків або через організацію широких обчислювальних трактів. У моделі планування цей аспект проявляється через те, що продуктивність реалізації визначається параметрами структури ядра, зокрема шириною обробки, кількістю

паралельних каналів та співвідношенням між обчислювальними ресурсами і пам’яттю. Паралелізм на рівні потоків пов’язаний із можливістю одночасно підтримувати кілька незалежних потоків даних або кілька контекстів обробки на одному пристрої, коли в конфігурації розміщено кілька ядер або коли одне ядро має багатоканальний інтерфейс. Для планування це означає, що FPGA може виступати як ресурс із внутрішньою багатозадачністю, де обмеження визначаються не кількістю “ядер” у класичному розумінні, а наявними LUT/BRAM/DSP, пропускною здатністю інтерконектів і доступністю портів пам’яті []99.

Модель використання FPGA визначає, яким чином прискорювач включений у контур виконання задачі та як організовано керування. У межах теми планування важливо фіксувати, що одна й та сама FPGA може виступати різним типом ресурсу залежно від способу підключення та від того, чи перебуває вона “поза” процесором як окремий пристрій, чи інтегрована в спільний домен пам’яті, чи працює “в лінії” мережевого або потокового тракту.

У випадку використання FPGA як копроцесора в моделі accelerator-offload прикладна програма на CPU формує запити на виконання певних фрагментів, передає вхідні дані до домену пам’яті, доступного FPGA, ініціює виконання апаратного ядра та очікує завершення або асинхронно обробляє інші роботи. З погляду планування це означає наявність чіткої межі між хостом і прискорювачем, явного маршруту передачі даних та необхідності синхронізувати результати. Рішення про призначення задачі на FPGA у цій моделі завжди супроводжується оцінкою вартості передачі та вартості керування запуском [100].

У випадку FPGA як частини SoC (CPU+FPGA на одному кристалі) модель виконання змінюється через наявність внутрішнього інтерконекту та потенційно спільного адресного простору або спільної фізичної пам’яті з апаратними механізмами когерентності чи арбітражу. Тут постановка задачі на FPGA може бути реалізована як взаємодія з апаратними блоками через memory-mapped інтерфейси або через внутрішні DMA-механізми, а передача даних має інші характеристики, ніж у дискретному підключенні. Для планування ця модель означає, що час передачі і синхронізації часто залежить від конкуренції за

внутрішній інтерконект і від політик доступу до спільної пам’яті, а також що можливе більш тісне переплетення CPU- та FPGA-фрагментів одного алгоритму.

У випадку використання FPGA як мережевого або потокового прискорювача (inline) FPGA розташовується в тракті проходження даних, наприклад між мережевим інтерфейсом і пам’яттю хоста або між джерелом потоку і споживачем. У такій моделі задача часто існує не як дискретний виклик, а як постійно активна функція обробки трафіку або потоку даних, де рішення планувальника проявляється як налаштування правил маршрутизації, виділення черг, пріоритетів або як перемикання оброблювальних контекстів. Для формалізації це означає, що “одиниця роботи” може бути пакетом, кадром, елементом потоку або інтервалом часу, а критерії якості обслуговування і політики керування чергами стають частиною постановки задачі [101].

Обмеження FPGA, які необхідно враховувати в задачі планування, задають простір допустимих конфігурацій і впливають на те, які набори задач можуть виконуватися одночасно або в якій послідовності. Ресурсні обмеження на LUT/FF/BRAM/DSP задають місткість кристала щодо розміщення логіки, регістрів, вбудованої пам’яті та блоків арифметики. Для планувальника це означає, що призначення задачі на FPGA може вимагати перевірки можливості розміщення відповідного ядра або набору ядер у межах доступних ресурсів, а при одночасному виконанні кількох задач виникає задача розподілу внутрішніх ресурсів між ядрами [102-105].

Комунікаційні обмеження визначаються пропускною здатністю пам’яті та шин, через які дані надходять до FPGA та повертаються назад. У дискретних системах це може бути шина підключення, у SoC є внутрішній інтерконект, а в inline-моделях є мережеві або потокові інтерфейси. Для планування важливим є те, що ефективна пропускна здатність залежить від одночасного доступу кількох задач до одного каналу, від шаблону доступу до пам’яті та від можливості перекривати передачі з обчисленнями. У результаті в моделі вартості виконання задачі на FPGA необхідно представляти не лише “розрахунковий” час обчислення, а й час доставки даних із врахуванням контенції [106-108].

Час конфігурації або часткової реконфігурації є окремою складовою, яка впливає на порядок виконання задач. Конфігураційні переходи можуть бути необхідними, якщо задачі потребують різних апаратних ядер або різних параметризацій ядер, і тоді планування має врахувати, що між двома задачами може вставлятися керована операція зміни конфігурації. Для формалізації це означає необхідність введення станів FPGA та вартості переходів між станами, а також можливість групування задач за конфігураційною сумісністю, коли послідовність призначень мінімізує кількість переходів [109].

Специфіка HLS/RTL реалізацій проявляється в тому, що один і той самий алгоритм може бути реалізований у вигляді різних апаратних структур із різними параметрами частоти, латентності та пропускної здатності, які залежать від рішень синтезу, від застосованих директив, від організації пам’яті та від цільової частоти. Для задач планування це означає, що модель продуктивності FPGA-ядра є параметризованою і може залежати від профілю реалізації, а отже планувальник повинен спиратися або на профілювальні дані, або на аналітичні моделі, що відображають зв’язок між параметрами реалізації і характеристиками виконання.

Переміщуваність задач на FPGA у межах даної роботи доцільно визначити як властивість задачі бути відображеною на реконфігурований апаратний ресурс із прийнятною формою представлення обчислень, із допустимими вимогами до даних та з можливістю інтеграції в загальний контур виконання гетерогенної системи. Це поняття включає кілька взаємопов’язаних аспектів: можливість подання алгоритму у вигляді апаратного ядра, наявність визначених інтерфейсів введення-виведення, відповідність обмеженням внутрішніх ресурсів FPGA, а також допустимість накладних витрат, пов’язаних із підготовкою, передачею даних і, за потреби, конфігураційними переходами [110].

У задачах планування переміщуваність виконує роль фільтра, який визначає множину задач, що можуть розглядатися як кандидати на виконання на FPGA, та умови, за яких таке призначення є коректним з точки зору моделі виконання. Формально переміщуваність можна трактувати як відношення між задачею та ресурсом, що залежить від типу даних і їхніх обсягів, від структури залежностей у

задачі, від можливості потокової або блокової організації обробки, від вимог до стану та від доступності відповідної конфігурації. У подальших підрозділах ця концепція дозволяє обґрунтувати введення правил класифікації задач, параметрів моделі вартості та критеріїв прийняття рішення про розподіл, які враховують специфіку FPGA як реконфігурованого виконавця [111].

1.3 Дослідження методів планування

У роботі [1] подано систематизований огляд механізмів планування у безсерверних обчисленнях у різних середовищах (хмарному, крайовому та туманному). Автори формують класифікацію підходів до складання розкладів і наголошують, що специфіка безсерверної моделі (динамічні навантаження, різнорідність ресурсів, мінливість інтенсивності запитів) безпосередньо ускладнює постановку задачі планування та зумовлює широкий спектр алгоритмічних рішень.

У дослідженні [2] представлено огляд моделей і механізмів обслуговування інференсу моделей машинного навчання у безсерверному середовищі. Робота зосереджується на тому, як організовується виконання запитів до моделей як сервісу у безсерверних платформах, і узагальнює класи механізмів оптимізації, які автори групують за призначенням, зокрема орієнтовані на дотримання цільових показників якості сервісу, зменшення затримки, використання апаратних прискорювачів та врахування особливостей програмних платформ. Це є важливим підґрунтям для вибору політик планування в реальному часі з урахуванням сервісних обмежень.

Автори дослідження [3] запропонували метод планування для масштабних різнорідних задач із великими обсягами даних на основі паралельного алгоритму GATS-TS (за матеріалами CISCE 2022). Підхід подано як придатний для ситуацій, у яких стратегія планування повинна враховувати паралельну обробку та витрати, пов’язані з опрацюванням і передаванням значних обсягів даних між складовими системи.

У роботі [4] подано підхід до адаптивного планування ресурсів у складних середовищах на основі навчання з підкріпленням (метод Q-навчання). Автори формалізують проблему як таку, де фіксовані політики (наприклад, циклічний розподіл або пріоритетна дисципліна) не забезпечують належної адаптації до змін стану системи; натомість Q-навчання навчається на спостережуваній динаміці середовища та формує рішення щодо диспетчеризації й розподілу навантаження, а ефективність демонструється експериментально за показниками часу виконання та рівня завантаження ресурсів.

У дослідженні [5] розглянуто задачу перенесення обчислень у крайових обчисленнях мобільних мереж для ультращільних мереж, сформульовану як взаємодія багатьох користувачів із багатьма серверами. Автори розділяють постановку на етапи вибору сервера та ухвалення рішення щодо перенесення обчислень, застосовуючи генетичний алгоритм із двійковим кодуванням для наближеного розв'язання підзадач і організації розподіленої стратегії перенесення.

У роботі [6] подано схему динамічного планування запитів у крайових обчисленнях для застосунків Інтернету речей в умовах ультращільної крайової інфраструктури, де запити можуть передаватися на обчислення до базових станцій різних рівнів покриття. Автори підкреслюють мінливість попиту та мобільність користувачів і розглядають спільну оптимізацію, включно з компонентом керування висхідним передаванням даних, як частину практичного отримання рішень для планування в такій мережі.

У дослідженні [7] запропоновано динамічне планування для стохастичних середовищ "край–хмара", яке поєднує метод навчання з підкріпленням типу актор–критик для ухвалення рішень у реальному часі та рекурентну нейронну модель для врахування великого набору параметрів вузлів і задач та часових закономірностей навантаження. Підхід подається як придатний для ієрархічних мереж і різнорідних ресурсів, де важливими є адаптація та узгодження рішень із вимогами до якості сервісу.

Автори дослідження [8] аналізують планування залежних робочих процесів у мікросервісних застосунках у крайовому середовищі за наявності обмежень за

термінами завершення та з метою зменшення вартості виконання. Задача трактується як планувальна проблема для мікросервісних робочих процесів, де ключовими є узгодження залежностей, дотримання термінів та мінімізація витрат у мобільному крайовому середовищі.

У роботі [9] подано підхід до планування навантаження у крайових обчисленнях для щільних мереж п’ятого покоління, де алгоритмічний каркас пов’язується з моделлю балансування і розподілу навантаження на основі апарату теорії ігор та аналізом системної “вартості” у щільних мережевих умовах. Такий підхід є показовим як приклад формалізації рішень планування через модель взаємодії учасників і ресурсів у щільній топології.

У дослідженні [10] розглянуто алгоритми керування розгортанням мікросервісів і механізми забезпечення відмовостійкості для контейнерного розміщення компонентів у контексті технічного проміжного програмного забезпечення. Робота належить до напряму, де планування розміщення задач поєднується з керуванням взаємодією компонентів і процедурами підтримки працездатності на рівні контейнерної інфраструктури, що може бути релевантним для системного рівня планувальника.

У роботі [11] представлено планування відмовостійких робочих процесів у крайовому середовищі на основі глибокого навчання з підкріпленням. Публікація розглядає глибоке навчання з підкріпленням як засіб побудови політики, що враховує динаміку станів крайової інфраструктури та потребу у відновленні виконання робочих процесів у разі збоїв, тобто відмовостійкість інтегрується безпосередньо в механізм формування розкладу.

У дослідженні [12] запропоновано омбінований відмовостійкий підхід до планування незалежних задач із обмеженням за терміном завершення у хмарних системах, де автори поєднують повторний запуск та дублювання як дві базові стратегії забезпечення відмовостійкості й інтегрують їх в єдиний алгоритм планування. Така постановка є релевантною, коли планувальник має вирішувати не лише питання призначення задачі в часі та на ресурсі, а й вибір форми резервування для гарантування завершення до встановленого терміну.

У роботі [13] запропоновано оптимізацію відмовостійкого планування для хмарних робочих процесів на основі багатокритеріальної оптимізації. Підхід представляє напрям, у якому відмовостійкість і якість розкладу формалізуються як багатокритеріальна задача, наприклад із одночасним урахуванням тривалості виконання, вартості, показників надійності та використання ресурсів, із подальшим пошуком компромісних рішень.

У дослідженні [14] розроблено відмовостійкий евристичний алгоритм планування задач для хмарних середовищ із акцентом на ефективне використання ресурсів, де підвищення надійності виконання реалізується через механізми на кшталт дублювання та споріднені стратегії резервування під час складання розкладу. Робота є типовим представником напряму евристичного планування з інтегрованими механізмами підвищення надійності виконання.

1.4 Застосування FPGA як засобу побудови планування та розподілу задач у гетерогенних обчислювальних системах

Розгляд FPGA у контексті планування та розподілу задач у гетерогенних обчислювальних системах доцільно вести не лише як про виконавчий ресурс, на який можуть призначатися окремі фрагменти обчислень, а і як про технічну основу для реалізації самого механізму планування. У такій постановці FPGA виступає компонентом, що забезпечує апаратну підтримку прийняття рішень щодо призначення задач, диспетчеризації потоків робіт, керування чергами запитів і синхронізації доступу до прискорювачів та каналів передавання даних. Це змінює інтерпретацію планувальника: він не обов’язково є виключно програмною службою на CPU, а може бути реалізований як апаратно-програмний модуль, у якому частина функцій управління переноситься на FPGA.

У гетерогенній системі механізм планування зазвичай здійснює три взаємопов’язані дії: визначає, який виконавець має обслуговувати задачу, встановлює порядок обслуговування задач у чергах та ініціює виконання з відповідною підготовкою даних і синхронізацією. Застосування FPGA як засобу

побудови такого механізму означає, що логіка цих дій подається як апаратний тракт обробки подій, де надходження задачі, оновлення станів ресурсів і формування рішення є операціями над структурованими даними, які можуть виконуватися у конвеєрному та паралельному режимі. Це особливо актуально для систем із високою інтенсивністю дрібних задач або запитів, коли програмний планувальник на CPU може ставати “вузьким місцем” через накладні витрати на обробку подій, блокування, контекстні перемикання та синхронізацію між потоками.

FPGA як реалізаційна платформа дозволяє описувати планування як потік транзакцій: вхідними подіями є поява задачі, зміна доступності ресурсу, завершення виконання, оновлення показників черг і каналів введення-виведення; вихідними подіями є рішення про призначення, команди на запуск ядра, оновлення таблиць стану та сигнали синхронізації. У цій моделі функціональні блоки планувальника природно відображаються на апаратні модулі обчислення метрик, обчислення пріоритетів, вибору ресурсу за заданим правилом, керування чергами та диспетчеризації команд. Такий підхід важливий для гетерогенних систем, де одночасно присутні CPU, GPU та FPGA, а рішення планувальника повинні враховувати як обчислювальні характеристики ресурсів, так і часові витрати передавання даних між доменами пам’яті.

Практична побудова планувальника з використанням FPGA зводиться до формалізації правил планування у вигляді операцій над скінченними структурами даних, що зберігаються у вбудованій пам’яті FPGA або у спільній пам’яті вузла. Типовими структурами є таблиця станів ресурсів, таблиця профілів задач або класів задач, апаратні черги подій, а також механізми атомарного оновлення лічильників і міток часу. На основі цих структур апаратний модуль може реалізовувати пріоритетну дисципліну обслуговування, підтримувати кілька класів задач із різними політиками, виконувати відбір ресурсу за мінімальним прогнозованим часом завершення або за іншими критеріями, якщо вони подаються як обчислювані функції. Для FPGA-орієнтованої системи критично, що такі обчислення мають бути спроєктовані як детерміновані та обмежені за апаратними ресурсами, оскільки “складні” алгоритми планування з великою кількістю

глобальних оптимізацій у чистому вигляді можуть бути непридатні для апаратного втілення без спрощення або без ієрархізації.

Ще одним аспектом застосування FPGA як засобу побудови планування є можливість апаратної підтримки планування в inline-сценаріях, коли обробка даних відбувається “у тракті”, наприклад у мережевих або потокових системах. У таких випадках планування часто означає не класичне призначення задач до виконавців, а керування пріоритетами потоків, маршрутизацією даних між конвеєрами обробки, виділенням буферів і підтримкою якості обслуговування. FPGA може реалізувати ці функції як частину мережевого або обчислювального тракту, використовуючи апаратні черги та арбітраж доступу до ресурсів. У гетерогенній системі це дозволяє об’єднати механізм обробки даних і механізм планування у спільному апаратному контурі, де рішення ухвалюються за фактом надходження даних і подій, а не лише на рівні програмного керування.

З точки зору розподілу задач у гетерогенній системі застосування FPGA як планувального модуля набуває особливого значення в умовах, коли самі задачі або їхні параметри надходять потоком, а ресурси змінюють доступність у часі. Тоді апаратний планувальник може виконувати класифікацію задач за атрибутами, підтримувати апаратні профілі виконання для різних виконавців і формувати рішення про перенесення задачі на той чи інший тип ресурсу, з урахуванням поточного стану черг і каналів. У випадку наявності кількох FPGA або кількох конфігураційних контекстів на одному FPGA рішення розподілу додатково включає вибір конфігураційного стану або вибір “слоту” для ядра, а також порядок виконання конфігураційних переходів. Тобто планування для FPGA-ресурсів може бути двошаровим: на верхньому рівні визначається, чи виконується задача на CPU/GPU/FPGA, а на нижньому як саме вона розміщується і в якому порядку всередині FPGA-простору.

Важливою є й системна інтеграція апаратного планувальника з програмним оточенням. У гетерогенних системах керування ресурсами часто реалізується через драйвери, бібліотеки часу виконання та системні служби, що відповідають за виділення буферів, підготовку команд, синхронізацію та облік. Використання

FPGA як засобу побудови планування означає необхідність чіткого розмежування відповідальності між апаратною та програмною частинами: які стани вимірюються та передаються на FPGA, які рішення приймаються апаратно, які команди формуються та як забезпечується узгодженість (консистентність) таблиць станів при паралельних подіях. У практичній моделі це часто реалізується як обмін через memory-mapped регістри або через черги команд у спільній пам’яті, що забезпечує мінімальний шлях передачі подій і рішень між CPU і FPGA.

Застосування FPGA як засобу побудови планування та розподілу задач доцільно трактувати як апаратно-програмний підхід, у якому FPGA використовується для реалізації частини функцій диспетчеризації, арбітражу, керування чергами та прийняття рішень про призначення задач. У межах теми магістерської роботи це відкриває можливість формалізувати метод планування так, щоб він враховував як специфіку FPGA як виконавця, так і потенціал FPGA як носія логіки керування, що є важливим для систем із високою подієвістю та різнорідністю ресурсів, де критичною є швидкість і детермінованість прийняття планувальних рішень.

1.5 Висновки до першого розділу та постановка задачі

У результаті аналізу предметної області планування та розподілу задач у гетерогенних обчислювальних системах встановлено, що ефективність функціонування таких систем визначається не лише обчислювальною продуктивністю окремих компонентів, а й організацією пам’яті, характеристиками міжкомпонентного обміну даними, політиками диспетчеризації та здатністю системи адаптуватися до різних типів навантаження.

У результаті аналізу показано, що FPGA займає особливе місце серед ресурсів гетерогенних систем, оскільки поєднує реконфігурованість, можливість глибокої конвеєризації та високий рівень паралелізму обробки даних. Водночас використання FPGA у задачах планування та розподілу робіт потребує врахування специфічних чинників, зокрема обмежень за LUT, FF, BRAM і DSP, витрат на

передавання даних, часу конфігурації або часткової реконфігурації, а також залежності продуктивності від конкретної HLS/RTL-реалізації. Це означає, що класичні підходи до планування, орієнтовані на однорідні або слабко гетерогенні середовища, не можуть бути безпосередньо використані без адаптації.

Аналіз сучасних методів планування засвідчив, що найбільш перспективними є адаптивні, багатокритеріальні та інтелектуальні підходи, які враховують динаміку стану ресурсів, структуру задач, вимоги до QoS, обмеження за затримкою, вартістю й надійністю. Разом із тим для FPGA-орієнтованих систем недостатньо враховувати лише вибір виконавця: необхідно також моделювати конфігураційний стан прискорювача, сумісність задач із наявною апаратною реалізацією та доцільність перенесення обчислень з урахуванням комунікаційних витрат.

Отже, за результатами дослідження обґрунтовано доцільність розроблення спеціалізованого методу планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA, який має поєднувати вибір ресурсу, оцінювання вартості передавання даних, урахування внутрішніх ресурсних обмежень FPGA та керування конфігураційними переходами. Крім того, встановлено, що FPGA може розглядатися не лише як виконавчий ресурс, а і як засіб апаратно-програмної реалізації самого планувальника, що відкриває можливість підвищення швидкодії, детермінованості та ефективності прийняття рішень у системах із високою інтенсивністю подій.

Поставлена мета досягається розв’язанням таких основних завдань:

проаналізувати відомі методи оптимізації продуктивності обчислювальних систем;

- розробити метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA;

- здійснити дослідження методу планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA;

- реалізувати метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA у вигляді системи.

2 ПОБУДОВА МОДЕЛІ СИСТЕМИ

2.1 Модель системи

У сучасних умовах розвитку високопродуктивних обчислень та кіберфізичних систем особливої актуальності набуває задача ефективного використання гетерогенних обчислювальних ресурсів, що поєднують різнорідні за архітектурою та продуктивністю процесорні вузли.

Такі системи характеризуються відмінностями у швидкодії, енергоспоживанні, обсягах пам’яті та пропускній здатності каналів передавання даних, що істотно ускладнює процес узгодженого виконання обчислювальних завдань. За цих умов традиційні підходи до планування, орієнтовані на однорідне середовище, втрачають ефективність і не забезпечують належного рівня використання ресурсів.

Розв’язання зазначеної проблеми потребує формалізації обчислювального процесу у вигляді моделі, що адекватно відображає як структуру самих завдань, так і взаємозв’язки між ними.

У даному контексті доцільним є використання моделі орієнтований ациклічний граф, яка дозволяє подати обчислювальний процес як сукупність вершин, що відповідають окремим підзадачам, та дуг, які визначають відношення передування і залежності за даними. Така модель забезпечує однозначне визначення допустимих порядків виконання підзадач і створює основу для подальшого планування.

Великі за обсягом модельні задачі, що допускають декомпозицію та паралельне виконання, надходять до централізованого планувальника, де формується узагальнене подання обчислювального процесу. При цьому виникає необхідність визначення такого відображення підзадач на доступні обчислювальні ресурси, яке б враховувало як топологію залежностей, так і характеристики обчислювальних вузлів.

Ускладнення полягає в тому, що різні підзадачі можуть по-різному поводитися на різних типах процесорів, а витрати часу на передавання даних між

вузлами можуть бути співставними або навіть перевищувати час виконання самих обчислень.

Додатковим обмежувальним чинником виступає необхідність забезпечення енергоефективності функціонування системи, що є критичним для розподілених та вбудованих середовищ.

Таким чином, задача планування набуває багатокритеріального характеру і передбачає одночасну оптимізацію тривалості виконання обчислювального процесу, рівня завантаження ресурсів та енергетичних витрат.

Водночас необхідно забезпечити узгодженість виконання залежних підзадач і мінімізувати простої обчислювальних вузлів, що виникають через нераціональний розподіл робіт.

Отже, постає науково-прикладна задача розроблення методу планування та розподілу задач у гетерогенних обчислювальних системах, який би враховував структурні особливості обчислювального процесу, представленого у вигляді орієнтованого ациклічного графа, а також різнорідність обчислювального середовища.

Такий метод має забезпечувати побудову ефективного розкладу виконання підзадач на множині доступних процесорів із урахуванням обмежень на залежності, ресурси та енергоспоживання, що в сукупності дозволить зменшити загальний час обробки та підвищити ефективність функціонування системи в цілому.

Задамо систему позначень в дослідженні (Таблиця 2.1).

Практична розподілена обчислювальна платформа у сучасних умовах характеризується наявністю значної кількості складних і різнорідних обчислювальних ресурсів, що відрізняються за архітектурними особливостями, обчислювальною потужністю, енергетичними характеристиками та параметрами взаємодії.

Таблиця 2.1 – Система позначень в дослідженні

Нотація	Опис
Q	Робочий процес орієнтований ациклічний граф
F_i	і-та обчислювальна задача робочого процесу
M_j	j-й термінальний процесор
A_{ij}	Затримка поширення між і-м завданням та j-м завданням
B_{ij}	Затримка обробки j-го завдання на і-му процесорі
C	Кількість процесорів
R	Кількість завдань
O_i	Поточне завдання
hr_i	Попереднє завдання і-го завдання
xu_i	Наступне завдання і-го завдання
$fc_{sna}^{o(i)}$	Час початку і-го завдання
$fc_{end}^{o(i)}$	Час завершення і-го завдання
$u_{sta}^{c(i)}$	Статичне енергоспоживання і-го процесора
$u_{dyn}^{c(i)}$	Динамічне енергоспоживання і-го процесора
p_i	Стан процесора
cs	Максимальний часовий проміжок виконання завдань робочого процесу
λ_{ij}	Надійність j-го процесора за одиницю часу
L_Q	Повна надійність завдань робочого процесу
L_j^i	Надійність і-го завдання, що виконується на j-му процесорі
y_{ij}	Повна надійність завдань робочого процесу

Таке середовище формується як сукупність взаємопов’язаних обчислювальних вузлів, кожен з яких здатний виконувати визначений клас задач із різною ефективністю. Для формалізації цієї системи введемо множину

обчислювальних ресурсів $M = \{M_1, M_2, \dots, M_C\}$, де кожен елемент відповідає окремому термінальному процесору або вузлу, а їх загальна кількість дорівнює C. Водночас розглядається множина завдань $F = \{F_1, F_2, \dots, F_R\}$, що підлягають виконанню, причому кожне із завдань може бути декомпозоване на підзадачі або розглядатися як атомарна обчислювальна одиниця залежно від рівня абстракції моделі.

У випадку, коли зазначені R завдань необхідно розподілити між Cобчислювальними ресурсами з метою їх паралельного виконання, реальний робочий процес доцільно подати у вигляді формалізованої структури, що відображає як внутрішні залежності між завданнями, так і особливості їх виконання у розподіленому середовищі.

Для цього використовується модель орієнтований ациклічний граф $Q(A, B, N)$, яка забезпечує наочне та математично строге представлення процесу обчислень.

У межах цієї моделі вершини графа відповідають окремим завданням або підзадачам, тоді як дуги визначають відношення передування, тобто логічні та інформаційні залежності між ними, що обмежують можливість їх одночасного виконання.

Матриця Aу даній моделі відіграє ключову роль, оскільки описує структуру залежностей між завданнями, а також включає параметри, що характеризують затримки передавання даних між відповідними обчислювальними вузлами.

Кожен елемент цієї матриці відображає не лише факт існування залежності, але й кількісну оцінку часу, необхідного для передачі результатів виконання однієї підзадачі до іншої. Таким чином, урахування комунікаційних витрат стає невід’ємною складовою процесу планування, особливо в умовах, коли обсяг передаваних даних є значним, а мережна інфраструктура має обмежену пропускну здатність.

Матриця Bвизначає часові характеристики виконання завдань і відображає тривалість обробки кожного завдання на кожному з доступних обчислювальних ресурсів.

При цьому важливо враховувати, що одна й та сама задача може виконуватися з різною швидкістю на різних процесорах через відмінності у їхній архітектурі, тактовій частоті, наявності спеціалізованих обчислювальних блоків або ефективності використання пам’яті. Саме ця неоднорідність і формує складність задачі планування, оскільки вибір ресурсу для виконання конкретного завдання безпосередньо впливає на загальний час завершення обчислень.

Матриця Nвідображає енергетичні витрати, пов’язані з виконанням завдань на відповідних процесорах, що дозволяє враховувати ще один критично важливий аспект функціонування системи. У сучасних обчислювальних середовищах, особливо в умовах обмежених енергетичних ресурсів або необхідності дотримання вимог енергоефективності, мінімізація енергоспоживання стає не менш важливою, ніж скорочення часу виконання. Таким чином, задача розподілу завдань набуває багатокритеріального характеру, де необхідно знаходити компроміс між швидкодією та енергетичними витратами.

Особливу складність моделі зумовлює той факт, що обчислювальні вузли з’єднані між собою каналами зв’язку з різними характеристиками, зокрема різною пропускною здатністю та затримками передавання. Це означає, що час передачі даних між будь-якою парою процесорів не є сталим і залежить як від фізичних параметрів каналу, так і від поточного стану мережі. У результаті, затримка передавання даних між завданнями, які виконуються на різних вузлах, може суттєво впливати на загальну ефективність виконання всього обчислювального процесу. Крім того, час виконання одного й того ж завдання U_{ij}на різних процесорах M_j, а також відповідні затримки передавання A_{ij}, не є сталими величинами і змінюються залежно від конкретного обчислювального ресурсу. Це дозволяє адекватно відобразити реальні умови функціонування гетерогенних систем, у яких відсутня універсальність виконання, а ефективність обробки суттєво залежить від відповідності задачі характеристикам обраного процесора.

Такий підхід забезпечує більш точне моделювання поведінки системи та створює передумови для розроблення методів планування, що здатні враховувати як обчислювальні, так і комунікаційні аспекти виконання задач.

В таблиці 2.2 наведено ілюстративний приклад подання обчислювального процесу у вигляді орієнтованого ациклічного графу, у межах якої розглядається дев’ять взаємопов’язаних завдань, що підлягають виконанню на трьох різних обчислювальних процесорах. Така модель дозволяє формалізувати не лише множину самих завдань, але й структуру їхніх залежностей, що є визначальним чинником при побудові ефективного розкладу виконання у гетерогенному середовищі.

Таблиця 2.2 – Час та енергоспоживання завдання на гетерогенному процесорі

Завдання	P1	P2	P3
1	14	6	9
2	13	5	8
3	9	8	11
4	10	15	7
5	12	13	10
6	18	10	8
7	21	17	22
8	18	19	24
9	23	7	16
статичне споживання енергії	0.05	0.04	0.06
динамічне споживання енергії	0.8	1.1	1

У зазначеній моделі кожна вершина графа інтерпретується як окреме завдання або обчислювальна операція, тоді як орієнтовані ребра відображають відношення передування між завданнями, а також асоційовані з ними затримки передавання даних.

Наявність ребра, що спрямоване від завдання F_iдо завдання F_j, означає, що результат виконання F_iє необхідною вхідною інформацією для F_j. За таких умов F_iрозглядається як попередник відносно F_j, а F_j, відповідно, як наступник. Це визначає строгий частковий порядок виконання, відповідно до якого жодне завдання-наступник не може розпочати свою обробку до моменту повного завершення всіх своїх завдань-попередників.

Нехай розглядається довільне завдання O_i, для якого у структурі графа можна визначити множину безпосередніх попередників hr_i та множину безпосередніх наступників su_i.

У загальному випадку кількість таких попередників і наступників не обмежується одиницею, що відображає складний характер залежностей у реальних обчислювальних процесах.

Виконання завдання O_iможе бути ініційоване лише після того, як усі завдання з множини його попередників завершили свою роботу та передали необхідні результати. Така вимога формує умови синхронізації, які істотно впливають на загальний час виконання всього процесу.

Для будь-якої пари завдань, між якими існує відношення передування, характерною є залежність вхідних даних одного завдання від вихідних даних іншого. Це означає, що у випадку їх розміщення на різних обчислювальних ресурсах виникає додатковий часовий компонент, пов’язаний із передаванням даних через мережну інфраструктуру.

У такій ситуації загальна затримка включає як час обчислення, так і час комунікації. Водночас, якщо обидва завдання виконуються на одному і тому ж процесорі, необхідність у передаванні даних між вузлами відпадає, і визначальним чинником стає лише тривалість обчислення. Проте навіть у межах одного процесора час виконання різних завдань може істотно відрізнятися, що зумовлено їхньою обчислювальною складністю, обсягом оброблюваних даних та ефективністю використання апаратних ресурсів.

Додатковим фактором, що ускладнює процес планування, є гетерогенність обчислювального середовища.

Різні процесори можуть мати відмінні характеристики продуктивності, що призводить до ситуації, коли одне й те саме завдання виконується з різною швидкістю залежно від вибраного ресурсу.

У крайніх випадках окремі процесори можуть бути непридатними для виконання певних завдань через відсутність необхідних апаратних або програмних можливостей.

У такій ситуації відповідний час виконання формально вважається нескінченним, що фактично виключає можливість призначення такого завдання на даний ресурс у процесі побудови розкладу.

У структурі орієнтованого ациклічного графа особливе значення мають завдання, які не мають попередників, а також ті, що не мають наступників.

Перші інтерпретуються як початкові або вхідні завдання, виконання яких може бути розпочато без попередніх умов, тоді як другі розглядаються як завершальні або вихідні завдання, результати яких формують кінцевий результат усього обчислювального процесу.

У випадках, коли граф містить декілька таких початкових або завершальних вершин, доцільним є введення віртуальних завдань із нульовими ваговими характеристиками, які штучно об’єднують множину входів або виходів у єдину вершину.

Це спрощує подальший аналіз і дозволяє розглядати систему як таку, що має один початковий і один кінцевий елемент, які умовно позначаються як F_1 та F_R відповідно.

Додатково передбачається, що всі завдання характеризуються однаковими часовими обмеженнями щодо завершення та виконуються без переривань, тобто після початку виконання завдання на певному ресурсі його обробка триває до повного завершення без можливості призупинення або перенесення на інший ресурс.

Така умова, з одного боку, спрощує математичну постановку задачі, а з іншого боку наближує модель до ряду практичних сценаріїв, де перемикання контексту або міграція завдань є надто витратними або технічно обмеженими.

Орієнтований ациклічний граф подано на рисунку 2.1.

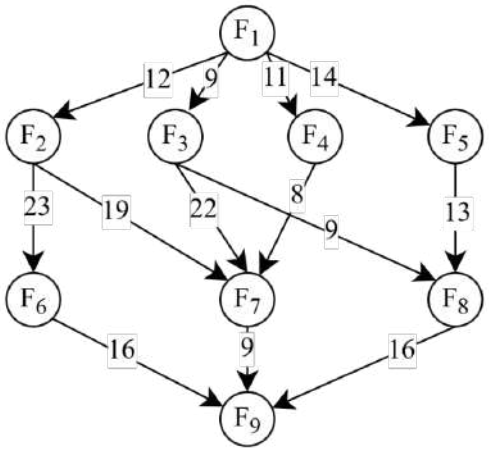


Рисунок 2.1 – Орієнтований ациклічний граф

2.2 Модель енергозберігаючого планування

У межах розглядуваної задачі планування та розподілу завдань у гетерогенних обчислювальних системах одним із ключових критеріїв ефективності функціонування виступає сукупне енергоспоживання, яке безпосередньо впливає як на експлуатаційні витрати, так і на надійність та автономність системи. Загальне споживання енергії формується як інтегральний показник, що враховує як витрати енергії під час активного виконання обчислень, так і енергетичні втрати, пов’язані з підтримкою працездатності обчислювальних ресурсів у часі.

У цьому контексті доцільно розрізняти динамічну та статичну складові енергоспоживання, які в сукупності визначають повний енергетичний баланс системи.

$$U = U_{\text{стат.}} + U_{\text{дин.}} = \sum_{i=1}^{C} u_{\text{стат.}}^{i} + \sum_{i=1}^{C} u_{\text{дин.}}^{i} \quad (2.1)$$

У наведеному співвідношенні відображено, що загальне енергоспоживання визначається сумою статичних та динамічних витрат для кожного з C обчислювальних ресурсів.

При цьому для кожного процесора встановлюється певна робоча частота k_i, яка характеризує інтенсивність його функціонування. З метою уніфікації розрахунків максимальне значення частоти нормалізується до одиниці, що дозволяє розглядати всі інші значення у відносній шкалі, де $k_i \in (0,1)$. Така нормалізація спрощує аналітичний опис моделі та дозволяє коректно порівнювати різні режими роботи процесорів незалежно від їх абсолютних характеристик.

Динамічна складова енергоспоживання безпосередньо пов’язана з процесом виконання завдань і визначається витратами енергії, що виникають під час активних обчислень.

Вона залежить як від тривалості виконання завдання, так і від частоти роботи процесора, а також від індивідуальних енергетичних характеристик конкретного обчислювального ресурсу.

$$U_{\text{дин.}} = \sum_{i=1}^{R} N_{\text{дин.}}^{c(i)} \cdot k_i \cdot B_{ij}, \qquad (2.2)$$

З цього виразу випливає, що енергетичні витрати для кожного завдання визначаються добутком коефіцієнта динамічного енергоспоживання відповідного процесора, його робочої частоти та часу виконання завдання.

У гетерогенному середовищі ці параметри можуть істотно відрізнятися, оскільки більш продуктивні процесори, як правило, забезпечують менший час обробки, але при цьому характеризуються підвищеним рівнем енергоспоживання.

Для кожного завдання формується індивідуальний компроміс між швидкістю виконання та витратами енергії, що ускладнює процес оптимального вибору ресурсу.

Разом із динамічною складовою суттєву роль відіграє статичне енергоспоживання, яке виникає незалежно від факту виконання обчислень і обумовлене необхідністю підтримки процесора у працездатному стані. Це включає витрати на живлення допоміжних підсистем, втрати на витік струму та інші фонові процеси, що відбуваються навіть за відсутності активного навантаження.

$$U_{\text{стат.}} = \sum_{i=1}^{C} N_{\text{стат.}}^{c(i)} \cdot p_i \cdot cs. \quad (2.3)$$

У цьому випадку введення бінарного параметра p_iдозволяє формалізувати стан процесора, де значення одиниці відповідає активному режиму, а нуль — вимкненому стану.

Статичне енергоспоживання враховується лише для тих ресурсів, які перебувають у робочому стані. Коефіцієнт csвідображає тривалість або масштаб часу, протягом якого процесор залишається активним, що дозволяє інтегрувати часовий аспект у модель енергоспоживання.

Важливо підкреслити, що мінімізація енергоспоживання на рівні окремого завдання не гарантує досягнення глобального оптимуму для всієї системи. Це пояснюється взаємозалежністю між динамічними та статичними складовими. Зокрема, зменшення динамічних витрат може досягатися за рахунок використання менш продуктивних процесорів або зниження частоти їх роботи, що призводить до збільшення часу виконання.

У свою чергу, збільшення тривалості обробки спричиняє зростання статичного енергоспоживання, оскільки процесори довше перебувають у активному стані. Таким чином, виникає складний компроміс між швидкодією та енергоефективністю, який необхідно враховувати при побудові алгоритму планування.

Додатковий рівень складності вносить необхідність врахування затримок, пов’язаних із передаванням даних між завданнями, особливо у випадку їх розміщення на різних процесорах.

Для завдання O_i, виконання якого можливе лише після завершення всіх його попередників hr_i, момент початку обчислення визначається не лише часом завершення попередніх завдань, але й витратами часу на передавання відповідних даних. У спрощеному випадку, коли кожне завдання має одного безпосереднього попередника, момент початку виконання може бути формалізований залежно від розміщення завдань на обчислювальних ресурсах.

$$fc_{\text{стат.}}^{o(i)} = fc_{\text{кінц.}}^{hr(i)}. \quad (2.4)$$

Це співвідношення описує ситуацію, коли обидва завдання виконуються на одному процесорі, і, відповідно, відсутні додаткові витрати часу на передавання даних. У випадку ж, коли завдання розміщені на різних процесорах, необхідно враховувати затримку комунікації:

$$fc_{\text{стат.}}^{o(i)} = fc_{\text{стат.}}^{hr(i)} + A_{o(i)hr(i)}. \quad (2.5)$$

При виборі обчислювального ресурсу для виконання конкретного завдання необхідно враховувати не лише локальні характеристики, пов’язані з часом виконання та динамічним енергоспоживанням, але й глобальні фактори, зокрема витрати на передавання даних і вплив цього вибору на подальший перебіг виконання залежних завдань. Крім того, прийняте рішення щодо призначення завдання певному процесору впливає на стан системи в цілому, включаючи доцільність активації або деактивації окремих ресурсів, що безпосередньо позначається на статичному енергоспоживанні.

2.2 Модель несправності

У реальних розподілених обчислювальних системах показники надійності окремих обчислювальних ресурсів не є сталими величинами, а змінюються під впливом як внутрішніх, так і зовнішніх чинників, зокрема режимів роботи, навантаження, температурних умов та характеристик живлення.

У зв’язку з цим задача оцінювання та забезпечення надійності функціонування системи набуває принципового значення при розробленні методів планування, оскільки відмова окремого вузла або некоректне виконання завдання може призвести до порушення цілісності всього обчислювального процесу.

Надійність системи в загальному випадку визначається як імовірність того, що вона функціонуватиме безперервно та безвідмовно протягом заданого інтервалу часу. Для моделювання процесів відмов у обчислювальних вузлах доцільно використовувати стохастичні підходи, зокрема припущення про те, що кількість відмов підпорядковується розподіл Пуассона.

У такому випадку імовірність безвідмовної роботи протягом малого інтервалу часу Δможе бути описана експоненціальною залежністю, де параметр λінтерпретується як інтенсивність відмов, тобто середня кількість відмов за одиницю часу. Такий підхід дозволяє аналітично пов’язати часові характеристики виконання завдань із імовірністю їх успішного завершення.

Для конкретного завдання F_i, яке виконується на процесорі M_j, імовірність безвідмовного виконання визначається з урахуванням тривалості його обробки. Чим довше триває виконання завдання, тим вищою є ймовірність виникнення відмови, що обумовлено накопиченням ризику у часі.

$$L_j^i = e^{-\lambda_j B_{ji}}. \tag{2.6}$$

Цей вираз відображає експоненційне зменшення надійності зі зростанням часу виконання завдання B_{ji}на відповідному процесорі. Таким чином, навіть за однакових умов виконання, задачі з більшою тривалістю мають меншу ймовірність завершення без відмов.

Разом із часовими характеристиками суттєвий вплив на надійність має робоча частота процесора. Зміна частоти, яка часто використовується як засіб регулювання енергоспоживання, безпосередньо впливає на інтенсивність відмов. Зниження частоти може призводити до зменшення теплового навантаження та енергоспоживання, однак одночасно може негативно позначатися на стабільності функціонування, зокрема через зменшення запасу стійкості до збурень.

Залежність інтенсивності відмов від частоти процесора може бути описана наступним співвідношенням:

$$\lambda_j = \lambda_{j,\max} \cdot 10^{\frac{q(k_{j,\max} - k_j)}{k_{j,\max} - k_{j,\min}}}. \quad (2.7)$$

У цьому виразі параметр $\lambda_{j,\max}$ відповідає інтенсивності відмов при максимальній частоті роботи процесора, тоді як k_j визначає поточну робочу частоту.

Константа q характеризує чутливість системи до зміни частоти і визначає, наскільки швидко зростає інтенсивність відмов при її зменшенні. Така залежність відображає практичні спостереження, згідно з якими режими енергозбереження можуть супроводжуватися підвищенням ризику помилок обчислення або нестабільної роботи.

Підставляючи отриманий вираз для інтенсивності відмов у формулу оцінювання надійності виконання завдання, отримуємо узагальнену модель, яка одночасно враховує як часові, так і частотні характеристики процесора:

$$L_j^i = e^{-B_{ji} \cdot \lambda_{j,\max} \cdot 10^{\frac{q(k_{j,\max} - k_j)}{k_{j,\max} - k_{j,\min}}}}. \quad (2.8)$$

З наведеного співвідношення випливає, що надійність виконання завдання є монотонно спадною функцією часу виконання та, за певних умов, зменшується зі зниженням робочої частоти процесора. Це створює додатковий компроміс між енергоефективністю та надійністю, який необхідно враховувати при побудові оптимізаційних алгоритмів.

Варто також враховувати, що збої у системі можуть виникати як внаслідок апаратних відмов процесорів, так і через помилки у виконанні окремих завдань. Для підвищення надійності функціонування системи доцільним є впровадження механізмів оперативного контролю стану обчислювальних вузлів.

Одним із ефективних підходів є використання технології періодичних сигнальних повідомлень, відомої як механізм “серцебиття”, який передбачає регулярну передачу коротких контрольних сигналів від кожного вузла до системи

моніторингу. Якщо протягом визначеного інтервалу часу очікуваний сигнал не надходить, система інтерпретує це як потенційну відмову і ініціює відповідні дії, спрямовані на відновлення працездатності, зокрема повторне виконання завдання, перенесення його на резервний ресурс або формування повідомлення про помилку.

Завдяки такому підходу можна вважати, що відмова окремого завдання має локальний характер і не впливає на здатність процесора виконувати інші завдання у наступні моменти часу. Це припущення суттєво спрощує математичну модель і дозволяє розглядати ймовірності безвідмовного виконання окремих завдань як статистично незалежні події.

У такому випадку загальна надійність виконання всього обчислювального процесу, представленого у вигляді орієнтований ациклічний граф Q, може бути визначена як добуток імовірностей безвідмовного виконання всіх завдань, що входять до його складу:

$$L_Q = \prod\nolimits_{F_i \in Q} L_j^i. \tag{2.9}$$

Такий підхід відображає кумулятивний характер надійності, де відмова будь-якого окремого завдання призводить до порушення виконання всього процесу. Отже, навіть незначне зниження надійності на рівні окремих задач може суттєво вплинути на загальний результат.

Урахування показників надійності у задачі планування є критично важливим і вимагає комплексного підходу, який поєднує аналіз часових характеристик, енергетичних параметрів та режимів функціонування обчислювальних ресурсів. Це дозволяє формувати такі стратегії розподілу завдань, які забезпечують не лише ефективність, але й стабільність роботи гетерогенних обчислювальних систем.

2.3 Обмеження дослідження

З урахуванням того, що розглядуваний обчислювальний процес представлено у вигляді паралельного робочого потоку Q, сформованого на основі моделі орієнтований ациклічний граф, ключова задача полягає у побудові такого розкладу виконання, який забезпечує узгоджене досягнення кількох взаємопов'язаних цілей.

Йдеться насамперед про мінімізацію сукупного енергоспоживання системи за одночасного дотримання обмежень на час завершення обчислювального процесу та забезпечення необхідного рівня надійності функціонування. Така постановка задачі відображає реальні умови функціонування гетерогенних обчислювальних середовищ, у яких оптимізація за одним критерієм неминуче впливає на інші показники ефективності.

Формалізація цієї задачі передбачає введення системи обмежень, що визначають допустимі варіанти розподілу завдань між обчислювальними ресурсами та регламентують порядок їх виконання. Перше з таких обмежень відображає вимогу однозначності призначення кожного завдання певному процесору і задається співвідношенням:

$$\sum_{j=1}^{C} y_{i,j} = 1, \forall i = 1, \dots, R. \qquad (2.10)$$

де бінарна змінна $y_{i,j}$ визначає факт призначення завдання iпроцесору j. Це означає, що кожне завдання повинно бути виконане рівно на одному обчислювальному ресурсі, що виключає можливість його дублювання або паралельного виконання на кількох вузлах у межах даної моделі.

Подальші обмеження пов'язані з необхідністю дотримання залежностей між завданнями, які визначаються структурою графа.

Час завершення кожного завдання-наступника повинен враховувати момент завершення всіх його попередників, а також можливі затримки, пов'язані з передаванням даних між різними процесорами. У загальному вигляді ця умова може бути представлена як:

$$fc_{\text{кінц.}}^{o(i)} \geq \max\left(fc_{\text{кінц.}}^{n(i)} + (p_{o(i)} \oplus p_{n(i)})A_{o(i)n(i)} + B_{ij} \cdot p_i\right), \quad (2.11)$$

де оператор виключного «або» відображає факт розміщення завдань на різних процесорах, що призводить до необхідності врахування комунікаційної затримки. Таким чином забезпечується коректність виконання з урахуванням усіх структурних залежностей, а також узгодженість обчислювального процесу у розподіленому середовищі.

Не менш важливим є обмеження, що виключає можливість переривання виконання завдань, тобто забороняє їх випередження або фрагментацію. Це означає, що після початку виконання завдання на певному процесорі воно має бути завершене без перерв і без перенесення на інший ресурс.

Формалізація цієї вимоги передбачає узгодження моментів початку та завершення виконання завдань, що конкурують за використання одного і того ж процесора, і може бути подана у вигляді системи нерівностей, які забезпечують неперетин відповідних часових інтервалів:

$$\left(\sum_{j=1}^{C} y_{i_1,j}\, y_{i_2,j}\right) \cdot \left(fc_{\text{стат.}}^{i_1} + \sum_{j=1}^{C} y_{i_1,j} \cdot \frac{B_{i_1,j}}{k_{i_1}} - fc_{\text{стат.}}^{i_2}\right) \cdot \left(fc_{\text{стат.}}^{i_2} + \sum_{j=1}^{C} y_{i_2,j} \cdot \frac{B_{i_2,j}}{k_{i_2}} - fc_{\text{стат.}}^{i_1}\right) \leq 0. \quad (2.12)$$

Це співвідношення гарантує, що два завдання, призначені одному процесору, не будуть виконуватися одночасно, що відповідає обмеженням реальних обчислювальних ресурсів.

Додатково вводиться обмеження, яке забезпечує дотримання глобального часу завершення всього обчислювального процесу, що є критичним для систем реального часу або систем із жорсткими вимогами до затримок:

$$fc_{\text{кінц.}}^{o(i)} \leq fc_{\text{кінц.}}^{o(R)}. \quad (2.13)$$

Це означає, що завершення кожного окремого завдання не повинно перевищувати момент завершення всього процесу, що фактично визначає граничний термін виконання.

Аналіз наведених співвідношень та моделей енергоспоживання і надійності дозволяє зробити висновок про складний характер цільової функції задачі планування. Наявність залежностей між завданнями призводить до необхідності врахування вкладених максимумів і умовних переходів, що формує нелінійні залежності.

Додатково ускладнення виникає через мультиплікативний характер взаємозв'язку між параметрами енергоспоживання, тривалості виконання та розподілу завдань між процесорами.

У результаті цільова функція набуває вираженого нелінійного характеру, що істотно обмежує можливість застосування класичних методів оптимізації.

Особливістю цієї задачі є також змішаний характер змінних.

Зокрема, змінні, що визначають стан обчислювальних ресурсів або факт призначення завдань, мають дискретну, зокрема бінарну природу, тоді як часові параметри, такі як моменти початку і завершення виконання, є неперервними величинами.

Поєднання таких змінних у межах однієї математичної моделі призводить до формування задачі, яка належить до класу змішане цілочисельне нелінійне програмування.

Складність розв'язання цієї задачі обумовлена сукупністю взаємопов'язаних факторів.

По-перше, структурні залежності між завданнями формують складну топологію обчислювального процесу, яка вимагає врахування як часових, так і комунікаційних аспектів.

По-друге, необхідність розподілу великої кількості завдань між множиною гетерогенних процесорів створює комбінаторний простір можливих рішень,

розмірність якого зростає експоненційно зі збільшенням розмірності задачі. По-третє, нелінійний характер цільової функції ускладнює застосування стандартних оптимізаційних процедур і потребує використання спеціалізованих підходів.

У сукупності ці фактори дозволяють класифікувати розглядувану задачу як таку, що належить до класу NP-складні задачі. Це означає, що для неї не існує відомих алгоритмів, здатних знаходити точний оптимальний розв’язок за поліноміальний час для довільного розміру вхідних даних.

У зв’язку з цим доцільним є застосування наближених або евристичних методів оптимізації, які дозволяють отримати близькі до оптимальних рішення за прийнятний час, що і визначає напрям подальших досліджень у даній роботі.

2.11 Висновки

У другому розділі магістерської роботи здійснено формалізацію обчислювального процесу та побудовано комплексну модель системи планування задач у гетерогенних обчислювальних середовищах.

Розроблена модель системи базується на поданні обчислювального процесу у вигляді орієнтованого ациклічного графа, що дозволяє формально описати залежності між підзадачами, часові характеристики їх виконання на різнорідних процесорах та комунікаційні затримки між вузлами. Введена система позначень забезпечує однозначність математичного опису всіх компонентів системи, зокрема множини обчислювальних ресурсів, множини завдань та відповідних матриць параметрів. Показано, що гетерогенність середовища виявляється як у різних часах виконання однієї й тієї ж задачі на різних процесорах, так і у відмінних енергетичних характеристиках ресурсів.

Побудована модель енергозберігаючого планування розкладає сукупне енергоспоживання системи на динамічну та статичну складові. Динамічна складова визначається витратами під час активних обчислень і залежить від коефіцієнта динамічного споживання процесора, його робочої частоти та тривалості виконання завдань. Статична складова обумовлена підтримкою процесорів у працездатному

стані та залежить від тривалості їх перебування в активному режимі. Встановлено, що між цими складовими існує складний взаємозв'язок: зниження динамічних витрат шляхом зменшення робочої частоти призводить до збільшення часу виконання і, як наслідок, до зростання статичного енергоспоживання, що формує нетривіальний компроміс при оптимізації.

Розроблена модель надійності враховує стохастичну природу відмов обчислювальних вузлів на основі розподілу Пуассона. Отримано аналітичний вираз для оцінювання надійності виконання окремого завдання на конкретному процесорі, що має експоненційний характер і залежить як від тривалості виконання, так і від робочої частоти ресурсу. Загальна надійність обчислювального процесу визначається як добуток надійностей усіх складових завдань, що підкреслює кумулятивний характер ризиків у розподілених системах.

Формалізовано систему обмежень задачі планування, що включає вимогу однозначності призначення кожного завдання одному процесору, дотримання відношень передування між залежними задачами з урахуванням комунікаційних затримок, заборону переривання виконання завдань та дотримання глобального часового дедлайну. Аналіз сформованої математичної постановки показав, що вона є задачею змішаного цілочисельного нелінійного програмування, яка належить до класу NP-складних задач. Це обумовлено одночасною присутністю дискретних змінних призначення та неперервних часових параметрів, а також нелінійним характером цільової функції внаслідок мультиплікативних залежностей між параметрами енергоспоживання, надійності та розподілу ресурсів.

3 МЕТОД ПЛАНУВАННЯ ТА РОЗПОДІЛУ ЗАДАЧ У ГЕТЕРОГЕННИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ НА БАЗІ FPGA

3.1 Основи методу планування та розподілу задач у гетерогенних обчислювальних системах

У дослідженні пропонується удосконалений метод енергоефективного планування задач у гетерогенних паралельних системах, який на відміну від відомих поєднує евристичну оптимізацію та навчання з підкріпленням із урахуванням залежностей задач і неоднорідності ресурсів, і який дає змогу зменшити енергоспоживання та підвищити надійність і швидкодію системи за рахунок динамічної надлишковості [112]

Метод планування, орієнтований на розв’язання задачі розподілу та впорядкування виконання завдань робочого процесу в умовах використання складних гетерогенних обчислювальних ресурсів. Запропонований підхід базується на поєднанні аналітичної оптимізаційної моделі та адаптивних механізмів прийняття рішень, що дозволяє враховувати як формалізовані обмеження задачі, так і динамічні особливості функціонування системи.

Загальна структура алгоритму передбачає послідовну реалізацію двох взаємопов’язаних фаз, кожна з яких виконує окрему функціональну роль у процесі формування ефективного розкладу.

На першому етапі здійснюється побудова початкового розкладу виконання завдань на основі формалізованої моделі змішане цілочисельне нелінійне програмування. У межах цього методу враховуються обмеження, пов’язані з залежностями між завданнями, доступністю обчислювальних ресурсів, часовими характеристиками виконання та енергетичними витратами.

Основною метою даного етапу є мінімізація сукупного енергоспоживання системи за умови дотримання встановлених вимог до часу завершення обчислювального процесу.

Узагальнена схема методу подана на рисунку 3.1.

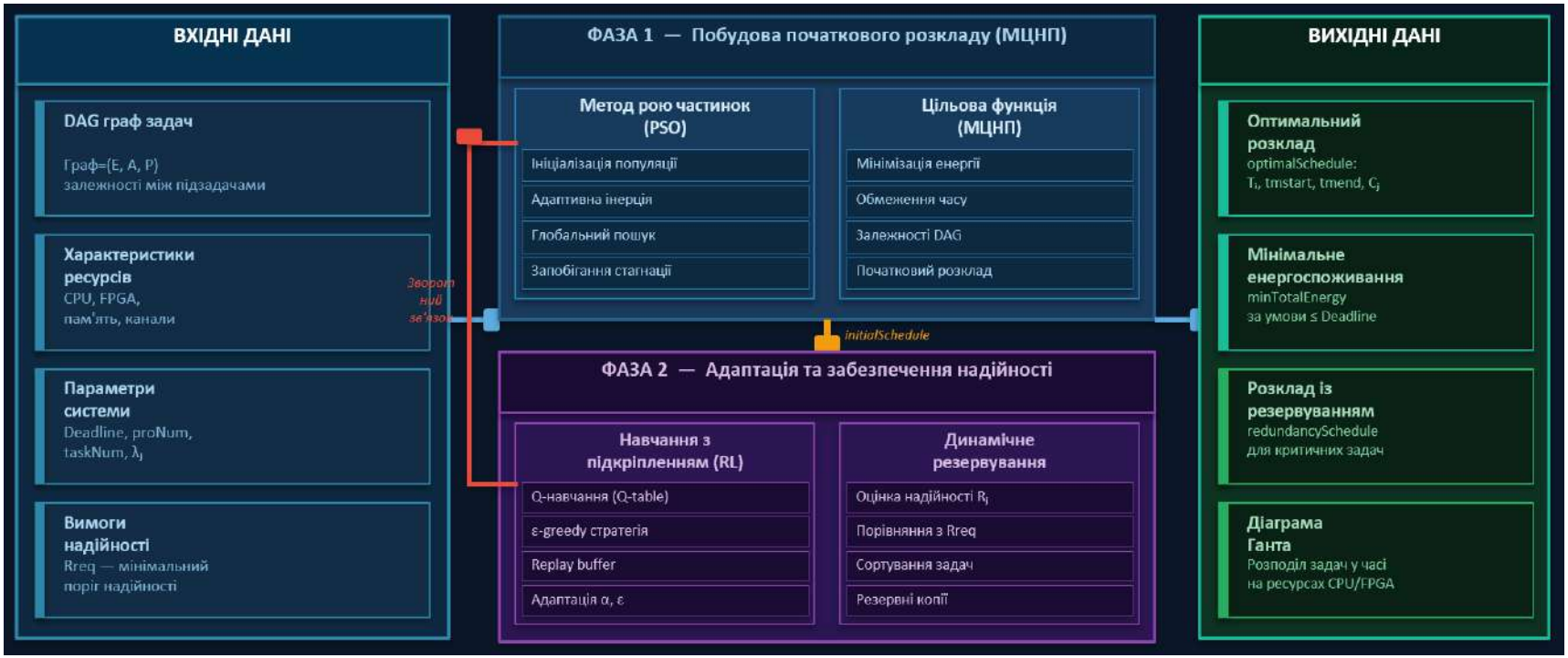


Рисунок 3.1 – Узагальнена схема методу

При цьому використовується структуроване подання задачі у вигляді орієнтований ациклічний граф, що дозволяє формалізувати частковий порядок виконання завдань і забезпечити коректність планування.

На другому етапі виконується адаптація отриманого розкладу з урахуванням показників надійності виконання окремих завдань.

Зокрема, для тих підзадач, які характеризуються зниженими значеннями імовірності безвідмовного виконання, застосовується механізм динамічного резервування. Такий підхід передбачає можливість дублювання або повторного призначення критичних завдань на альтернативні обчислювальні ресурси з метою зменшення ризику відмови.

У результаті досягається підвищення загального рівня якості обслуговування системи, що проявляється у зростанні надійності та стабільності виконання обчислювального процесу.

Для детального аналізу взаємодії між компонентами запропонованого методу використовується діаграма послідовності, яка відображає основні етапи прийняття рішень, обмін інформацією між модулями системи та порядок виконання ключових операцій у процесі планування. Така діаграма дозволяє формалізувати логіку функціонування алгоритму та забезпечує наочне уявлення про динаміку обробки завдань.

Вхідними даними для алгоритму є орієнтований ациклічний граф, що відображає залежності між підзадачами, а також характеристики обчислювальних ресурсів. Для дослідження початкового простору можливих рішень використовується евристичний підхід, який дозволяє сформувати початковий розклад із прийнятними характеристиками за обмежений час.

Водночас для подальшого вдосконалення стратегії планування застосовується апарат навчання з підкріпленням, що забезпечує адаптивне коригування прийнятих рішень на основі накопиченого досвіду. Такий підхід дозволяє досягти балансу між дослідженням нових варіантів розподілу завдань і використанням вже знайдених ефективних рішень, що є критично важливим для задач із великою розмірністю та складною структурою.

У межах формування розкладу важливу роль відіграє визначення пріоритетів виконання завдань, які задаються за допомогою рангових оцінок. Ці оцінки відображають відносне положення завдання у структурі обчислювального процесу та визначають допустимий порядок їх виконання:

$$(\text{ранг}(i) - \text{ранг}(j)) \cdot A_{ij} \geq 0, \tag{3.1}$$

Наведене співвідношення забезпечує узгодженість порядку планування із топологією графа залежностей, виключаючи можливість порушення логічної послідовності виконання завдань.

Паралельно вводиться механізм фіксації моменту завершення виконання завдання на процесорі $k_{\text{lag}}(j)$, що дозволяє синхронізувати запуск наступних підзадач $fc_{\text{кінц.}}^{o(i)}$:

$$k_{\text{lag}}(j) = fc_{\text{кінц.}}^{o(i)}. \tag{3.2}$$

Цей параметр визначає момент, з якого наступне завдання може бути допущене до виконання. Відповідно, умова $fc_{\text{кінц.}}^{o(i)} \geq k_{\text{lag}}(j)$ гарантує, що жодне завдання не розпочне виконання раніше, ніж завершиться його попередник і буде зафіксовано відповідний стан системи.

Важливим аспектом є також врахування обмеженості ресурсів граничних обчислювальних пристроїв, які часто мають обмежені обчислювальні потужності, пам’ять і пропускну здатність.

У зв’язку з цим складні задачі зазвичай декомпозуються на множину підзадач, що виконуються на різних вузлах, а їх взаємозв’язки описуються за допомогою орієнтованого ациклічного графа. Такий підхід дозволяє ефективно використовувати доступні ресурси та забезпечити масштабованість обчислювального процесу.

Незважаючи на те, що задачі даного класу належать до NP-складні задачі і традиційно розв’язуються за допомогою евристичних методів, останні часто демонструють обмежену гнучкість у динамічних умовах.

З іншого боку, підходи, засновані на навчанні з підкріпленням, здатні адаптуватися до змін середовища, але на початкових етапах можуть характеризуватися низькою ефективністю та схильністю до збіжності до локальних оптимумів.

У зв’язку з цим доцільним є поєднання цих двох підходів у межах єдиного алгоритму, що дозволяє використати їхні переваги та компенсувати недоліки.

Запропонований метод, таким чином, поєднує строгість математичного моделювання із гнучкістю адаптивного навчання, що дозволяє формувати ефективні розклади виконання завдань із мінімальним енергоспоживанням, забезпечуючи при цьому дотримання обмежень часу відгуку та необхідного рівня надійності.

Його детальна структура та послідовність виконання операцій відображені у відповідній діаграмі, яка ілюструє логіку функціонування та взаємодію основних компонентів запропонованого методу (рисунок 3.1, Алгоритм 3.1).

Вхідні дані:
орієнтований ациклічний граф задач Граф $= (E, A, P)$;
відносний дедлайн $relativeDeadline$;
кількість процесорів $proNum$;
кількість задач $taskNum$
Вихідні дані:
оптимальний розклад $optimalSchedule$;
мінімальне сумарне енергоспоживання $minTotalEnergy$

1. Ініціалізація популяції частинок:
2. для кожної частинки iсформувати початковий розклад задач.
3. Фаза глобальної оптимізації (метод рою частинок):
4. поки не виконується умова зупинки:
    - адаптивно оновлювати коефіцієнти навчання wта інерційні ваги c;
    - для кожної частинки $p \in \{1, \dots, numParticles\}$:
        - сформувати розклад
        - $schedule = [T_i, tm_{start}^{(o(i))}, A_{ij}, tm_{end}^{(o(i))}, C_j]$;

        - оновити глобально найкраще значення енергоспоживання $globalBestEnergy$
          та індивідуальні найкращі значення $personalBestEnergies$;
    - оновити швидкості та позиції частинок;
    - зберегти поточний розклад як $initialSchedule$.
5. Ініціалізація Q-функції:
6. встановити $Q \leftarrow initialSchedule$.
7. Фаза локальної оптимізації (Q-навчання):
8. поки не виконується умова зупинки:
    - вибрати процесор відповідно до політики:
      $$processor = \arg\max Q(currentTask, \cdot)$$
    - отримати досвід у вигляді трійки $(state, action, reward)$;
    - оновити Q-функцію за правилом:
      $$Q \leftarrow Q + \alpha\big(reward + \gamma Q_{target} - Q(state, action)\big)$$
    - оновити цільову функцію: $Q_{target} \leftarrow Q$;
    - адаптивно оновити параметри навчання αта ε;
    - якщо поточний час виконання менший за $relativeDeadline$, тоді
    - $optimalSchedule \leftarrow schedule$.
9. Повернення результату:
10. повернути $optimalSchedule$та $minTotalEnergy$.

Алгоритм 3.1 – Змішане цілочисельне нелінійне програмування

На початковому етапі реалізації Алгоритму 3.1, який ґрунтується на підходах метод рою частинок, ключову роль відіграє механізм керування інерційними властивостями частинок, що безпосередньо визначає характер дослідження простору рішень.

Зокрема, на стартових ітераціях кожній частинці призначається підвищене значення ваги інерції, що стимулює її рух у напрямках, відмінних від вже досліджених, і сприяє охопленню максимально широкої області пошуку. Такий підхід дозволяє уникнути передчасної збіжності та створює передумови для виявлення перспективних областей, у яких можуть міститися глобально або близькі до глобальних оптимальні рішення задачі планування з мінімальним енергоспоживанням.

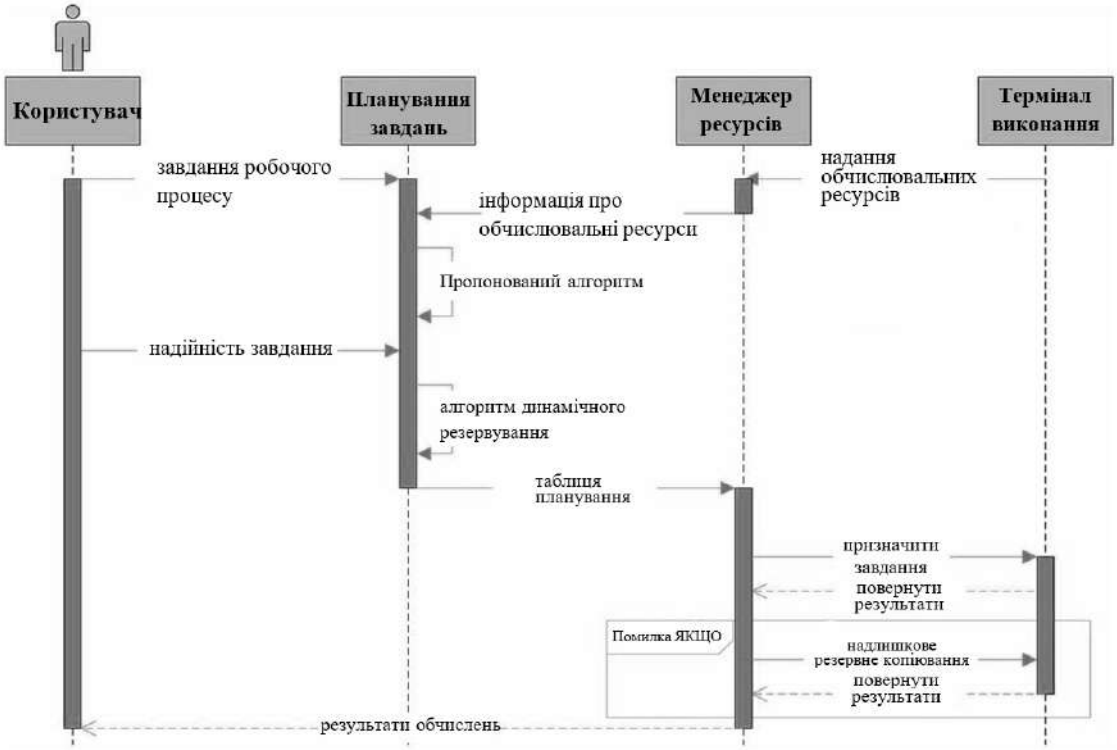


Рисунок 3.1 – Діаграма послідовності планування завдань

У процесі подальшого виконання ітерацій відбувається поступове зменшення ваги інерції, що змінює характер поведінки частинок від глобального дослідження до локального уточнення знайдених рішень. Це забезпечує більш детальне опрацювання областей простору рішень, які вже продемонстрували високу ефективність, і сприяє підвищенню точності отриманого розкладу. Така стратегія керування інерцією дозволяє поєднати переваги широкого охоплення простору та високої точності локального пошуку:

$$u = u_{\max} - \frac{(u_{\max} - u_{\min})}{F_{\max}} \cdot f. \qquad (3.3)$$

Наведене співвідношення формалізує лінійну зміну ваги інерції залежно від номера ітерації, де f визначає поточний крок алгоритму, а $F_{\max}$— максимальну кількість ітерацій. Такий механізм дозволяє плавно переходити від режиму

інтенсивного дослідження до режиму цілеспрямованої експлуатації знайдених рішень.

Додатково адаптивність алгоритму забезпечується динамічним коригуванням коефіцієнтів індивідуального та колективного навчання частинок. Ці параметри визначають, наскільки сильно частинка орієнтується на власний досвід або на найкращі рішення, знайдені всією популяцією.

У випадках, коли спостерігається збіжність популяції до локального оптимуму, підсилюється індивідуальна складова, що стимулює частинки до самостійного пошуку нових напрямків.

Натомість при значній розсіюваності рою підвищується роль колективної складової, що сприяє консолідації пошуку навколо перспективних областей.

З метою запобігання передчасному застряганню у локальних оптимумах реалізується механізм виявлення стагнації, який аналізує динаміку зміни глобально найкращого рішення. Якщо протягом певної кількості ітерацій покращення не спостерігається, ініціюється процедура часткового перезапуску, що передбачає випадкову зміну положень або швидкостей окремих частинок. Це дозволяє вивести систему з локального мінімуму та відновити процес дослідження нових областей простору рішень.

Для підвищення точності отриманих результатів використовується механізм локального пошуку, який застосовується до частинок, що вже досягли високої якості рішень.

У цьому випадку навколо поточного положення частинки генерується множина сусідніх рішень, які оцінюються за критеріями задачі. Такий підхід дозволяє здійснювати тонке налаштування параметрів розкладу та знаходити більш оптимальні конфігурації в околі вже знайдених рішень.

Важливим доповненням є механізм підтримки різноманітності популяції, який активується у випадках, коли частинки надмірно зближуються у просторі рішень.

У таких ситуаціях застосовуються невеликі випадкові збурення, що змінюють траєкторії руху частинок і сприяють відновленню дослідницької активності.

Це дозволяє уникнути деградації алгоритму та забезпечує його стійкість у складних оптимізаційних середовищах.

На початкових етапах роботи алгоритм рою частинок демонструє високу ефективність у глобальному дослідженні простору рішень, що дозволяє швидко сформувати якісне початкове наближення.

Однак із зростанням кількості ітерацій швидкість збіжності поступово зменшується, що обмежує можливість подальшого покращення результату.

Саме тому доцільним є поєднання цього підходу з методами навчання з підкріпленням, які забезпечують більш глибокий і адаптивний аналіз простору рішень.

Алгоритм навчання з підкріпленням використовує початкове рішення, сформоване методом рою частинок, як базову точку для подальшого вдосконалення.

Завдяки здатності до адаптації на основі зворотного зв’язку він дозволяє більш ефективно досліджувати локальні області та знаходити рішення, які важко отримати суто евристичними методами. Баланс між дослідженням нових варіантів і використанням вже знайдених рішень регулюється параметрами швидкості дослідження та швидкості навчання.

$$\mu = \max\left(\mu_{\min}, \mu \cdot \mu_{\text{розпад}}\right),$$
$$\varphi = \max\left(\varphi_{\min}, \varphi \cdot \varphi_{\text{розпад}}\right). \qquad (3.4)$$

Наведені співвідношення визначають механізм поступового зменшення відповідних параметрів, що забезпечує перехід від інтенсивного дослідження до більш цілеспрямованої експлуатації знайдених рішень. Це дозволяє підвищити стабільність алгоритму та прискорити його збіжність.

Для покращення якості навчання використовується механізм відтворення досвіду, який передбачає накопичення попередніх станів системи та результатів дій у спеціальному буфері.

Під час навчання випадкові підмножини цих даних використовуються для оновлення параметрів моделі, що дозволяє зменшити кореляцію між послідовними спостереженнями та підвищити стійкість процесу навчання.

Додатково застосовується підхід подвійного оцінювання, який передбачає використання двох незалежних структур оцінювання для зменшення ефекту переоцінки якості рішень. Введення цільової моделі з повільною динамікою оновлення параметрів додатково стабілізує процес навчання та запобігає різким коливанням у значеннях функції корисності.

Формування сигналу винагороди здійснюється з урахуванням цільової функції задачі, яка орієнтована на мінімізацію енергоспоживання при дотриманні обмежень часу відгуку. Це стимулює алгоритм обирати такі варіанти планування, які забезпечують більш ефективне використання ресурсів і зменшення витрат енергії.

Завершення роботи алгоритму відбувається у момент, коли подальше покращення розв’язку не спостерігається ні з боку методу рою частинок, ні з боку алгоритму навчання з підкріпленням.

У результаті формується оптимізований розклад виконання завдань, який задовольняє задані обмеження та мінімізує енергоспоживання. Отримане рішення може бути представлене у вигляді матриці призначень або у формі часових діаграм, зокрема діаграми Ганта, що дозволяє наочно відобразити розподіл завдань у часі та між обчислювальними ресурсами.

Розроблений метод планування та розподілу задач доцільно розглядати не лише як абстрактну алгоритмічну процедуру, а як компонент, що безпосередньо взаємодіє з архітектурними особливостями обчислювальних систем на базі програмованих логічних інтегральних схем. Специфіка таких систем визначається можливістю реконфігурації апаратної структури, високим ступенем паралелізму та

наявністю різнорідних обчислювальних ресурсів, що потребує відповідної адаптації механізмів планування.

У запропонованому методі обчислювальний процес представлено у вигляді орієнтованого ациклічного графа, що природно узгоджується з потоковою моделлю виконання, характерною для ПЛІС. Кожна вершина графа інтерпретується як окрема обчислювальна функція, яка може бути реалізована у вигляді апаратного модуля або програмного процесу, залежно від обраної конфігурації системи. При цьому дуги графа відповідають каналам передачі даних між функціональними блоками, що відображає реальні інформаційні потоки у структурі ПЛІС.

Перший етап методу, пов’язаний із формуванням початкового розкладу на основі математичної оптимізації, забезпечує визначення доцільного відображення задач на доступні ресурси. У контексті ПЛІС це означає вибір між різними варіантами реалізації обчислень, зокрема між виконанням на вбудованих процесорних ядрах або у вигляді спеціалізованих апаратних структур. Такий вибір здійснюється з урахуванням обмежень, пов’язаних із кількістю логічних елементів, обсягом вбудованої пам’яті та пропускною здатністю внутрішніх з’єднань.

Особливістю застосування методу у середовищі ПЛІС є необхідність врахування часу реконфігурації, який може суттєво впливати на загальну ефективність системи. У цьому контексті розклад, сформований на першому етапі, інтерпретується як послідовність не лише виконання задач, але й зміни конфігурацій апаратних ресурсів. Таким чином, планування охоплює як обчислювальні, так і структурні аспекти функціонування системи.

Другий етап методу, що передбачає адаптацію розкладу із використанням механізмів навчання, набуває особливої ваги в умовах динамічної реконфігурації. ПЛІС дозволяють змінювати свою структуру під час виконання, що відкриває можливість для оперативного перерозподілу задач між ресурсами. У цьому випадку навчальний компонент методу виконує функцію керування процесом реконфігурації, формуючи рішення щодо доцільності перенесення обчислень, зміни топології з’єднань або повторного використання апаратних модулів.

Важливим аспектом інтеграції є використання механізму динамічного резервування в апаратному середовищі. На відміну від традиційних обчислювальних систем, ПЛІС забезпечують можливість фізичного дублювання функціональних блоків або створення альтернативних шляхів обробки даних. Це дозволяє реалізувати резервування на рівні апаратної структури, що значно підвищує надійність виконання критичних задач.

Запропонований підхід до резервування, заснований на оцінюванні надійності та пріоритетності задач, безпосередньо відображається у процесі розміщення апаратних модулів. Завдання з низькою надійністю або високою критичністю можуть бути реалізовані у вигляді кількох незалежних апаратних екземплярів, розташованих у різних частинах кристала, що зменшує ймовірність одночасної відмови. Водночас для менш критичних задач використовується спільне використання ресурсів, що дозволяє зберігати баланс між надійністю та енергоефективністю.

Енергетичний аспект методу також набуває специфічного змісту в умовах ПЛІС. Оскільки споживання енергії визначається не лише інтенсивністю обчислень, але й конфігурацією логічної структури, оптимізація розкладу передбачає вибір таких варіантів реалізації, які мінімізують активність перемикання та обсяг задіяних ресурсів. У цьому контексті результати оптимізаційної моделі безпосередньо впливають на формування конфігураційних бітових потоків, що визначають фізичну структуру пристрою.

Інтеграція методу у концепцію ПЛІС також передбачає врахування ієрархічної організації ресурсів, де окремі обчислювальні вузли можуть мати різну продуктивність і спеціалізацію. Це узгоджується з ідеєю гетерогенності, закладеною у методі, та дозволяє ефективно використовувати як універсальні, так і спеціалізовані обчислювальні блоки. Відповідно, процедура планування трансформується у задачу відображення графа задач на багаторівневу апаратну структуру з урахуванням її топологічних та функціональних обмежень (рисунок 3.2).

Таким чином, запропонований метод органічно інтегрується в архітектуру обчислювальних систем на базі ПЛІС, перетворюючись із суто алгоритмічного інструменту на засіб керування апаратною реконфігурацією та розподілом ресурсів. Його застосування дозволяє узгодити процеси планування, виконання та забезпечення надійності з фізичною природою обчислювального середовища, що забезпечує підвищення продуктивності, зниження енергоспоживання та покращення стійкості системи до відмов.

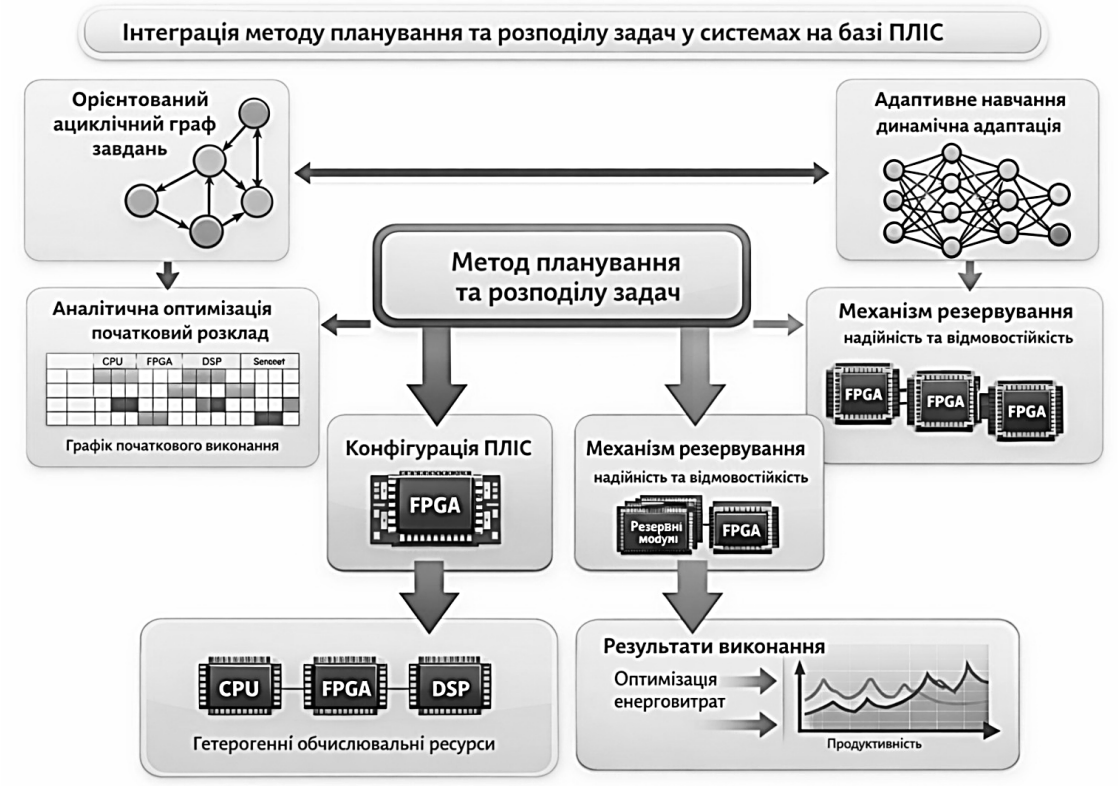


Рисунок 3.2 – Інтеграція методу у концепцію ПЛІС

3.2 Динамічне резервування

Після реалізації алгоритму планування, побудованого на основі підходу змішане цілочисельне нелінійне програмування, формується оптимізований розклад виконання завдань, який забезпечує мінімізацію енергоспоживання за

умови дотримання часових обмежень. Однак у реальних умовах функціонування розподілених обчислювальних систем така оптимальність має відносний характер, оскільки вона досягається за припущення відсутності відмов. Урахування ж стохастичної природи обчислювального середовища, де можливі апаратні збої, програмні помилки або порушення комунікацій, суттєво змінює поведінку системи.

З огляду на те, що обчислювальний процес подано у вигляді орієнтований ациклічний граф, відмова навіть одного завдання може мати каскадний ефект, порушуючи виконання всіх залежних підзадач. Це призводить до повторного виконання частини обчислень, збільшення часу завершення всього процесу та, відповідно, до зростання сукупного енергоспоживання. Тоді навіть за початково оптимального розкладу система може втратити ефективність унаслідок впливу факторів ненадійності.

Для компенсації цього негативного впливу доцільним є впровадження механізмів відмовостійкого планування, які забезпечують здатність системи зберігати працездатність навіть за наявності часткових збоїв. Одним із найбільш ефективних підходів у цьому напрямі є використання стратегії динамічного резервування, що передбачає створення та управління резервними копіями критичних завдань у процесі виконання. Така стратегія дозволяє мінімізувати наслідки відмов шляхом оперативного переключення на альтернативні варіанти виконання.

Запропонований у даному дослідженні підхід базується на принципі проактивного резервування, який передбачає не лише реакцію на вже виниклі збої, але й попереднє врахування ризиків відмов при формуванні розкладу. У межах цього підходу для кожного завдання оцінюється його надійність, після чого приймається рішення щодо доцільності створення резервної копії. Особлива увага приділяється тим завданням, які характеризуються підвищеним ризиком відмови або мають критичне значення для подальшого виконання обчислювального процесу.

Механізм динамічного резервування функціонує в режимі реального часу та адаптується до поточного стану системи. Це означає, що рішення про створення, активацію або скасування резервних копій приймаються з урахуванням доступності обчислювальних ресурсів, рівня їх завантаженості, енергетичних характеристик та поточної ймовірності відмов.

Така адаптивність дозволяє уникати надмірного використання ресурсів, яке могло б виникнути у випадку статичного резервування, і водночас забезпечує достатній рівень захисту від збоїв.

У разі виявлення потенційної або фактичної відмови система може оперативно ініціювати виконання резервного завдання на альтернативному процесорі, що дозволяє зберегти цілісність обчислювального процесу та мінімізувати затримки.

При цьому враховується як час, необхідний для активації резервного ресурсу, так і додаткові витрати енергії, пов’язані з підтримкою резервних копій. Таким чином, задача резервування інтегрується у загальну задачу оптимізації і розглядається як додатковий фактор, що впливає на баланс між енергоефективністю, швидкодією та надійністю.

Застосування динамічного резервування особливо актуальне для систем реального часу, де порушення часових обмежень може призвести до критичних наслідків.

У таких системах забезпечення гарантованого завершення завдань у визначений термін є не менш важливим, ніж мінімізація ресурсних витрат. Запропонований підхід дозволяє досягти цього за рахунок гнучкого управління резервами та адаптації до змінних умов функціонування.

Поєднання алгоритму планування, заснованого на строгій математичній моделі, із механізмами динамічного резервування формує комплексний підхід до організації обчислювального процесу. Це дозволяє не лише ефективно розподіляти завдання між гетерогенними ресурсами, але й забезпечувати стійкість системи до збоїв, зменшуючи ризик втрати результатів обчислень.

У результаті досягається підвищення загальної якості функціонування системи, що проявляється у зниженні середнього часу відгуку, зменшенні енергоспоживання в довгостроковій перспективі та зростанні надійності виконання задач.

Отже, запропонований метод, який поєднує оптимізаційне планування та динамічне резервування, створює основу для побудови ефективних і відмовостійких систем обробки задач, здатних функціонувати в умовах невизначеності та змінного навантаження.

Деталізований опис реалізації цього підходу подано у вигляді відповідного Алгоритму 3.2, що визначає послідовність дій при формуванні розкладу та управлінні резервними ресурсами.

Вхідні дані:

орієнтований ациклічний граф задач $DAG = (E, A, P)$;

інтенсивності відмов λ_j;

кількість процесорів $proNum$;

кількість задач $taskNum$

Вихідні дані:

оптимальний розклад із резервуванням $redundancySchedule$;

енергоспоживання $redundancyEnergy$

1. Оцінювання надійності системи:
2. обчислити індивідуальні показники надійності задач R_j^iта загальну надійність системи R_D.
3. Визначення максимальної надійності:
4. обчислити:

$$R_{\max} \leftarrow \prod R_j^i$$

5. Перевірка обмеження надійності:
6. якщо $R_{\max} < R_{req}$, тоді
7. завершити алгоритм із результатом *false*.

8. Формування впорядкованого списку задач:
9. інакше:
    - виконати сортування задач за показником надійності:

$$order \leftarrow sort(R_j^i)$$

оновити структуру графа:

- $DAG.E \leftarrow order$
- $DAG.W \leftarrow order$

10. Побудова розкладу:
11. для кожної задачі $t \in \{1, \dots, taskNum\}$:
    - сформувати базовий розклад:

$$schedule = [T_i, tm_{start}^{(o(i))}, A_{ij}, tm_{end}^{(o(i))}, C_j]$$

якщо виконується умова узгодженості порядку задач

- $taskOrder(t) = order(t)$та $F_t \neq 0$, тоді:
    - ініціалізувати Q-структуру: $Q \leftarrow order$;
    - модифікувати розклад із використанням резервування:

$$schedule \leftarrow redundancy$$

Повернення результату:

12. повернути $redundancySchedule$та $redundancyEnergy$.

Алгоритм 3.2 – Алгоритм динамічного резервування

Спочатку надійність кожного завдання, що виконується на кожному процесорі, розраховується на основі параметрів надійності кожного процесора, причому добуток позначається як загальна надійність. Ця загальна надійність потім порівнюється з вимогами надійності. Якщо вона вища за необхідне значення, система може працювати; в іншому випадку вимоги надійності не можуть бути виконані. Після результатів планування кожне завдання призначається окремому процесору.

Завдання з нижчими показниками надійності вибираються пропорційно для обробки резервного копіювання з резервуванням. Важливо зазначити, що процесор, вибраний для резервного завдання, не повинен бути таким самим, як оригінал, щоб у разі збою пакет даних результату резервного копіювання міг бути надісланий наступним завданням якомога швидше. Такий підхід, що передбачає обмін простору на час, підвищує загальну якість обслуговування системи.

У процесі побудови ефективного відмовостійкого механізму планування ключовим завданням є досягнення раціонального балансу між рівнем відмовостійкості та накладними витратами, пов’язаними із використанням обчислювальних ресурсів. Застосування резервування, з одного боку, підвищує імовірність успішного завершення обчислювального процесу, однак, з іншого боку, призводить до додаткового навантаження на систему, збільшення енергоспоживання та зростання часу обробки. У зв’язку з цим механізм резервування повинен базуватися на адаптивній стратегії, яка враховує як індивідуальні показники надійності завдань, так і їхню роль у структурі обчислювального процесу.

У межах розглядуваної моделі, де обчислювальний процес представлено у вигляді орієнтований ациклічний граф, кожне завдання має певний вплив на виконання наступних підзадач, що визначається його положенням у графі залежностей. Відповідно, відмова окремого завдання може мати різний ступінь критичності для системи, залежно від кількості та важливості залежних від нього операцій. Це обумовлює необхідність урахування пріоритетності завдань поряд із їхньою надійністю при прийнятті рішень щодо резервування.

Основою запропонованого підходу є використання порогового механізму оцінювання надійності. Для кожного завдання визначається його поточна імовірність безвідмовного виконання, яка порівнюється із заданим системним порогом. У випадку, коли значення цього показника знижується нижче допустимого рівня, відповідне завдання автоматично розглядається як кандидат для резервування. Такий підхід дозволяє своєчасно виявляти потенційно вразливі

елементи обчислювального процесу та підвищувати їхню надійність шляхом створення резервних копій або альтернативних шляхів виконання.

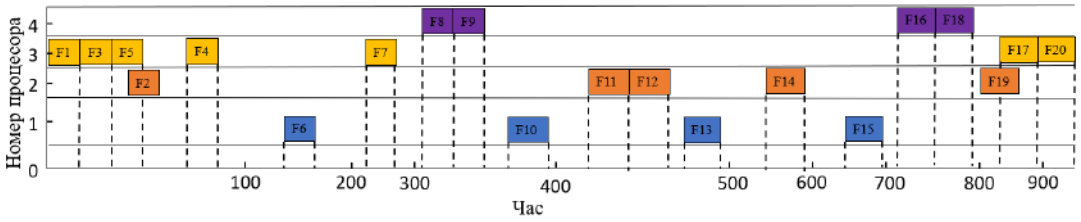


Рисунок 3.4 – Приклад діаграми Ганта для планування завдань робочого процесу

Водночас пріоритетність завдань відіграє не менш важливу роль у формуванні стратегії резервування. Завдання з високим пріоритетом, які визначають критичні ділянки обчислювального процесу або впливають на загальний час завершення, потребують підвищеного рівня захисту навіть у випадках, коли їхні показники надійності є відносно високими. Це пояснюється тим, що відмова таких завдань може спричинити значні затримки або повне порушення виконання всієї системи.

У зв’язку з цим для них доцільним є застосування більш агресивних стратегій резервування, що передбачають створення додаткових копій або використання більш надійних обчислювальних ресурсів.

На противагу цьому, завдання з низьким пріоритетом, навіть за наявності підвищеного ризику відмови, не завжди потребують інтенсивного резервування.

Надмірне використання ресурсів для їх захисту може призвести до нераціонального розподілу обчислювальної потужності та зниження загальної ефективності системи.

Тому для таких завдань застосовується обмежене або вибіркове резервування, яке дозволяє зберегти баланс між витратами ресурсів і рівнем відмовостійкості.

Запропонований підхід передбачає динамічне регулювання кількості резервних копій залежно від поточного стану системи, включаючи рівень

завантаженості процесорів, доступність ресурсів та загальні вимоги до якості обслуговування. Це дозволяє системі адаптуватися до змінних умов функціонування, забезпечуючи ефективне використання ресурсів у кожний конкретний момент часу.

У результаті формується гнучка конфігурація резервування, яка забезпечує підвищення надійності без суттєвого збільшення накладних витрат.

Таким чином, інтеграція показників надійності та пріоритетності завдань у механізм прийняття рішень щодо резервування дозволяє досягти збалансованого функціонування системи. Це забезпечує концентрацію ресурсів на найбільш критичних елементах обчислювального процесу, мінімізує ризик виникнення збоїв та водночас запобігає неефективному використанню обчислювальних потужностей.

У підсумку підвищується загальна продуктивність, стабільність і надійність функціонування гетерогенних обчислювальних систем у складних та невизначених умовах експлуатації.

3.3 Модель часової складності

Аналіз часової складності запропонованих алгоритмів є важливим етапом оцінювання їхньої практичної придатності, особливо з огляду на використання у великомасштабних гетерогенних обчислювальних системах, де кількість завдань і ресурсів може бути значною. Розглянемо детально структуру обчислювальних витрат для кожного з алгоритмів, а також інтерпретацію відповідних асимптотичних оцінок.

У межах Алгоритму 3.1, який поєднує метод рою частинок та навчання з підкріпленням, часові витрати формуються двома основними складовими, що відповідають різним фазам роботи алгоритму.

На першому етапі, пов’язаному з еволюційним оновленням популяції частинок, кожна частинка представляє потенційне рішення задачі планування.

Для коректного оцінювання такого рішення необхідно врахувати залежності між усіма завданнями, які задаються передбаченою структурою орієнтований ациклічний граф. Це означає, що для кожного завдання виконується перевірка його відношень із усіма іншими завданнями, що формує квадратичну залежність від кількості завдань:

$$O(V \cdot N \cdot R^2), \tag{3.5}$$

де V - кількість ітерацій алгоритму, тобто кількість циклів оновлення популяції частинок;

N - кількості частинок у рої, кожна з яких незалежно проходить процедуру оцінювання та оновлення;

R - кількість завдань у робочому процесі;

Квадратичний множник R^2 виникає через необхідність врахування попарних залежностей між завданнями при обчисленні пріоритетів і часових характеристик. Таким чином, ця частина складності відображає витрати на глобальне дослідження простору рішень.

На другому етапі Алгоритму 3.1 використовується механізм навчання з підкріпленням, у межах якого для кожного завдання приймається рішення щодо вибору процесора з урахуванням поточної політики. Це передбачає оцінювання як множини доступних процесорів, так і взаємозв’язків із іншими завданнями:

$$O\big(F \cdot R \cdot (C + R)\big), \tag{3.6}$$

де F - кількість ітерацій навчання, протягом яких відбувається оновлення політики;

R - кількості завдань, для кожного з яких виконується вибір дії;

C – кількість доступних процесорів, серед яких здійснюється вибір, тоді як доданок R враховує необхідність аналізу взаємозв’язків із іншими завданнями,

зокрема залежностей і обмежень. Таким чином, ця складність відображає витрати на адаптивне уточнення розкладу.

Оскільки обидві фази виконуються послідовно, загальна часова складність Алгоритму 3.1 визначається домінуючою складовою, тобто тією, яка зростає швидше при збільшенні розмірності задачі:

$$O(\max(V \cdot N \cdot R^2,\ F \cdot R \cdot (C + R))), \tag{3.7}$$

Це означає, що в реальних умовах продуктивність алгоритму визначатиметься або складністю еволюційного пошуку, або витратами на навчання, залежно від конкретних параметрів системи.

Розглянемо тепер Алгоритм 3.2, який відповідає за реалізацію механізму динамічного резервування та підвищення надійності. Його часова складність формується як сума витрат на кілька послідовних обчислювальних процедур.

Першим етапом є обчислення показників надійності для кожної пари «завдання–процесор», що вимагає перебору всіх завдань і всіх доступних ресурсів $O(C \cdot R)$, де C - кількість процесорів, а R - кількість завдань. Цей етап формує базову інформацію для подальшого прийняття рішень щодо резервування.

Далі виконується обчислення агрегованого показника надійності для всього робочого процесу, що передбачає перемноження значень надійності окремих завдань $O(R)$.

Оскільки це лінійна операція, вона не створює значного обчислювального навантаження. Порівняння отриманого значення з заданим порогом надійності виконується за сталий час і не впливає на асимптотичну оцінку $O(1)$.

Наступним важливим етапом є впорядкування завдань за рівнем їхньої надійності або пріоритетності, що дозволяє визначити черговість застосування резервування $O(R\log\ R)$.

Ця складність характерна для алгоритмів сортування порівнянням і є критичною при великих значеннях R.

Після цього виконується ітеративна обробка кожного завдання з метою уточнення параметрів планування, включаючи визначення часу початку, завершення та необхідності резервування $O(R)$.

Завершальним етапом є безпосереднє призначення резервних копій завдань відповідно до сформованого порядку, що також має лінійну складність $O(R)$.

Об’єднуючи всі складові, отримуємо узагальнену оцінку часової складності Алгоритму 3.2 $O(C \cdot R + R\log\ R + R)$.

З урахуванням домінуючих членів ця оцінка відображає баланс між витратами на обчислення надійності та операціями впорядкування.

Узагальнюючи результати аналізу для обох алгоритмів, можна записати повну часову складність запропонованого підходу як суму відповідних складових:

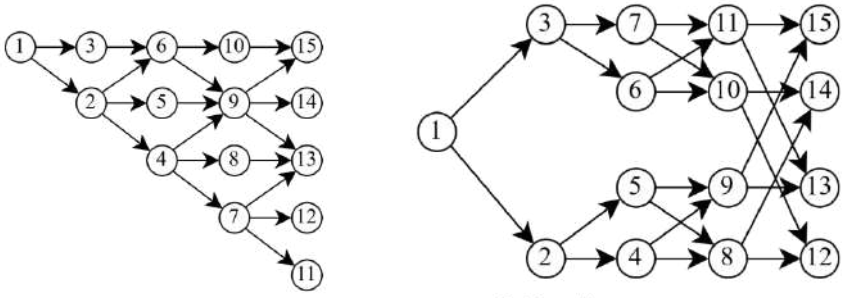
$$O(\max\,(V \cdot N \cdot R^2,\, F \cdot R \cdot (C + R))) + O(C \cdot R + R\log\ R + R).\,, \quad (3.8)$$

Цей вираз відображає комбінований характер алгоритму, у якому поєднуються евристичний глобальний пошук, адаптивне навчання та механізми підвищення надійності.

Аналіз показує, що основний внесок у часову складність робить перша частина, пов’язана з оптимізаційним пошуком, тоді як механізм резервування додає помірні додаткові витрати, що є прийнятним з огляду на суттєве підвищення відмовостійкості системи.

Рисунок 3.5 подає два паралельних робочих процеси в реальному випадку: робочий процес Гаусового виключення та робочий процес швидкого перевторення Фур'є.

Таким чином, загальна часова складність наших алгоритмів становить $\mathrm{O}(max(V\ \times \mathrm{M}\ \times \mathrm{R}^2, \mathrm{F}\ \times \mathrm{R} \times (\mathrm{C} + \mathrm{R}))) +\ \mathrm{O}(\mathrm{R}\ \times \mathrm{C} + \mathrm{R}\log\ \mathrm{R} + \mathrm{R})$.



а) б)

Рисунок 3.5 – Два паралельних робочих процеси в реальному випадку:
а) робочий процес Гаусового виключення;
б) робочий процес швидкого перевторення Фур'є

3.7 Висновки

У третьому розділі магістерської роботи розроблено та детально обґрунтовано метод планування та розподілу задач у гетерогенних обчислювальних системах на базі програмованих логічних інтегральних схем, що поєднує математичну строгість оптимізаційного моделювання з гнучкістю адаптивного навчання.

Запропонований метод реалізує двофазову архітектуру прийняття рішень. На першій фазі здійснюється побудова початкового розкладу на основі апарату змішаного цілочисельного нелінійного програмування з використанням методу рою частинок. Обчислювальний процес подається у вигляді орієнтованого ациклічного графа, що забезпечує формалізацію часткового порядку виконання задач та коректне врахування залежностей між ними. Для управління пошуком у просторі рішень застосовується адаптивний механізм зміни ваги інерції частинок, що реалізує плавний перехід від глобального дослідження до локального уточнення. Додатково введено механізми виявлення стагнації з частковим перезапуском, локального пошуку в околі перспективних рішень та підтримки різноманітності популяції, що в сукупності підвищує стійкість алгоритму до передчасної збіжності до локальних оптимумів.

На другій фазі виконується адаптивне вдосконалення отриманого розкладу за допомогою алгоритму навчання з підкріпленням на основі Q-навчання. Початкове рішення, сформоване методом рою частинок, використовується як базова точка для подальшого пошуку. Баланс між дослідженням нових варіантів розподілу задач і використанням вже знайдених ефективних рішень регулюється адаптивними параметрами швидкості дослідження та швидкості навчання, які поступово зменшуються в процесі виконання алгоритму. Для стабілізації навчання застосовується механізм відтворення досвіду та підхід подвійного оцінювання з цільовою моделлю з уповільненою динамікою оновлення параметрів, що зменшує ефект переоцінки якості рішень та запобігає різким коливанням функції корисності.

Встановлено, що поєднання двох підходів дозволяє компенсувати їхні індивідуальні недоліки: метод рою частинок забезпечує ефективне глобальне дослідження на початкових ітераціях, тоді як навчання з підкріпленням здійснює глибший адаптивний аналіз локальних областей, недоступних для суто евристичних методів. Сигнал винагороди формується з урахуванням цільової функції мінімізації енергоспоживання при дотриманні обмежень часу відгуку, що спрямовує алгоритм до енергоефективних варіантів розподілу задач.

Додатково розроблено механізм адаптації розкладу за показниками надійності, який застосовує динамічне резервування для задач з низькими значеннями імовірності безвідмовного виконання. Передбачається можливість дублювання або повторного призначення критичних завдань на альтернативні обчислювальні ресурси, що підвищує стійкість системи до відмов без суттєвого збільшення загального часу виконання.

4 ПРОГРАМНО-ТЕХНІЧНИЙ ЗАСІБ РЕАЛІЗАЦІЇ МЕТОДУ ПЛАНУВАННЯ ТА РОЗПОДІЛУ ЗАДАЧ У ГЕТЕРОГЕННИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ НА БАЗІ FPGA

4.1 Архітектури програмно-технічний засіб реалізації методу планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA

4.1.1 Загальний опис архітектурного підходу до побудови системи

Загальний архітектурний підхід до побудови системи реалізації методу планування та розподілу задач ґрунтується на використанні гетерогенної обчислювальної платформи типу SoC, у якій поєднуються процесорна підсистема загального призначення та реконфігурована апаратна логіка. Такий підхід дозволяє розділити функції керування та інтенсивних обчислень між компонентами, що мають різну природу та ефективність для різних класів задач. Основною ідеєю архітектури є винесення задач із високим ступенем паралелізму та регулярною структурою обчислень на FPGA, тоді як функції планування, диспетчеризації, аналізу стану системи та взаємодії з користувачем залишаються на стороні процесора.

Обґрунтування вибору гетерогенної архітектури з FPGA як прискорювача базується на кількох ключових положеннях. По-перше, класичні однорідні процесорні системи не забезпечують достатнього рівня продуктивності та енергоефективності при обробці задач із високим рівнем паралелізму або потоковою природою. По-друге, використання GPU, хоча і забезпечує значний приріст продуктивності, обмежене фіксованою архітектурою та меншою гнучкістю щодо адаптації під конкретні алгоритми. Натомість FPGA дозволяє реалізовувати спеціалізовані обчислювальні ядра, оптимізовані під конкретні задачі, з можливістю конвеєризації, паралельної обробки та адаптації до вимог щодо затримки та енергоспоживання. Крім того, FPGA забезпечує можливість апаратної реалізації нестандартних алгоритмів планування або їх окремих компонентів, що є важливим для запропонованого методу.

Архітектурно система реалізується на базі платформи DE10-Standard із SoC Altera Cyclone V, яка інтегрує два основні домени: HPS (процесорна підсистема) та FPGA (програмована логіка). HPS включає багатоядерний ARM-процесор, оперативну пам’ять та периферійні інтерфейси, що забезпечують виконання системного та прикладного програмного забезпечення. FPGA-домен представлений програмованою логікою, у якій реалізуються апаратні прискорювачі та спеціалізовані модулі обробки даних.

Роль процесорної підсистеми полягає у виконанні функцій глобального керування системою. Вона здійснює формування задач, побудову їх графової моделі у вигляді DAG, оцінювання параметрів задач (обчислювальна складність, обсяг даних, залежності), а також виконує алгоритм планування (зокрема HRLHS). Процесор також відповідає за прийняття рішень щодо розподілу задач між доступними ресурсами, ініціацію виконання на FPGA або CPU, синхронізацію результатів та обробку виключних ситуацій.

FPGA виконує роль високопродуктивного обчислювального прискорювача, орієнтованого на виконання обчислювально інтенсивних підзадач. У програмованій логіці реалізуються апаратні ядра, які можуть бути оптимізовані під конкретні операції або класи задач як DSP-обчислення, обробка потоків даних, матричні операції. Завдяки можливості глибокої конвеєризації та паралелізму на рівні даних FPGA забезпечує значне скорочення часу виконання та зменшення енергоспоживання.

Підсистема пам’яті виконує роль буферного середовища для обміну даними між HPS та FPGA. Вона включає як спільну оперативну пам’ять, доступну обом доменам, так і локальні буфери FPGA (BRAM). Ефективна організація пам’яті є критичною для забезпечення високої пропускної здатності та мінімізації затримок передачі даних.

Інтерфейси взаємодії, зокрема шини AXI, виконують функцію комунікаційного середовища між процесорною системою та FPGA. Вони забезпечують передачу керуючих сигналів, даних задач та результатів обчислень.

Для підвищення ефективності обміну використовуються механізми прямого доступу до пам’яті, що дозволяють зменшити навантаження на процесор.

Окремим компонентом архітектури є модуль диспетчеризації задач, який реалізує механізм передачі задач із програмного рівня на апаратний. Він забезпечує формування черг задач, контроль їх виконання, а також взаємодію з механізмом динамічної надлишковості, який відповідає за резервування та повторне виконання задач у випадку зниження надійності.

4.1.2 Загальна структурна схема системи

Загальна структурна схема системи реалізації методу планування та розподілу задач у гетерогенному середовищі на базі FPGA формується як інтегрована система-на-кристалі, у якій поєднуються процесорна підсистема (HPS), програмована логіка (FPGA), підсистема пам’яті та інтерфейси обміну даними. Така архітектура відповідає принципу тісної апаратно-програмної інтеграції, що забезпечує низькі затримки взаємодії та високу пропускну здатність передачі даних між компонентами (рис. 4.1).

Центральним елементом системи є HPS-підсистема на базі ARM-процесора, інтегрованого в Cyclone V SoC. Вона виконує функції керування обчислювальним процесом, включаючи запуск алгоритму планування, формування та аналіз графа задач, розподіл задач між ресурсами, а також координацію обміну даними. HPS працює під керуванням операційної системи, що дозволяє використовувати стандартні механізми керування пам’яттю, процесами та пристроями.

Програмована логіка FPGA є другим ключовим компонентом і виконує функцію апаратного прискорювача. У її межах реалізуються спеціалізовані обчислювальні ядра, які виконують окремі підзадачі з високим рівнем паралелізму. Ці ядра можуть бути організовані у вигляді конвеєрів обробки або набору паралельних функціональних блоків, що дозволяє ефективно обробляти потоки даних або великі масиви інформації.

Рисунок 4.1 – Загальна структурна схема системи

Підсистема пам’яті включає як зовнішню оперативну пам’ять, доступну для HPS, так і внутрішню пам’ять FPGA. У більшості конфігурацій використовується спільний адресний простір або механізм відображення пам’яті, що дозволяє обом доменам працювати з одними й тими самими даними. HPS відповідає за розміщення вхідних даних у пам’яті, після чого ці дані передаються до FPGA для обробки. Результати обчислень повертаються у пам’ять і зчитуються процесором.

Взаємодія між HPS та FPGA реалізується через високошвидкісні інтерфейси, основним з яких є шина AXI.

У структурі Cyclone V було використано такі типи AXI-інтерфейсів:

– AXI Master, через який HPS було ініційовано доступ до ресурсів FPGA;

– AXI Slave, який дозволяє FPGA отримати доступ до пам’яті HPS;

– Lightweight AXI, який було використано для передачі керуючих сигналів і конфігураційних параметрів.

Для передачі великих обсягів даних використано механізм прямого доступу до пам’яті (DMA). DMA-контролери дозволили організувати передачу даних між пам’яттю та FPGA без активної участі процесора, що значно зменшує накладні витрати та підвищує ефективність системи.

Експериментальний сценарій передбачав, що HPS ініціює DMA-передачу, вказуючи адреси джерела та призначення, після чого дані автоматично переміщуються до буферів FPGA або назад у пам’ять після завершення обробки.

Обмін даними між HPS і FPGA організувався за схемою “керування–дані”: керуючі сигнали (запуск задачі, параметри обробки, статус виконання) передаються через AXI-інтерфейс, тоді як основні масиви даних передаються через високопродуктивні AXI-канали з використанням DMA. FPGA, у свою чергу, може сигналізувати про завершення обчислень через механізми переривань , що дозволяє HPS оперативно реагувати та запускати наступні задачі.

Узагальнено, взаємодія між компонентами системи відбувається за таким циклом: процесор формує задачу та розміщує дані в пам’яті, ініціює передачу даних до FPGA через DMA, конфігурує апаратне ядро через AXI-інтерфейс, після чого FPGA виконує обчислення. По завершенні результати повертаються у

пам’ять, генерується сигнал завершення, і процесор виконує подальшу обробку або планує наступні задачі.

4.1.3 Архітектура реалізації методу планування та розподілу задач у гетерогенній системі на базі FPGA

Архітектура реалізації методу планування та розподілу задач у гетерогенній системі на базі FPGA будується як багаторівнева функціональна структура, у якій логіка прийняття рішень відокремлена від механізмів виконання. У її основі лежить послідовність взаємопов’язаних модулів, кожен із яких відповідає за окремий етап обробки задач від їх формалізації до контролю завершення.

Початковим елементом є модуль формування графа задач, який виконує перетворення вхідного набору задач у формалізовану модель у вигляді орієнтованого ациклічного графа. У цьому модулі кожна задача або підзадача представляється вершиною графа, а залежності між ними орієнтованими ребрами. Окрім структури залежностей, модуль формує атрибути вершин, що включають оцінки обчислювальної складності, обсяг вхідних і вихідних даних, допустимі ресурси виконання (CPU, FPGA), а також можливі варіанти реалізації на різних виконавцях. Результатом роботи модуля є повна структурно-параметрична модель задачі, придатна для подальшого аналізу.

Наступним є модуль оцінки ресурсів, який забезпечує кількісну характеристику доступних обчислювальних і комунікаційних ресурсів системи. У цьому модулі формується модель стану ресурсів, що включає завантаженість CPU, доступність апаратних ядер на FPGA, обсяг вільної пам’яті, пропускну здатність каналів передачі даних, а також енергетичні характеристики. Важливим аспектом є оцінка вартості виконання кожної підзадачі на різних ресурсах, яка включає час обчислення, затримки передачі даних та можливі накладні витрати, пов’язані з конфігурацією FPGA. Ці оцінки використовуються як вхідні параметри для планувальника. Архітектура реалізації методу планування та розподілу задач у гетерогенній системі на базі FPGA подана на рисунку 4.2.

Рисунок 4.2 – Архітектура реалізації методу планування та розподілу задач у гетерогенній системі на базі FPGA

Ключовим компонентом архітектури є планувальник, реалізований на основі алгоритму HRLHS або його адаптованої версії. Цей модуль виконує прийняття рішень щодо розподілу задач між доступними ресурсами з урахуванням багатокритеріальної оптимізації. Планувальник поєднує евристичні методи (зокрема, ідеї рою частинок для глобального пошуку оптимального розподілу) та підходи навчання з підкріпленням для адаптації до змінного стану системи. У процесі роботи він аналізує структуру DAG, оцінки ресурсів і поточний стан системи, після чого формує план виконання, який визначає для кожної задачі: виконавця (CPU або FPGA), момент запуску та необхідні ресурси. Важливою особливістю є здатність планувальника враховувати динамічні зміни навантаження та коригувати рішення в процесі виконання.

Модуль диспетчеризації задач відповідає за реалізацію сформованого плану на рівні системи виконання. Його функція полягає у перетворенні абстрактного плану у конкретні дії: постановку задач у черги, ініціацію передачі даних, запуск обчислень на CPU або FPGA. Для задач, призначених на FPGA, диспетчер формує відповідні керуючі сигнали, ініціює DMA-передачу даних і запускає апаратні ядра. Для задач CPU передає їх у відповідні потоки виконання операційної системи. Модуль також відповідає за синхронізацію виконання залежних задач, забезпечуючи дотримання порядку, визначеного графу.

Завершальним елементом є модуль контролю виконання, який здійснює моніторинг стану системи в процесі роботи. Він відстежує статус виконання задач, збирає інформацію про фактичний час виконання, затримки, використання ресурсів і можливі помилки. На основі цих даних модуль може ініціювати повторне планування або активувати механізми динамічної надлишковості, якщо виявлено зниження надійності або перевищення допустимих параметрів. Крім того, цей модуль забезпечує зворотний зв’язок для планувальника, що є критично важливим для адаптивних алгоритмів, зокрема тих, що використовують навчання з підкріпленням.

4.1.4 Апаратна складова FPGA

Апаратна складова FPGA у запропонованій системі формується як сукупність спеціалізованих обчислювальних ядер, які реалізують окремі класи підзадач і інтегруються у спільну обчислювальну інфраструктуру. Кожне апаратне ядро проєктується як самостійний функціональний модуль із чітко визначеним інтерфейсом обміну даними та керування, що дозволяє ефективно інтегрувати його у загальну систему планування та диспетчеризації.

Структурно апаратне ядро складається з кількох основних компонентів: інтерфейсного блоку, обчислювального ядра та підсистеми локальної пам’яті. Інтерфейсний блок забезпечує взаємодію з зовнішнім середовищем через шини AXI або потокові інтерфейси AXI-Stream, приймає вхідні дані, керуючі сигнали та передає результати. Обчислювальне ядро реалізує основну функціональність алгоритмічну обробку даних і є оптимізованим під конкретний тип задач. Підсистема локальної пам’яті на базі BRAM використовується для буферизації даних, зменшення затримок доступу та організації конвеєрної обробки.

Ключовим принципом побудови апаратних ядер є конвеєризація обчислень. Вона передбачає розбиття алгоритму на послідовність стадій, кожна з яких виконує частину обробки даних. Дані проходять через ці стадії як потік, що дозволяє одночасно обробляти кілька елементів на різних етапах конвеєра. Це забезпечує суттєве підвищення пропускної здатності, оскільки після заповнення конвеєра результати можуть генеруватися на кожному такті. Глибина конвеєра визначається складністю алгоритму та допустимими апаратними витратами.

Другим фундаментальним принципом є паралелізм на рівні даних і задач. Паралелізм на рівні даних реалізується шляхом дублювання обчислювальних блоків або використання SIMD-подібних структур, що дозволяє одночасно обробляти кілька елементів даних.

Паралелізм на рівні задач досягається через одночасне функціонування кількох незалежних апаратних ядер, кожне з яких виконує окрему підзадачу. У

контексті гетерогенної системи це дозволяє планувальнику розподіляти задачі між різними ядрами FPGA відповідно до їх доступності та характеристик.

Ефективність реалізації апаратних ядер значною мірою визначається раціональним використанням ресурсів FPGA, зокрема Look-Up Tables (LUT), DSP-блоків та BRAM. LUT використовуються для реалізації комбінаторної логіки, керуючих автоматів і простих арифметичних операцій. Вони забезпечують гнучкість реалізації довільних логічних функцій, але їх надмірне використання може обмежувати масштабованість системи.

DSP-блоки є спеціалізованими апаратними ресурсами, призначеними для виконання арифметичних операцій, зокрема множення та операцій типу множення з накопиченням. Використання DSP дозволяє значно підвищити продуктивність обчислювальних ядер, особливо для задач цифрової обробки сигналів, лінійної алгебри або машинного навчання. Оптимальне розміщення обчислювальних операцій у DSP-блоках є критичним для досягнення високої продуктивності при мінімальних витратах LUT.

BRAM використовується як швидка локальна пам’ять для зберігання проміжних результатів, буферизації потоків даних та реалізації черг. Завдяки низькій затримці доступу BRAM є ключовим елементом для підтримки конвеєризації та безперервного потоку даних.

Крім того, ефективна організація пам’яті дозволяє зменшити кількість звернень до зовнішньої пам’яті, що є вузьким місцем у багатьох системах.

Узгоджене використання LUT, DSP та BRAM дозволяє досягти балансу між продуктивністю, ресурсною ефективністю та енергоспоживанням. При цьому важливим є врахування обмежень конкретної FPGA-платформи Cyclone V, що визначає максимальну кількість доступних ресурсів і впливає на кількість та складність реалізованих апаратних ядер.

4.1.5 Взаємодія обчислень і передачі даних

Взаємодія обчислень і передачі даних у запропонованій гетерогенній системі організована як узгоджений конвеєр «підготовка даних – передача – обробка – повернення результатів», у якому ключову роль відіграють механізми буферизації, черг задач, синхронізації та керування потоками даних. Такий підхід дозволяє мінімізувати простої обчислювальних ресурсів і забезпечити безперервність обробки.

Передача даних між процесорною підсистемою та FPGA здійснюється з використанням багаторівневої буферизації. На рівні оперативної пам’яті HPS формуються вхідні та вихідні буфери, які містять дані задач. Ці буфери організовані за принципом кільцевих або подвійних, що дозволяє одночасно виконувати підготовку нових даних і обробку попередніх. На стороні FPGA використовуються локальні буфери на базі BRAM, які приймають дані через DMA та забезпечують їх подальшу передачу до обчислювальних ядер у потоковому режимі. Така ієрархія буферів зменшує затримки доступу до даних і дозволяє узгодити швидкість передачі та обробки.

Черги задач реалізуються як на рівні програмного забезпечення, так і на апаратному рівні. У HPS формується глобальна черга задач, яка містить елементи DAG, готові до виконання з урахуванням залежностей. Планувальник формує впорядковану послідовність задач і передає її до модуля диспетчеризації. Далі задачі розподіляються у локальні черги: для CPU у вигляді потоків або процесів операційної системи, для FPGA у вигляді командних дескрипторів, що обробляються апаратним планувальником або контролером запуску ядер. На FPGA також можуть реалізовуватися FIFO-черги для подачі даних у конвеєри обчислювальних ядер.

Механізми синхронізації забезпечують коректне узгодження між етапами обробки та між різними ресурсами системи. На рівні HPS використовуються програмні засоби синхронізації (семафори, події, блокування), які координують доступ до спільних ресурсів і відстежують завершення задач. На рівні FPGA

застосовуються апаратні сигнали готовності, механізми зворотного тиску у потокових інтерфейсах та переривання (interrupts), що сигналізують процесору про завершення обчислень або виникнення помилок. Взаємодія між HPS і FPGA також включає синхронізацію через регістри керування, доступні через AXI-інтерфейс.

Управління потоками даних реалізується за принципом потокової обробки (stream processing), де дані передаються між модулями без необхідності повного збереження в пам’яті. Це досягається завдяки використанню AXI-Stream інтерфейсів і конвеєрної організації обчислювальних ядер. Керування потоками включає регулювання швидкості подачі даних, балансування навантаження між ядрами та уникнення переповнення буферів. Планувальник може враховувати пропускну здатність окремих каналів і динамічно змінювати порядок виконання задач для оптимізації використання ресурсів.

З позиції підтримки динамічної надлишковості архітектура передбачає механізми резервування задач і дублювання обчислень. На рівні планувальника визначаються задачі з підвищеним ризиком через нестабільність ресурсу або критичність результату, для яких створюються резервні копії. Ці копії можуть бути призначені як на інший ресурс (наприклад, дублювання виконання на CPU і FPGA, так і на той самий ресурс із часовим зсувом.

У системі реалізовано подвійну або адаптивну чергу виконання, де основна задача і її резервна версія розглядаються як пов’язані елементи. Буферизація даних організована таким чином, щоб забезпечити можливість повторного використання вхідних даних без додаткових витрат на передачу. Модуль контролю виконання відстежує результати обох виконань і виконує процедуру верифікації: у разі успішного завершення основної задачі резервна може бути скасована або проігнорована, тоді як у випадку помилки або перевищення часу виконання активується резервний результат.

Механізми синхронізації також розширюються для підтримки надлишковості: вводяться додаткові сигнали стану задач, а також логіка узгодження результатів. Управління потоками даних у цьому контексті враховує можливість дублювання потоків і необхідність їх узгодження на виході системи.

4.2 Розробка системного програмного забезпечення

Також було реалізовано архітектуру програмно-технічного засобу реалізації методу планування побудована як ієрархічна апаратно-орієнтована система, у якій функціональні модулі чітко відокремлено за етапами обробки задач, а вся логіка реалізована у вигляді синхронних процесів, що працюють під керуванням єдиного тактового сигналу.

Лістинг системного програмного забезпечення реалізації системи подано у додатку А.

Базовим рівнем архітектури виступає пакет типів scheduler_pkg, у якому сформовано формальну модель системи. У цьому модулі було задано параметри розмірності системи, типи даних для представлення задач, процесорів, часових і енергетичних характеристик, а також матричні структури для опису DAG-графа, часу виконання та комунікаційних затримок. Було реалізовано структуру task_record_t, яка інкапсулює повний стан задачі, включаючи її життєвий цикл, призначення на ресурс, часові параметри, пріоритет та ознаку резервування. Таким чином було сформовано уніфіковане подання задачі, яке використано всіма іншими модулями системи. На рівні ініціалізації було реалізовано модуль dag_init, у якому здійснено формування графової моделі задач. У цьому модулі було завантажено матрицю часу виконання задач на різних процесорах, матрицю суміжності, що визначає залежності між задачами, а також матрицю комунікаційних затримок. Було реалізовано синхронний процес, який при активації сигналу reset переводив систему у початковий стан, а при надходженні тактового імпульсу здійснював ініціалізацію всіх параметрів. У результаті було сформовано повністю визначену модель DAG, яка використовується планувальником.

Архітектуру оцінювання параметрів було реалізовано через окремі логічні блоки, які обчислюють характеристики задач, зокрема надійність і енергоспоживання. Було закладено модель надійності у вигляді експоненціальної функції, що залежить від часу виконання задачі, а також модель енергоспоживання, яка враховує динамічну та статичну складові. Обчислення цих параметрів було

організовано як апаратні операції над фіксованою розрядністю, що дозволило інтегрувати їх безпосередньо у процес планування.

Центральним елементом архітектури було реалізовано планувальник, який працює з використанням пріоритетних черг. Було організовано механізм аналізу стану задач, у якому перевірено виконання залежностей на основі матриці суміжності. Задачі, для яких виконано всі попередники, було переведено у стан TASK_READY. Далі було виконано сортування задач за пріоритетами, після чого здійснено призначення задач на доступні процесори з урахуванням їх стану. Було реалізовано логіку переходів станів задач (від TASK_IDLE до TASK_DONE або TASK_FAILED), що дозволило відстежувати повний життєвий цикл кожної задачі.

Було також реалізовано модуль диспетчеризації, який здійснив безпосереднє призначення задач на обчислювальні ресурси. У цьому модулі було виконано перевірку стану процесорів, після чого задачі зі станом TASK_ASSIGNED було переведено у стан TASK_RUNNING. Було забезпечено запис часу початку виконання та подальший контроль завершення задачі. У випадку завершення було встановлено стан TASK_DONE, а у випадку відмови TASK_FAILED.

Також було реалізовано механізм динамічної надлишковості. Було введено окремий стан TASK_BACKUP та відповідні поля у структурі задачі, які дозволили ідентифікувати резервні копії. Було реалізовано механізм створення резервних задач для критичних або ненадійних обчислень. У разі виявлення відмови основної задачі було активовано резервну копію, що дозволило забезпечити підвищення відмовостійкості системи. Було також передбачено можливість паралельного виконання основної та резервної задачі з подальшим вибором коректного результату.

4.3 Експерименти

Для вирішення задачі оптимізації продуктивності обчислювальних систем в частині зменшення енергоспоживання та підвищення швидкодії системи за рахунок динамічної надлишковості було проведено ряд експериментів.

Цей підрозділ в основному поділено на три частини: порівняння з традиційними алгоритмами, порівняння зі статичним плануванням та порівняльний аналіз з алгоритмами, запропонованими в статтях та.

Серед них, експеримент порівняння з традиційними алгоритмами використовує два еталонні робочі процеси, які часто використовуються для оцінки продуктивності планування: робочий процес гаусового виключення (GE) та робочий процес швидкого перетворення Фур'є (FFT), як показано на рис. 5. Гетерогенний паралелізм процесорів представлений генерацією випадкових параметрів.

В дослідженні було проведено статистичний аналіз результатів оцінювання, щоб забезпечити надійність наших висновків. Було проведено кілька експериментів на обох робочих процесах, а відповідні дані продуктивності були зібрані в коробкові діаграми, як показано на рис. 4.3.

Діаграми пропонують представлення розподілу та дисперсії показників продуктивності за різними сценаріями. Щоб оцінити статистичну значущість нашого запропонованого методу, було застосовано дисперсійний аналіз (ANOVA). Цей метод дозволив нам порівняти середні значення показників продуктивності за різних умов. Результати перевірки гіпотези показали, що p-значення для всіх порівнянь були нижчими за поріг значущості 0,05, що підтверджує, що спостережувані відмінності були статистично значущими та не зумовлені випадковістю. Ця ретельна статистична перевірка не лише підвищує надійність наших результатів, але й додатково демонструє ефективність запропонованого методу. На рисунку 4.4 порівнюється алгоритм на основі змішаного цілочисельного нелінійного програмування з навчанням з підкріпленням, глибоким навчанням з підкріпленням, оптимізацією рою частинок та жадібними алгоритмами з точки зору споживання енергії та часу відгуку в двох різних робочих процесах.

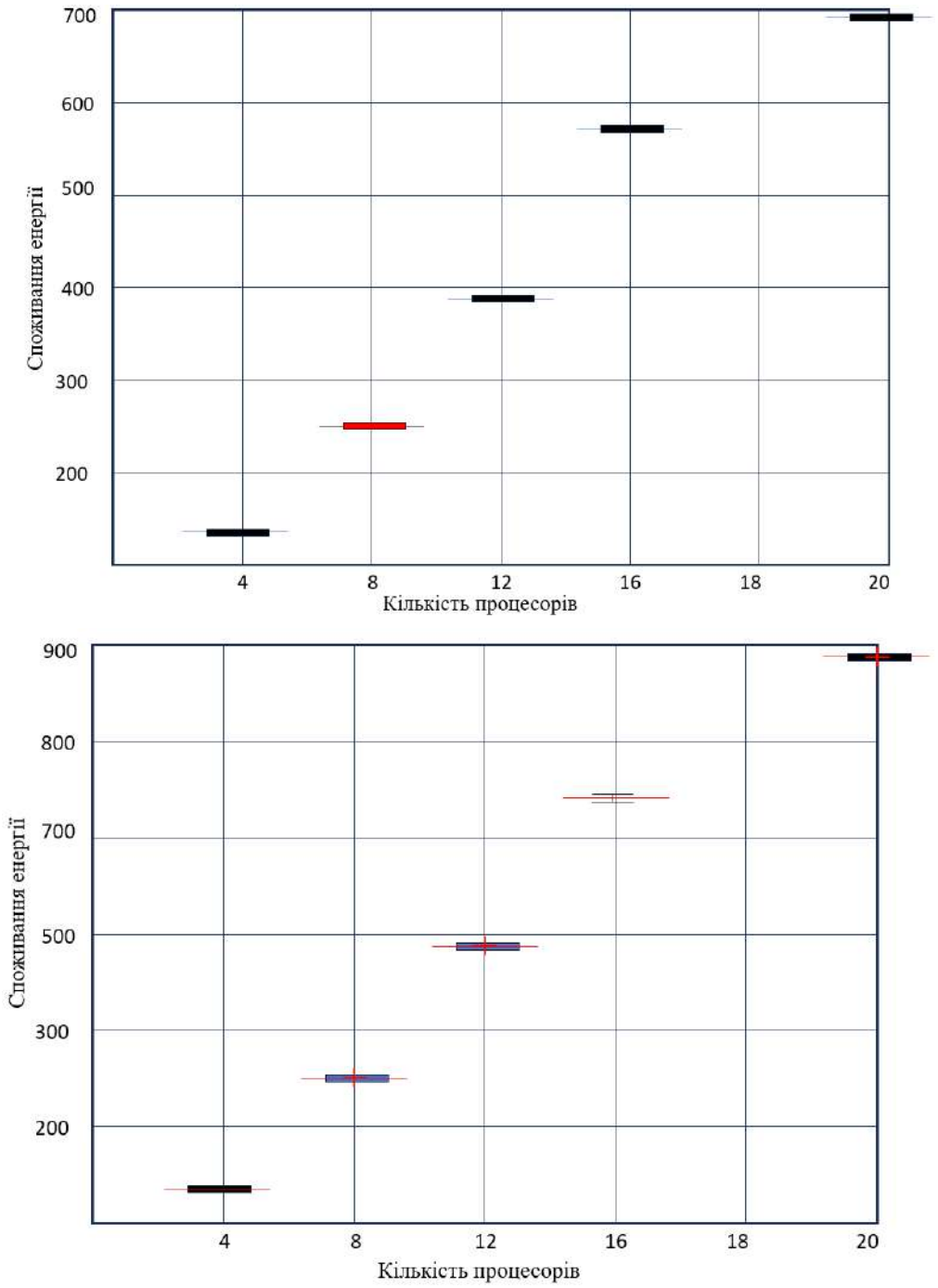
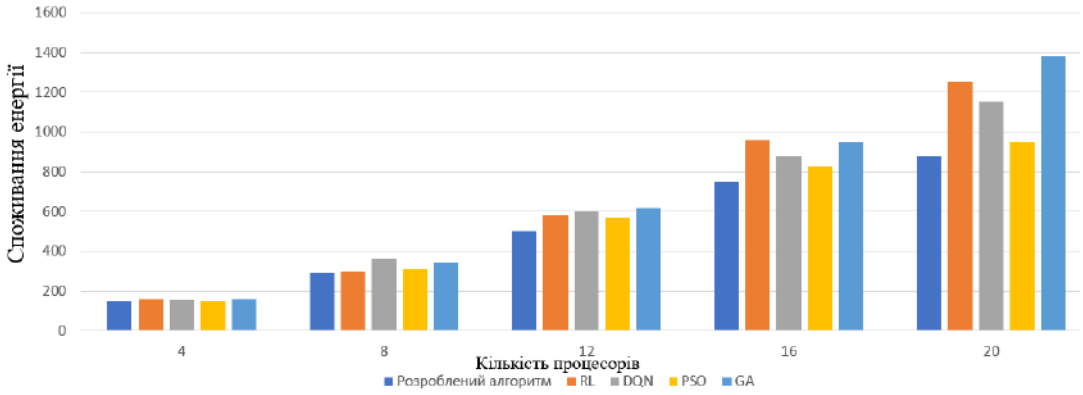
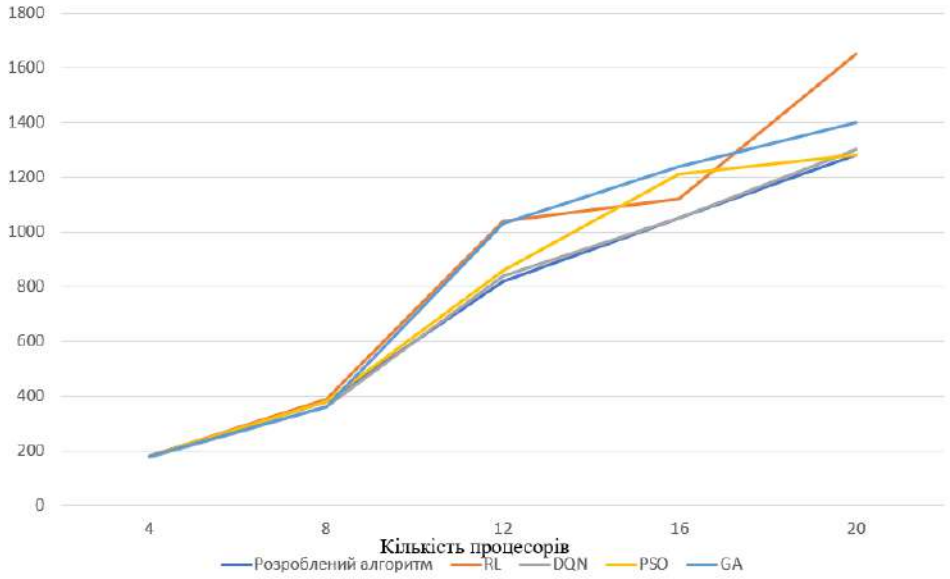
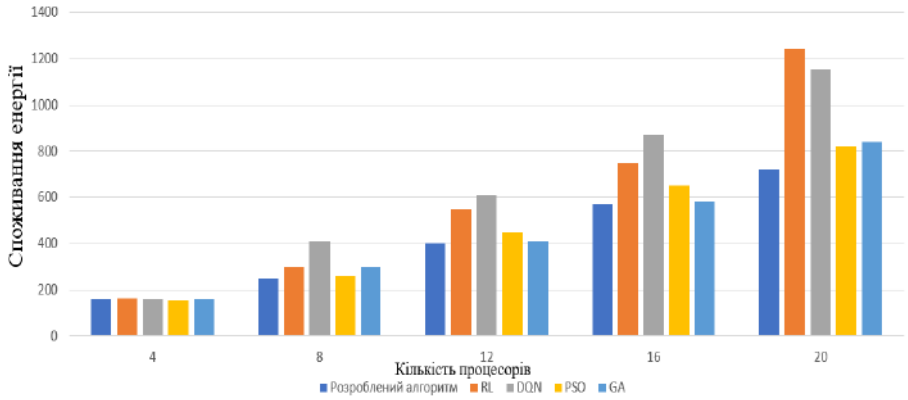


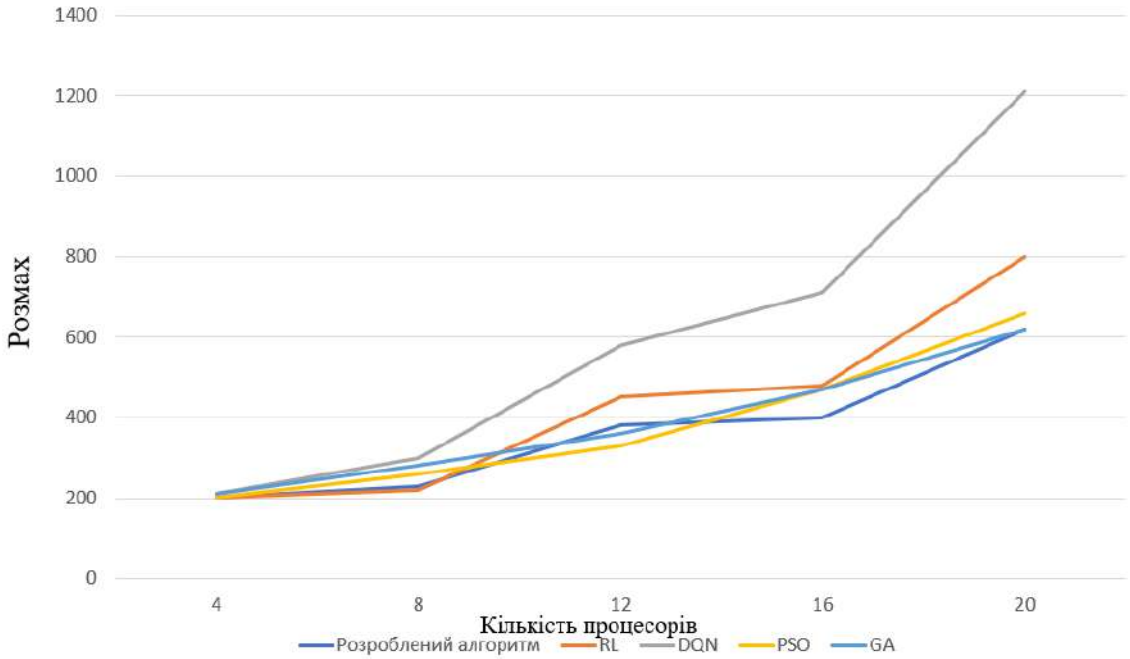
Рисунок 4.3 – Статистичний аналіз споживання енергії для двох робочих процесів


Споживання енергії
1600
1400
1200
1000
800
600
400
200
0
4
8
12
16
20
Кількість процесорів
Розроблений алгоритм
RL
DQN
PSO
GA


1800
1600
1400
1200
1000
800
600
400
200
0
4
8
12
16
20
Кількість процесорів
Розроблений алгоритм
RL
DQN
PSO
GA

а)


Споживання енергії
1400
1200
1000
800
600
400
200
0
4
8
12
16
20
Кількість процесорів
Розроблений алгоритм
RL
DQN
PSO
GA



б)

Рисунок 4.4 – Порівняння споживання енергії та часу відгуку між запропонованим методом та традиційним методом за двох робочих процесів: а) робочий процес швидкого перетворення Фур'є»; б) робочий процес обробки графа

Кількість гетерогенних паралельних процесорів збільшується від 4 до 20. Можна спостерігати, що зі збільшенням кількості процесорів споживання енергії та час відгуку більшості методів лінійно зростають, причому алгоритм HRLHS постійно підтримує найнижче споживання енергії та прийнятний час відгуку протягом кількох раундів планування в обох робочих процесах.

Окрім запропонованого методу, алгоритм PSO працює відносно добре, оскільки він може уникнути локальних оптимумів та виконати глобальний пошук.

На противагу цьому, навчання з підкріпленням та глибоке навчання з підкріпленням, як правило, розміщують більше завдань на одному процесорі, щоб зменшити затримки зв'язку, що може призвести до більшого споживання енергії. Жадібний алгоритм, надаючи пріоритет процесорам з нижчим споживанням енергії

для кожного завдання, нехтує залежностями завдань, що призводить до того, що майже всі процесори залишаються активними, що збільшує загальне споживання енергії. Більше того, процесори, які не використовуються, не сприятимуть статичному споживанню енергії, якщо їх вимкнути.

Динамічне планування може коригувати ресурси на основі станів системи в режимі реального часу, тоді як статичне планування не має гнучкості та вимагає попередньо спланованих призначень завдань. Динамічне планування може адаптуватися до раптових змін, перерозподіляючи ресурси відповідно до терміновості, пріоритету, залежностей завдань та поточного стану системи, тим самим оптимізуючи час виконання завдань або споживання енергії. Це дає динамічному плануванню явну перевагу.

На рисунку 4.5 цей метод порівнюється з двома класичними алгоритмами статичного планування. Метод пріоритетного планування (PS) визначає порядок планування на основі пріоритету споживання енергії завданнями, тоді як метод «найкоротше завдання спочатку» (SJF) зменшує динамічне споживання енергії, мінімізуючи час виконання завдань.

Обидва алгоритми визначають вибір процесора та порядок завдань по черзі під час планування, враховуючи лише залежність між завданнями-попередниками та наступниками. Вони повністю ігнорують динамічні взаємодії між завданнями робочого процесу, тому результати їх планування не відповідають вимогам.

На рисунку 4.6 представлено порівняння споживання енергії між запропонованим алгоритмом планування та трьома існуючими алгоритмами: алгоритмом об'єднання процесорів та алгоритмом мінімізації тривалості планування з урахуванням вимог до надійності (MSLSRR), а також алгоритмом адаптивного Q-навчання. Експериментальні результати показують, що запропонований алгоритм має значні переваги в оптимізації споживання енергії. Основна ідея алгоритму PM полягає в мінімізації кількості активних процесорів, тим самим зменшуючи статичне споживання енергії системою. Це досягається шляхом об'єднання процесорів для зменшення кількості активних процесорів.

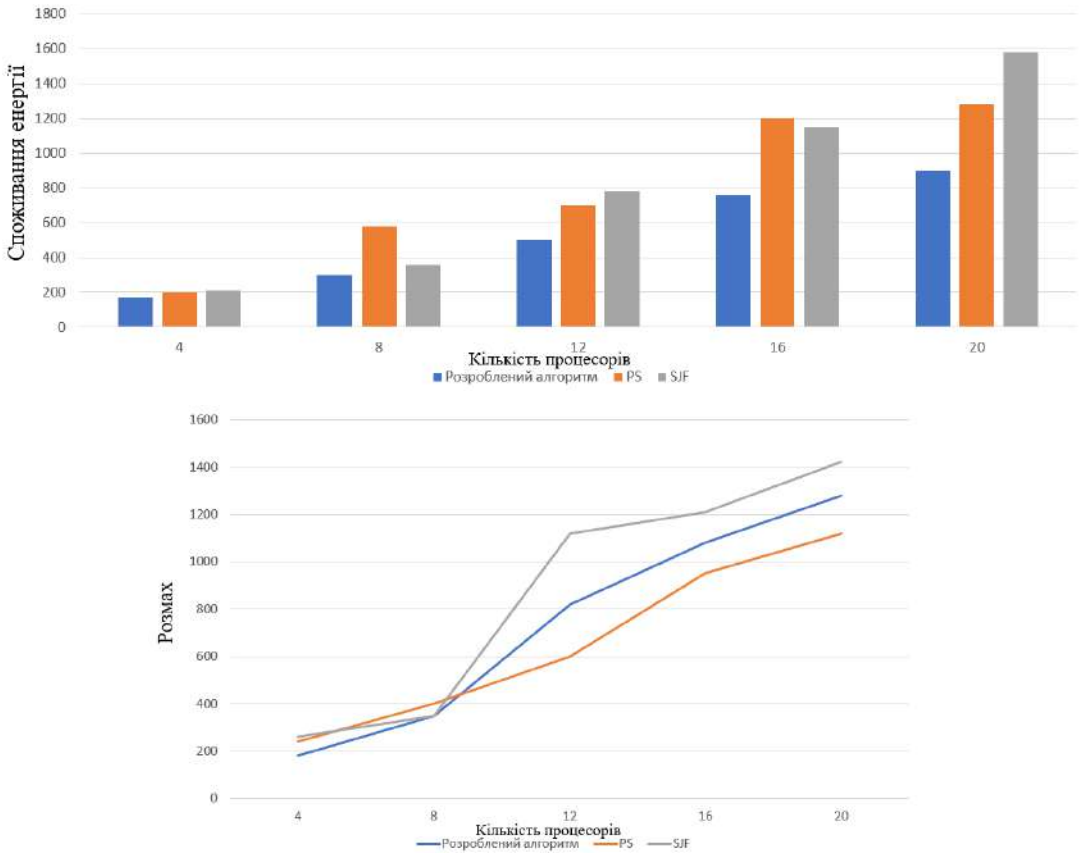


Рисунок 4.5 – Порівняння споживання енергії та затримки між динамічним та статичним методами планування

Однак, зменшуючи кількість процесорів, алгоритм PM не враховує ефективність виконання завдань, що призводить до неоптимальної оптимізації динамічного споживання енергії.

Алгоритм MSLSRR зосереджується на мінімізації тривалості планування, одночасно задовольняючи вимоги до надійності системи для економії енергії. Він балансує час виконання завдань та надійність системи під час планування, прагнучи зменшити загальне споживання енергії за рахунок ефективного планування завдань.

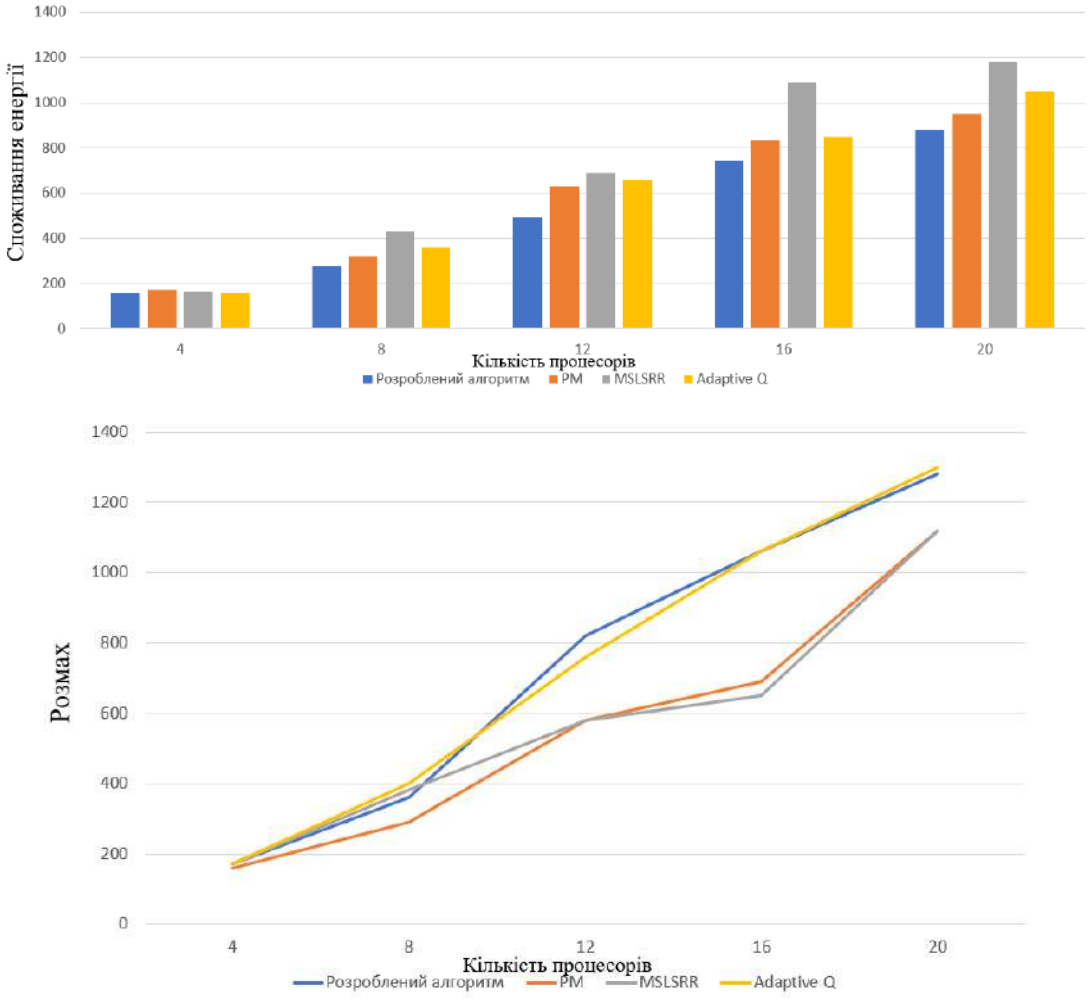


Рисунок 4.6 – Порівняння споживання енергії та затримки між нашим та трьома вдосконаленими методами

Однак основна увага цього алгоритму зосереджена на мінімізації тривалості планування з обмеженою оптимізацією динамічного споживання енергії. Адаптивний алгоритм Q-навчання дозволяє адаптивне управління ресурсами в складних обчислювальних середовищах, але на його продуктивність може впливати вибір представлень станів та дій, і може вимагати значного часу для збіжності в високодинамічних середовищах.

Порівняно з цими алгоритмами, запропонований метод враховує як статичне, так і динамічне споживання енергії та включає різні стратегії оптимізації під час

процесу планування завдань. Це гарантує мінімізацію споживання енергії, одночасно задовольняючи залежності завдань та балансуючи навантаження процесора.

Експериментальні результати показують, що запропонований алгоритм зменшує споживання енергії на 14,3% порівняно з алгоритмом PM, на 45,1% порівняно з алгоритмом MSLSRR та на 26,8% порівняно з іншим алгоритмом, демонструючи значне покращення продуктивності енергозбереження.

Експериментальні результати показують, що запропонований метод добре працює та досягає чудових результатів у вирішенні проблем споживання енергії.

4.6 Висновки

У четвертому розділі магістерської роботи розроблено та описано програмно-технічний засіб реалізації методу планування та розподілу задач у гетерогенних обчислювальних системах на базі програмованих логічних інтегральних схем, проведено його експериментальну перевірку та порівняльний аналіз ефективності.

Обґрунтовано архітектурний підхід на основі гетерогенної платформи типу «система-на-кристалі» DE10-Standard із SoC Altera Cyclone V, яка поєднує процесорну підсистему ARM (HPS) та програмовану логіку (FPGA). Показано, що такий підхід дозволяє раціонально розподілити функції між компонентами: процесорна підсистема відповідає за глобальне керування, побудову графової моделі задач та виконання алгоритму планування, тоді як FPGA виконує роль високопродуктивного обчислювального прискорювача для задач із регулярною структурою та високим ступенем паралелізму. Взаємодія між доменами реалізована через інтерфейси AXI та механізм прямого доступу до пам'яті, що забезпечує мінімізацію затримок обміну даними та зниження навантаження на процесор.

Розроблено багаторівневу функціональну архітектуру системи, що включає п'ять взаємопов'язаних модулів: формування графа задач, оцінки ресурсів, планувальника на основі алгоритму HRLHS, диспетчеризації задач та контролю виконання. Апаратна складова FPGA побудована на принципах глибокої конвеєризації та паралелізму на рівні даних і задач з використанням ресурсів LUT, DSP-блоків та BRAM. Це забезпечує суттєве підвищення пропускної здатності при мінімальних енергетичних витратах. Взаємодія обчислень і передачі даних організована як узгоджений конвеєр із багаторівневою буферизацією, апаратними чергами типу FIFO та механізмами синхронізації через сигнали переривань і регістри керування.

Системне програмне забезпечення реалізовано у вигляді ієрархічної апаратно-орієнтованої структури синхронних модулів. Базовим рівнем є пакет типів, що формалізує модель системи через структуру повного стану задачі з відстеженням її життєвого циклу. Реалізовано механізм динамічної надлишковості з окремим станом резервних задач, що дозволяє створювати резервні копії для критичних обчислень, активувати їх у разі відмови основної задачі та здійснювати паралельне виконання з подальшою верифікацією результатів.

Проведено три серії порівняльних експериментів на еталонних робочих процесах гаусового виключення та швидкого перетворення Фур'є. Статистична значущість результатів підтверджена дисперсійним аналізом ANOVA із рівнем значущості $p < 0,05$. Встановлено, що алгоритм HRLHS стабільно забезпечує найнижче енергоспоживання та прийнятний час відгуку при збільшенні кількості процесорів від 4 до 20. Порівняно зі статичними методами планування підтверджено переваги динамічного підходу завдяки здатності адаптуватися до змін стану системи в режимі реального часу. У порівнянні з алгоритмами об'єднання процесорів, мінімізації тривалості планування з урахуванням надійності та адаптивного Q-навчання запропонований метод забезпечив зменшення енергоспоживання на 14,3%, 45,1% та 26,8% відповідно, що підтверджує його практичну ефективність для застосування у реальних гетерогенних обчислювальних середовищах.

ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень проаналізовано відомі методи оптимізації продуктивності обчислювальних систем; розроблено метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA; здійснено дослідження методу планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA; реалізовано метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA у вигляді системи. Проведені експерименти на еталонних робочих процесах гаусового виключення та швидкого перетворення Фур'є, показали, що запропонований метод забезпечує нижче енергоспоживання при збереженні прийнятного часу відгуку в діапазоні 4–20 процесорів. Підтверджено перевагу динамічного планування над статичними підходами. Досягнуто зниження енергоспоживання на 14,3%, 45,1% та 26,8% порівняно з відомими методами, що доводить ефективність запропонованого методу.

За результатами досліджень отримано наукову новизну:

- удосконалено метод енергоефективного планування задач у гетерогенних паралельних системах, який на відміну від відомих поєднує евристичну оптимізацію та навчання з підкріпленням із урахуванням залежностей задач і неоднорідності ресурсів, і який дає змогу зменшити енергоспоживання та підвищити швидкодію системи за рахунок динамічної надлишковості;
- набула подальшого розвитку система планування задач у гетерогенних паралельних системах на основі FPGA.

Практична значимість отриманих результатів полягає у розробленні системи, яка здійснює оптимізацію планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA.

За темою кваліфікаційної роботи опубліковано публікацію [112].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Ghorbian, M., Ghobaei-Arani, M., Esmaeili, L. *A survey on the scheduling mechanisms in serverless computing: a taxonomy, challenges, and trends.* Cluster Computing (2024). DOI: 10.1007/s10586-023-04264-8.

2. Aslani, A., Ghobaei-Arani, M. *Machine learning inference serving models in serverless computing: a survey.* Computing 107 (2025). DOI: 10.1007/s00607-024-01377-9.

3. Xu, X., Tian, Q., Xing, Y., Yin, B., Hu, A. *Large-scale data intensive heterogeneous task scheduling method based on parallel GATS-TS algorithm.* CISCE (2022). DOI: 10.1109/CISCE55963.2022.9851157.

4. Li, P., Xiao, Y., Yan, J., Li, X., Wang, X. *Reinforcement learning for adaptive resource scheduling in complex system environments.* ISCEIC (2024). DOI: 10.1109/ISCEIC63613.2024.10810157.

5. Liao, Z., Peng, J., Xiong, B., Huang, J. *Adaptive offloading in mobile-edge computing for ultra-dense cellular networks based on genetic algorithm.* Journal of Cloud Computing 10 (2021). DOI: 10.1186/s13677-021-00232-y.

6. Hu, S., Li, G. *Dynamic request scheduling optimization in mobile edge computing for IoT applications.* IEEE Internet of Things Journal 7 (2019). DOI: 10.1109/JIOT.2019.2955311.

7. Tuli, S., Ilager, S., Ramamohanarao, K., Buyya, R. *Dynamic scheduling for stochastic edge-cloud computing environments using A3C learning and residual recurrent neural networks.* IEEE TMC (публікація доступна як препринт, 2020). DOI в записі користувача: 10.1109/TMC.2020.3017079.

8. Mahesar, A. R., Xiaoping, L., Sajnani, D. K., Rajput, K. Y. *Efficient workflow scheduling and cost optimization for deadline-constrained microservice applications in mobile edge computing.* CSCWD (2024). DOI: 10.1109/CSCWD61410.2024.10580475.

9. Xia, L. et al. *Optimal load scheduling based on mobile edge computing technology in 5G dense networking.* ACCC (2022). DOI: 10.1109/ACCC58361.2022.00030.

10. Dai, H., Liu, S., Liu, B., Fan, Z., Wang, J. *Technical middleware microservice orchestration and fault-tolerant mechanism algorithms for containerized deployment.* ICCASIT (2024). DOI: 10.1109/ICCASIT62299.2024.10828011.

11. Long, T., Xia, Y., Ma, Y., Peng, Q., Zhao, J. *A fault-tolerant workflow scheduling method on deep reinforcement learning-based in edge environment.* ICNSC (2022). DOI: 10.1109/ICNSC55942.2022.10004189.

12. Yao, G., Ren, Q., Li, X., Zhao, S., Ruiz, R. *A hybrid fault-tolerant scheduling for deadline-constrained tasks in cloud systems.* IEEE TSC 15 (2020/2022). DOI: 10.1109/TSC.2020.2992928.

13. Yin, C., Shi, X. *Fault-tolerant scheduling optimization of cloud workflow based on multi-objective optimization algorithm.* CAC (2023). DOI: 10.1109/CAC59555.2023.10450946.

14. Chawla, S., Kaur, A. *Fault-tolerant heuristic task scheduling algorithm for efficient resource utilization in cloud computing.* AUTOCOM (2024). DOI: 10.1109/AUTOCOM60220.2024.10486076.

15. Wang L, Ma C, Feng X, Zhang Z, Yang H, Zhang J, Chen Z, Tang J, Chen X, Lin Y, et al. A survey on large language model based autonomous agents. Front Comput Sci. 2024;18: Article 186345.

16. Cheng Y, Zhang C, Zhang Z, Meng X, Hong S, Li W, Wang Z, Wang Z, Yin F, Zhao J, et al. Exploring large language model based intelligent agents: Definitions, methods, and prospects. arXiv. 2024. https://doi.org/10.48550/arXiv.2401.03428

17. Xu X, Wang Y, Xu C, Ding Z, Jiang J, Ding Z, Karlsson BF. A survey on game playing agents and large models: Methods, applications, and challenges. CoRR. arXiv. 2024. https://doi.org/10.48550/arXiv. 2403.10249

18. Huang X, Liu W, Chen X, Wang X, Hang H, Lian D, Wang Y, Tang R, Chen E, et al. Understanding the planning of LLM agents: A survey. arXiv. 2024. https://doi.org/10.48550/arXiv.2402.02716

19. Zhang Y, Mao S, Ge T, Wang X, de Wynter A, Xia Y, Wu W, Song T, Lan M, Wei F, et al. LLM as a mastermind: A survey of strategic reasoning with large language models. arXiv. 2024. https://doi.org/10.48550/arXiv.2404.01230

20. Pallagani V, Muppasani BC, Roy K, Fabiano F, Loreggia A, Murugesan K, Srivastava B, Rossi F, Horesh L, Sheth AP. On the prospects of incorporating large language models (LLMs) in automated planning and scheduling (APS). Paper presented at: Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2024; 2024 Jun 1–6; Banff, Alberta, Canada.

21. Wei J, Wang X, Schuurmans D, Bosma M, Ichter B, Xia F, Chi E, Le Q, Zhou D. Chain-of-thought prompting elicits reasoning in large language models. Paper presented at: Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022; 2022 Nov 28–Dec 9; New Orleans, LA, USA.

22. Fu Y, Peng H, Sabharwal A, Clark P, Khot T. Complexity-based prompting for multistep reasoning. Paper presented at: The Eleventh International Conference on Learning Representations, ICLR 2023; 2023 May 1–5; Kigali, Rwanda.

23. Kojima T, Gu SS, Reid M, Matsuo Y, Iwasawa Y. Large language models are zero-shot reasoners. Paper presented at: Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022; 2022 Nov 28–Dec 9; New Orleans, LA, USA.

24. Zhang Z, Zhang A, Li M, Smola A. Automatic chain of thought prompting in large language models. Paper presented at: The Eleventh International Conference on Learning Representations, ICLR 2023; 2023 May 1–5; Kigali, Rwanda.

25. Wang L, Xu W, Lan Y, Hu Z, Lan Y, Lee RKW, Lim EP. Plan-and-Solve Prompting: Improving zero-shot chain-of-thought reasoning by large language models. Paper presented at: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023; 2023 Jul 9–14; Toronto, Canada.

26. Yao S, Yu D, Zhao J, Shafran I, Griffiths T, Cao Y, Narasimhan K. Tree of thoughts: Deliberate problem solving with large language models. Paper presented at:

Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023; 2023 Dec 10–16; New Orleans, LA, USA.

27. Besta M, Blach N, Kubicek A, Gerstenberger R, Podstawski M, Gianinazzi L, Gajda J, Lehmann T, Niewiadomski H, Nyczyk P, et al. Graph of thoughts: Solving elaborate problems with large language models. AAAI Conf Artif Intell. 2024;38(16):17682–17690.

28. Yao S, Zhao J, Yu D, Du N, Shafran I, Narasimhan KR, Cao Y. ReAct: Synergizing reasoning and acting in language models. Paper presented at: The Eleventh International Conference on Learning Representations, ICLR 2023; 2023 May 1–5; Kigali, Rwanda.

29. Qin Y, Liang S, Ye Y, Zhu K, Yan L, Lu Y, Lin Y, Cong X, Tang X, Qian B, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. Paper presented at: The Twelfth International Conference on Learning Representations, ICLR 2024; 2024 May 7–11; Vienna, Austria. OpenReview.net, 2024. url: https://openreview.net/forum?id=dHng2O0Jjr

30. Ye Y, Cong X, Tian S, Qin Y, Liu C, Lin Y, Liu Z, Sun M. Rational decision-making agent with internalized utility judgment. arXiv. 2024. https://doi.org/10.48550/arXiv.2308.12519

31. Hou X, Yang M, Jiao W, Wang X, Tu Z, Zhao WX. CoAct: A global-local hierarchy for autonomous agent collaboration. arXiv. 2024. https://doi.org/10.48550/arXiv.2406.13381

32. Zhou D, Schärli N, Hou L, Wei J, Scales N, Wang X, Schuurmans D, Cui C, Bousquet O, Le QV, et al. Least-to-most prompting enables complex reasoning in large language models. Paper presented at: The Eleventh International Conference on Learning Representations, ICLR 2023; 2023 May 1–5; Kigali, Rwanda.

33. Team K, Du A, Gao B, Xing B, Jiang C, Chen C, Li C, Xiao C, Du C, Liao C, et al. Kimi k1.5: Scaling reinforcement learning with LLMs. arXiv. 2025. https://doi.org/10.48550/arXiv. 2501.12599

34. DeepSeek-AI, Guo D, Yang D, Zhang H, Song J, Zhang R, Xu R, Zhu Q, Ma S, Wang P, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. arXiv. 2025. https://doi.org/10.48550/arXiv.2501.12948

35. El-Kishky A, Wei A, Saraiva A, Minaiev B, Selsam D, Dohan D, Song F, Lightman H, Ignasi C, Pachocki J, et al. Competitive programming with large reasoning models. arXiv. 2025. https://doi.org/10.48550/arXiv.2502.06807

36. Wang X, Wei J, Schuurmans D, Le QV, Chi EH, Narang S, Chowdhery A, Zhou D. Self-consistency improves chain of thought reasoning in language models. Paper presented at: The Eleventh International Conference on Learning Representations, ICLR 2023; 2023 May 1–5; Kigali, Rwanda. OpenReview.net, 2023.

37. Zhang C, Liu L, Wang C, Sun X, Wang H, Wang J, Cai M. PREFER: Prompt ensemble learning via feedback-reflectrefine. Proc AAAI Conf Artif Intell. 2024;38(17):19525–19532.

38. Huang J, Gu S, Hou L, Wu Y, Wang X, Yu H, Han J. Large language models can self-improve. Paper presented at: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023; 2023 Dec 6–10; Singapore.

39. Madaan A, Tandon N, Gupta P, Gupta P, Hallinan S, Gao L, Wiegreffe S, Alon U, Dziri N, Prabhumoye S, Yang S, et al. Self-refine: Iterative refinement with self-feedback. Paper presented at: Advances in Neural Information Processing Systems 36: Annual Conference on Neual Information Processing Systems 2023, NeurIPS 2023; 2023 Dec 10–16; New Orleans, LA, USA.

40. Zhang W, Shen Y, Wu L, Peng Q, Wang J, Zhuang Y, Lu W. Self-contrast: Better reflection through inconsistent solving perspectives. Paper presented at: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024; 2024 Aug 11–16; Bangkok, Thailand.

41. Gou Z, Shao Z, Gong Y, Shen Y, Yang Y, Duan N, Chen W. CRITIC: Large language models can self-correct with toolinteractive critiquing. Paper presented at: The Twelfth International Conference on Learning Representations, ICLR 2024; 2024 May 7–11; Vienna, Austria.

42. Wang G, Xie Y, Jiang Y, Mandlekar A, Xiao C, Zhu Y, Fan L, Anandkumar A. Voyager: An open-ended embodied agent with large language models. arXiv. 2023. https://doi.org/10.48550/arXiv.2305.16291

43. Tan W, Ding Z, Zhang W, Li B, ZhouB, Yue J, Xia H, Jiang J, Zheng L, Xu X, et al. Towards general computer control: A multimodal agent for red dead redemption II as a case study. arXiv. 2024. https://doi.org/10.48550/arXiv.2403.03186

44. Shinn N, Cassano F, Gopinath A, Narasimhan K, Yao S. Reflexion: Language agents with verbal reinforcement learning. Paper presented at: Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023; 2023 Dec 10–16; New Orleans, LA, USA.

45. Lightman H, Kosaraju V, Burda Y, Edwards H, Baker B, Lee T, Leike J, Schulman J, Sutskever I, Cobbe K. Let's verify step by step. Paper presented at: The Twelfth International Conference on Learning Representations, ICLR 2024; 2024 May 7–11; Vienna, Austria.

46. Feng Y, Wang Y, Liu J, Zheng S, Lu Z. LLaMA-Rider: Spurring large language models to explore the open world. Paper presented at: Findings of the Association for Computational Linguistics: NAACL 2024; 2024 Jun 16–21; Mexico City, Mexico.

47. Liu H, Sferrazza C, Abbeel P. Chain of hindsight aligns language models with feedback. Paper presented at: The Twelfth International Conference on Learning Representations, ICLR 2024; 2024 May 7–11; Vienna, Austria.

48. Gao Y, Xiong Y, Gao X, Jia K, Pan J, Bi Y, Dai Y, Sun J, Wang M, Wang H. Retrieval-augmented generation for large language models: A survey. arXiv. 2023. https://doi.org/10.48550/arXiv. 2312.10997

49. Zhao P, Zhang H, Yu Q, Wang Z, Geng Y, Fu F, Yan L, Zhang W, Jiang J, Cui B. Retrieval-augmented generation for AI-generated content: A survey. arXiv. 2024. https://doi.org/10.48550/arXiv.2402.19473

50. Park JS, O'Brien JC, Cai CJ, Morris MR, Liang P, Bernstein MS. Generative agents: Interactive simulacra of human behavior. Paper presented at: Proceedings of the

36th Annual ACM Symposium on User Interface Software and Technology, UIST 2023; 2023 Oct 29–Nov 1; San Francisco, CA, USA.

51. Xu Y, Wang S, Li Ps. Exploring large language models for communication games: An empirical study on werewolf. arXiv. 2023. https://doi.org/10.48550/arXiv.2309.04658

52. Zhou W, Jiang YE, Li L, Wu J, Wang T, Qiu S, Zhang J, Chen J, Wu R, Wang S, et al. Agents: An open-source framework for autonomous language agents. arXiv. 2023. https://doi.org/10.48550/arXiv.2309.07870

53. Zhu X, Chen Y, Tian H, Tao C, Su W, Yang C, Huang G, Li B, Lu L, Wang X, et al. Ghost in the Minecraft: Generally capable agents for OpenWorld environments via large language models with text-based knowledge and memory. arXiv. 2023. https://doi.org/10.48550/arXiv.2305.17144

54. Piterbarg U, Pinto L, Fergus R. diff history for neural language agents. Paper presented at: Forty-first International Conference on Machine Learning, ICML 2024; 2024 Jul 21–27; Vienna, Austria.

55. Aeronautiques C, Howe A, Knoblock C, McDermott ISID, Ram A, Veloso M, Weld D, Sri DW, Barrett A, Christianson D. Pddl| the planning domain definition language. Technical Report; 1998.

56. Silver T, Hariprasad V, Shuttleworth RS, Kumar N, Lozano-Pérez T, Kaelbling LP. PDDL planning with pretrained large language models. Paper presented at: NeurIPS 2022 Foundation Models for Decision Making Workshop; 2022; New Orleans, LA, USA.

57. Silver T, Dan S, Srinivas K, Tenenbaum JB, Kaelbling LP, Katz M. Generalized planning in PDDL domains with pretrained large language models. Proc AAAI Conf Artif Intell. 2024;38(18):20256–20264.

58. Valmeekam K, Marquez M, Sreedharan S, Kambhampati S. On the planning abilities of large language models—A critical investigation. Paper presented at: Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023; 2023 Dec 10–16; New Orleans, LA, USA.

59. Chen G, Yang L, Jia R, Hu Z, Chen Y, Zhang W, Wang W, Pan J. Language-augmented symbolic planner for open-world task planning. arXiv. 2024. https://doi.org/10.48550/arXiv.2407.09792

60. Han M, Zhu Y, Zhu S, Wu YN, Zhu Y. InterPreT: Interactive predicate learning from language feedback for generalizable task planning. arXiv. 2024. https://doi.org/10.48550/arXiv.2405.19758

61. Guan L, Valmeekam K, Sreedharan S, Kambhampati S. Leveraging Pre-trained large language models to construct and utilize world models for model-based task planning. Paper presented at: Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023; 2023 Dec 10–16; New Orleans, LA, USA.

62. Kambhampati S, Valmeekam K, Guan L, Verma M, Stechly K, Bhambri S, Saldyt LP, Murthy AB. Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. Paper presented at: Forty-first International Conference on Machine Learning, ICML 2024; 2024 Jul 21–27; Vienna, Austria.

63. Singh I, Traum D, Thomason J. TwoStep: Multi-agent task planning using classical planners and large language models. arXiv. 2024. https://doi.org/10.48550/arXiv.2403.17246.

64. Liu B, Jiang Y, Zhang X, Liu Q, Zhang S, Biswas J, Stone P, LLM+P: Empowering large language models with optimal planning proficiency. arXiv. 2023. https://doi.org/10.48550/arXiv.2304.11477

65. Xie Y, Yu C, Zhu T, Bai J, Gong Z, Soh H. Translating natural language to planning goals with large-language models. arXiv. 2023. https://doi.org/10.48550/arXiv.2302.05128

66. Oswald JT, Srinivas K, Kokel H, Lee J, Katz M, Sohrabi S. Large language models as planning domain generators (student abstract). Paper presented at: Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014; 2024 Feb 20–27; Vancouver, Canada.

67. Yang Z, Ishay A, Lee J. Coupling large language models with logic programming for robust and general reasoning from text. Paper presented at: Findings of the Association for Computational Linguistics: ACL 2023; 2023 Jul 9–14; Toronto, Canada.

68. Pan L, Albalak A, Wang X, Wang WY. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. Paper presented at: Findings of the Association for Computational Linguistics: EMNLP 2023; 2023 Dec 6–10; Singapore.

69. Zhou Z, Song J, Yao K, Shu Z, Ma L. ISR-LLM: Iterative self-refined large language model for long-horizon sequential task planning. Paper presented at: IEEE International Conference on Robotics and Automation, ICRA 2024; 2024 May 13–17; Yokohama, Japan.

70. Yang Y, Tomar A. On the planning, search, and memorization capabilities of large language models. arXiv. 2023. https://doi.org/10.48550/arXiv.2309.01868

71. Birr T, Pohl C, Younes A, Asfour T. AutoGPT+P: Affordance-based task planning with large language models. arXiv. 2024. https://doi.org/10.48550/arXiv.2402.10778

72. GO TO REFERENCE

73. Google Scholar

74. 69

75. Cao Y, Zhao H, Cheng Y, Shu T, Chen Y, Liu G, Liang G, Zhao J, Yan J, Li Y. Survey on large language model-enhanced reinforcement learning: Concept, taxonomy, and methods. arXiv. 2024. https://doi.org/10.48550/arXiv.2404.00282

76. Hao S, Gu Y, Ma H, Hong J, Wang Z, Wang D, Hu Z. Reasoning with language model is planning with world model. Paper presented at: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023; 2023 Dec 6–10; Singapore.

77. Wan Z, Feng X, Wen M, Mc Aleer SM, Wen Y, Zhang W, Wang J. AlphaZero-like tree-search can guide large language model decoding and training. Paper

presented at: Forty-first International Conference on Machine Learning, ICML 2024; 2024 Jul 21–27; Vienna, Austria.

78. Deng Y, Zhang W, Lam W, Ng SK, Chua TS. Plug-and-play policy planner for large language model powered dialogue agents. Paper presented at: The Twelfth International Conference on Learning Representations, ICLR 2024; 2024 May 7–11; Vienna, Austria.

79. Hazra R, Martires PZD, Raedt LD. SayCanPay: Heuristic planning with large language models using learnable domain knowledge. Paper presented at: Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014; 2024 Feb 20–27; Vancouver, Canada.

80. Liu Z, Hu H, Zhang S, Guo H, Ke S, Liu B, Wang Z. Reason for future, act for now: A principled architecture for autonomous LLM agents. Paper presented at: Forty-first International Conference on Machine Learning, ICML 2024; 2024 Jul 21–27; Vienna, Austria.

81. Murthy R, Heinecke S, Niebles JC, Liu Z, Xue L, Yao W, Feng Y, Chen Z, Gokul A, Arpit D, et al. REX: Rapid exploration and eXploitation for AI agents. arXiv. 2023. https://doi.org/10.48550/arXiv.2307.08962

82. Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, Lanctot M, Sifire L, Kumaran D, Graepel T, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv. 2017. https://doi.org/10.48550/arXiv.1712.01815

83. Ouyang L, Wu J, Jiang X, Almeida D, Wainwright CL, Mishkin P, Zhang C, Agarwal S, Slama K, Ray A, et al. Training language models to follow instructions with human feedback. Paper presented at: Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022; 2022 Nov 28–Dec 9; New Orleans, LA, USA.

84. Lee H, Phatale S, Mansoor H, Lu KR, Mesnard T, Ferret J, Bishop C, Hall E, Carbune V, Rastogi A. RLAIF vs. RLHF: Scaling reinforcement learning from human

feedback with AI feedback. Paper presented at: Forty-First International Conference on Machine Learning, ICML 2024; 2024 Jul 21–27; Vienna, Austria.

85. Yao W, Heinecke S, Niebles JC, Liu Z, Feng Y, Xue L, Rithesh RN, Chen Z, Zhang J, Arpit D, et al. Retroformer: Retrospective large language agents with policy gradient optimization. Paper presented at: The Twelfth International Conference on Learning Representations, ICLR 2024; 2024 May 7–11; Vienna, Austria.

86. Yang J, Dong Y, Liu S, Li B, Wang Z, Tan H, Jiang C, Kang J, Zhang Y, Zhou K, et al. Octopus: Embodied vision-language programmer from environmental feedback. Paper presented at: Computer Vision—ECCV 2024—18th European Conference; 2024 Sep 29–Oct 4; Milan, Italy.

87. Du Y, Watkins O, Wang Z, Colas C, Darrell T, Abbeel P, Gupta A, Andreas J. Guiding pretraining in reinforcement learning with large language models. Paper presented at: International Conference on Machine Learning, ICML 2023; 2023 July 23–29; Honolulu, Hawaii, USA.

88. Carta T, Romac C, Wolf T, Lamprier S, Sigaud O, Oudeyer P. Grounding large language models in interactive environments with online reinforcement learning. Paper presented at: International Conference on Machine Learning, ICML 2023; 2023 July 23–29; Honolulu, Hawaii, USA.

89. Wang K, Lu Y, Santacroce M, Gong Y, Zhang C, Shen Y. Adapting LLM agents through communication. arXiv. 2023. https://doi.org/10.48550/arXiv.2310.01444

90. Xu Z, Yu C, Fang F, Wang Y, Wu Y. Language agents with reinforcement learning for strategic play in the werewolf game. Paper presented at: Forty-first International Conference on Machine Learning, ICML 2024; 2024 Jul 21–27; Vienna, Austria.

91. Zhang D, Chen L, Zhang S, Xu H, Zhao Z, Yu K. Large language models are semiparametric reinforcement learning agents. Paper presented at: Advances in Neural Information ProcessingSystems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023; 2023 Dec 10–16; New Orleans, LA, USA.

92. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. arXiv. 2017. https://doi.org/10.48550/arXiv.1707.06347

93. Zhang Y, Mao S, Ge T, Wang X, Xia Y, Lan M, Wei F. K-level reasoning with large language models. arXiv. 2024. https://doi.org/10.48550/arXiv.2402.01521

94. Guo J, Yang B, Yoo P, Lin BY, Iwasawa Y, Matsuo Y. Suspicion-agent: Playing imperfect information games with theory of mind aware GPT-4. arXiv. 2023. https://doi.org/10.48550/arXiv.2309.17277

95. Gemp I, Patel R, Bachrach Y, Lanctot M, Dasagi V, Marris L, Piliouras G, Liu S, Tuyls K. States as strings as strategies: Steering language models with game-theoretic solvers. arXiv. 2024. https://doi.org/10.48550/arXiv.2402.01704

96. Mao S, Cai Y, Xia Y, Wu W, Wang X, Wang F, Ge T, Wei F. ALYMPICS: Language agents meet game theory. arXiv. 2023. https://doi.org/10.48550/arXiv.2311.03220

97. Duan J, Zhang R, Diffenderfer J, Kaikhura B, Sun L, Stengel-Eskin E, Bansal M, Chen T, Xu K. GTBench: Uncovering the strategic reasoning limitations of LLMs via game-theoretic evaluations. arXiv. 2024. https://doi.org/10.48550/arXiv. 2402.12348

98. Fan C, Chen J, Jin Y, He H. Can large language models serve as rational players in game theory? A systematic analysis. Paper presented at: Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014; 2024 Feb 20–27; Vancouver, Canada.

99. Huang J, Li EJ, Lam MH, Liang T, Wang W, Yuan Y, Jiao W, Wang X, Tu Z, Lyu MR. How far are we on the decision-making of LLMs? Evaluating LLMs' gaming ability in multi-agent environments. arXiv. 2024. https://doi.org/10.48550/arXiv. 2403.11807

100. Webb TW, Mondal SS, Wang C, Krabach B, Momennejad I. A prefrontal cortex-inspired architecture for planning in large language models. arXiv. 2023. https://doi.org/10.48550/arXiv.2308.09658

101. Hu P, Qi J, Li X, Li H, Wang X, Quan B, Wang R, Zhou Y. Tree-of-mixed-thought: Combining fast and slow thinking for multihop visual reasoning. arXiv. 2023. https://doi.org/10.48550/arXiv.2308.09658

102. Lin BY, Fu Y, Yang K, Brahman F, Huang S, Bhagavatula C, Ammanabrolu P, Choi Y, Ren X. SwiftSage: A generative agent with fast and slow thinking for complex interactive tasks. Paper presented at: Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023; 2023 Dec 10–16; New Orleans, LA, USA.

103. Wu X, Shen Y, Shan C, Song K, Wang S, Zhang B, Feng J, Cheng H, Chen W, Xiong Y, et al. Can graph learning improve task planning? arXiv. 2024. https://doi.org/10.48550/arXiv.2405.19119

104. Wang Z, Cai S, Chen G, Liu A, Ma X, Liang Y. Describe, explain, plan and select: Interactive planning with LLMs enables open-world multi-task agents. Paper presented at: Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023; 2023 Dec 10–16; New Orleans, LA, USA.

105. Cai S, Wang Z, Ma X, Liu A, Liang Y. Open-world multi-task control through goal aware representation learning and adaptive horizon prediction. Paper presented at: IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023; 2023 Jun 17–24; Vancouver, BC, Canada.

106. Yuan H, Zhang C, Wang H, Xie F, Cai P, Dong H, Lu Z. Plan4MC: Skill reinforcement learning and planning for open-world Minecraft tasks. arXiv. 2023. https://doi.org/10.48550/arXiv.2303.16563

107. Wu Y, Tang X, Mitchell TM, Li Y. SmartPlay: A benchmark for LLMs as intelligent agents. Paper presented at: The Twelfth International Conference on Learning Representations, ICLR 2024; 2024 May 7–11; Vienna, Austria.

108. Liu J, Yu C, Gao J, Xie Y, Liao Q, Wu Y, Wang Y. LLM-powered hierarchical language agent for real-time human AI coordination. Paper presented at: Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2024; 2024 May 6–10; Auckland, New Zealand.

109. Lan Y, Hu Z, Wang L, Ye D, Zhao P, Lim E-P, Xiong H, Wang H. LLM-based agent society investigation: Collaboration and confrontation in Avalon gameplay. arXiv. 2023. https://doi.org/10.48550/arXiv.2310.14985

110. Wang S, Liu C, Zheng Z, Qi S, Chen S, Yang Q, Zhao A, Wang C, Song S, Huang G. Avalon's game of thoughts: Battle against deception through recursive contemplation. arXiv. 2023. https://doi.org/10.48550/arXiv.2310.01320

111. Light J, Cai M, Shen S, Hu Z. Avalonbench: Evaluating llms playing the game of avalon. Paper presented at: NeurIPS 2023 Foundation Models for Decision Making Workshop; 2023; New Orleans, LA, USA.

112. Лисенко С., Величко Н. Метод планування та розподілу задач у гетерогенних обчислювальних системах. *Комп'ютерні інтелектуальні системи та мережі.* Матеріали XIX Всеукраїнської науково практичної WEB конференції аспірантів, студентів та молодих вчених (25-27 березня 2026 р.). Кривий Ріг: Криворізький національний університет, 2026. С. 24-27.

ДОДАТОК А
(обов'язковий)

Лістинг системного програмного забезпечення реалізації системи

```
-- ============================================================
--  HRLHS Task Scheduler for Heterogeneous FPGA-based Systems
--  Магістерська робота: Метод планування та розподілу задач
--  у гетерогенних обчислювальних системах на базі FPGA
-- ============================================================
--
--  Архітектура реалізує:
--    1. Модуль формування та зберігання DAG задач
--    2. Модуль оцінювання надійності (Lji = exp(-λ * B_ij))
--    3. Модуль обчислення енергоспоживання (Udin + Ustat)
--    4. Планувальник задач із пріоритетними чергами
--    5. Модуль динамічного резервування (Dynamic Redundancy)
--    6. Диспетчер призначення задач на процесори (CPU/FPGA)
--
--  Параметри системи (з Таблиці 2.1 роботи):
--    R = 9  задач
--    C = 3  процесори (P1, P2, P3)
--    Часи виконання Bij задані в таблиці ваг графа
-- ============================================================

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- ============================================================
--  ПАКЕТ ТИПІВ системи планування
-- ============================================================
package scheduler_pkg is

    -- Константи розмірності системи
    constant MAX_TASKS  : integer := 9;   -- R: кількість задач
    constant MAX_PROC   : integer := 3;   -- C: кількість процесорів
    constant TIME_BITS  : integer := 16;  -- розрядність часових
параметрів
    constant ENERGY_BITS: integer := 20;  -- розрядність
енергетичних параметрів
    constant RELY_BITS  : integer := 16;  -- розрядність показника
надійності (Q15)

    -- Стани задачі
    type task_state_t is (
        TASK_IDLE,      -- задача не активна
        TASK_READY,     -- готова до виконання (всі попередники
завершені)
        TASK_ASSIGNED,  -- призначена на процесор
        TASK_RUNNING,   -- виконується
        TASK_DONE,      -- завершена
```

```
        TASK_FAILED,    -- виникла відмова
        TASK_BACKUP     -- є резервною копією
    );

    -- Стани процесора
    type proc_state_t is (
        PROC_IDLE,      -- вільний
        PROC_BUSY,      -- зайнятий
        PROC_OFFLINE    -- вимкнений (для економії енергії)
    );

    -- Запис параметрів задачі
    type task_record_t is record
        state       : task_state_t;
        assigned_to : integer range 0 to MAX_PROC-1; -- індекс
процесора
        start_time  : unsigned(TIME_BITS-1 downto 0);
        end_time    : unsigned(TIME_BITS-1 downto 0);
        priority    : unsigned(7 downto 0);           -- пріоритет
(0=найвищий)
        is_backup   : std_logic;                      -- '1' якщо
резервна копія
        original_id : integer range 0 to MAX_TASKS-1; -- ID
оригінальної задачі
    end record;

    -- Масиви
    type task_array_t  is array(0 to MAX_TASKS-1) of task_record_t;
    type proc_state_array_t is array(0 to MAX_PROC-1) of
proc_state_t;
    type time_matrix_t is array(0 to MAX_TASKS-1, 0 to MAX_PROC-1)
                          of unsigned(TIME_BITS-1 downto 0);
    type rely_matrix_t is array(0 to MAX_TASKS-1, 0 to MAX_PROC-1)
                          of unsigned(RELY_BITS-1 downto 0);
    type adj_matrix_t  is array(0 to MAX_TASKS-1, 0 to MAX_TASKS-1)
                          of std_logic;  -- матриця суміжності DAG
    type comm_matrix_t is array(0 to MAX_TASKS-1, 0 to MAX_TASKS-1)
                          of unsigned(TIME_BITS-1 downto 0); --
затримки зв'язку Aij

end package scheduler_pkg;

-- ============================================================
--  МОДУЛЬ 1: Ініціалізатор параметрів системи (DAG + ваги)
--  Завантажує часи виконання Bij та матрицю суміжності
-- ============================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.scheduler_pkg.all;

entity dag_init is
    port (
```

```
        clk      : in  std_logic;
        reset    : in  std_logic;
        -- Матриця часів виконання Bij (задача x процесор)
        B_matrix : out time_matrix_t;
        -- Матриця суміжності DAG (скрінений ациклічний граф)
        adj      : out adj_matrix_t;
        -- Затримки зв'язку між задачами Aij
        A_comm   : out comm_matrix_t;
        -- Сигнал готовності
        ready    : out std_logic
    );
end entity dag_init;

architecture rtl of dag_init is
begin
    process(clk, reset)
    begin
        if reset = '1' then
            ready <= '0';
        elsif rising_edge(clk) then

            -- ------------------------------------------------
            -- Матриця часів виконання Bij (Таблиця 2.1 роботи)
            -- Рядки: задачі 0..8, Стовпці: процесори P1,P2,P3
            -- ------------------------------------------------
            -- Задача 1 (індекс 0)
            B_matrix(0,0) <= to_unsigned(14, TIME_BITS);
            B_matrix(0,1) <= to_unsigned( 6, TIME_BITS);
            B_matrix(0,2) <= to_unsigned( 9, TIME_BITS);
            -- Задача 2
            B_matrix(1,0) <= to_unsigned(13, TIME_BITS);
            B_matrix(1,1) <= to_unsigned( 5, TIME_BITS);
            B_matrix(1,2) <= to_unsigned( 8, TIME_BITS);
            -- Задача 3
            B_matrix(2,0) <= to_unsigned( 9, TIME_BITS);
            B_matrix(2,1) <= to_unsigned( 8, TIME_BITS);
            B_matrix(2,2) <= to_unsigned(11, TIME_BITS);
            -- Задача 4
            B_matrix(3,0) <= to_unsigned(10, TIME_BITS);
            B_matrix(3,1) <= to_unsigned(15, TIME_BITS);
            B_matrix(3,2) <= to_unsigned( 7, TIME_BITS);
            -- Задача 5
            B_matrix(4,0) <= to_unsigned(12, TIME_BITS);
            B_matrix(4,1) <= to_unsigned(13, TIME_BITS);
            B_matrix(4,2) <= to_unsigned(10, TIME_BITS);
            -- Задача 6
            B_matrix(5,0) <= to_unsigned(18, TIME_BITS);
            B_matrix(5,1) <= to_unsigned(10, TIME_BITS);
            B_matrix(5,2) <= to_unsigned( 8, TIME_BITS);
            -- Задача 7
            B_matrix(6,0) <= to_unsigned(21, TIME_BITS);
            B_matrix(6,1) <= to_unsigned(17, TIME_BITS);
            B_matrix(6,2) <= to_unsigned(22, TIME_BITS);
```

```
            -- Задача 8
            B_matrix(7,0) <= to_unsigned(18, TIME_BITS);
            B_matrix(7,1) <= to_unsigned(19, TIME_BITS);
            B_matrix(7,2) <= to_unsigned(24, TIME_BITS);
            -- Задача 9
            B_matrix(8,0) <= to_unsigned(23, TIME_BITS);
            B_matrix(8,1) <= to_unsigned( 7, TIME_BITS);
            B_matrix(8,2) <= to_unsigned(16, TIME_BITS);

            -- ------------------------------------------------
            -- Матриця суміжності DAG (з рисунка 2.1 роботи)
            -- adj(i,j)='1' означає: задача i є попередником j
            -- ------------------------------------------------
            for i in 0 to MAX_TASKS-1 loop
                for j in 0 to MAX_TASKS-1 loop
                    adj(i,j) <= '0';
                end loop;
            end loop;
            -- Ребра графа (типова топологія для 9-задачного DAG)
            adj(0,1) <= '1'; adj(0,2) <= '1'; adj(0,3) <= '1';
            adj(1,4) <= '1'; adj(1,5) <= '1';
            adj(2,4) <= '1'; adj(2,6) <= '1';
            adj(3,5) <= '1'; adj(3,6) <= '1';
            adj(4,7) <= '1';
            adj(5,7) <= '1'; adj(5,8) <= '1';
            adj(6,8) <= '1';

            -- ------------------------------------------------
            -- Затримки зв'язку Aij між задачами
            -- (використовуються якщо задачі на різних процесорах)
            -- ------------------------------------------------
            for i in 0 to MAX_TASKS-1 loop
                for j in 0 to MAX_TASKS-1 loop
                    A_comm(i,j) <= to_unsigned(0, TIME_BITS);
                end loop;
            end loop;
            A_comm(0,1) <= to_unsigned(3, TIME_BITS);
            A_comm(0,2) <= to_unsigned(3, TIME_BITS);
            A_comm(0,3) <= to_unsigned(2, TIME_BITS);
            A_comm(1,4) <= to_unsigned(2, TIME_BITS);
            A_comm(1,5) <= to_unsigned(3, TIME_BITS);
            A_comm(2,4) <= to_unsigned(2, TIME_BITS);
            A_comm(2,6) <= to_unsigned(2, TIME_BITS);
            A_comm(3,5) <= to_unsigned(2, TIME_BITS);
            A_comm(3,6) <= to_unsigned(3, TIME_BITS);
            A_comm(4,7) <= to_unsigned(1, TIME_BITS);
            A_comm(5,7) <= to_unsigned(1, TIME_BITS);
            A_comm(5,8) <= to_unsigned(2, TIME_BITS);
            A_comm(6,8) <= to_unsigned(2, TIME_BITS);

            ready <= '1';
        end if;
    end process;
```

```
end architecture rtl;

-- ============================================================
--  МОДУЛЬ 2: Обчислення надійності задач
--  Lji = exp(-λj * Bij) апроксимується лінійно в Q15
--  Вхід: Bij (ціле), λ_scaled (Q8 = λ*256)
--  Вихід: Lji у форматі Q15 (0..32767)
-- ============================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity reliability_calc is
    port (
        clk         : in  std_logic;
        reset       : in  std_logic;
        -- Час виконання задачі на процесорі (такти)
        B_ij        : in  unsigned(15 downto 0);
        -- Інтенсивність відмов λj (формат Q8 = λ * 256)
        lambda_q8   : in  unsigned(15 downto 0);
        -- Показник надійності Lji у форматі Q15
        L_ji        : out unsigned(15 downto 0);
        valid_out   : out std_logic
    );
end entity reliability_calc;

architecture rtl of reliability_calc is
    -- Апроксимація exp(-x) для x = λ*B
    -- exp(-x) ≈ 1/(1+x) для малих x, або кусково-лінійна
апроксимація
    -- Використовуємо: exp(-x) ≈ max(0, 1 - x) для спрощення
апаратної реалізації
    -- Більш точна апроксимація: exp(-x) ≈ (1 - x/2)^2 для малих x
    signal lambda_B   : unsigned(31 downto 0);
    signal exp_approx : unsigned(15 downto 0);
    signal stage      : integer range 0 to 3;
begin
    process(clk, reset)
        variable lB_shifted : unsigned(31 downto 0);
        variable result     : unsigned(31 downto 0);
    begin
        if reset = '1' then
            L_ji      <= (others => '0');
            valid_out <= '0';
            stage     <= 0;
            lambda_B  <= (others => '0');
        elsif rising_edge(clk) then
            case stage is
                when 0 =>
                    -- Крок 1: обчислити λ * B (Q8 * int = Q8)
                    lambda_B  <= lambda_q8 * B_ij;
                    valid_out <= '0';
                    stage     <= 1;
```

```
                when 1 =>
                    -- Крок 2: масштабувати до Q15:  x_q15 = λB >>
(8-15) = λB << 7
                    -- λ*B у Q8, треба Q15 → помножити на 128
                    -- Кусково-лінійна апроксимація exp(-x):
                    --   якщо x_q15 >= 32768 (x >= 1.0): Lji ≈ 0 (не
надійний)
                    --   інакше: Lji ≈ 32767 - x_q15   (лінійна
ділянка)
                    lB_shifted := lambda_B(24 downto 0) & "0000000";
-- << 7
                    if lB_shifted(31 downto 16) /= x"0000" then
                        -- x >= 1.0, надійність дуже низька
                        exp_approx <= to_unsigned(100, 16); --
~0.003
                    elsif lB_shifted(15 downto 0) >=
to_unsigned(16384, 16) then
                        -- 0.5 <= x < 1.0: лінійна ділянка 2
                        -- exp(-x) ≈ 0.5 * (1 - (x-0.5))
                        result := to_unsigned(32767, 32) -
                                  (lB_shifted(15 downto 0) & "0");
                        if result(31) = '1' then
                            exp_approx <= to_unsigned(200, 16);
                        else
                            exp_approx <= result(15 downto 0);
                        end if;
                    else
                        -- 0 <= x < 0.5: exp(-x) ≈ 1 - x
                        result := to_unsigned(32767, 32) -
                                  resize(lB_shifted(15 downto 0),
32);
                        exp_approx <= result(15 downto 0);
                    end if;
                    stage <= 2;

                when 2 =>
                    L_ji      <= exp_approx;
                    valid_out <= '1';
                    stage     <= 3;

                when 3 =>
                    valid_out <= '0';
                    stage     <= 0;
            end case;
        end if;
    end process;
end architecture rtl;

-- ============================================================
--  МОДУЛЬ 3: Обчислення енергоспоживання
--  U = Ustat + Udyn
--  Udyn(i) = Ndyn_c(j) * ki * Bij
```

```
--  Ustat(i) = Nstat_c(j) * pj * cs
-- ============================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.scheduler_pkg.all;

entity energy_calc is
    port (
        clk            : in  std_logic;
        reset          : in  std_logic;
        -- Час виконання задачі Bij
        B_ij           : in  unsigned(TIME_BITS-1 downto 0);
        -- Коефіцієнт динамічного споживання (Q8: Ndyn*256)
        ndyn_q8        : in  unsigned(15 downto 0);
        -- Коефіцієнт статичного споживання (Q8)
        nstat_q8       : in  unsigned(15 downto 0);
        -- Робоча частота процесора ki (Q8: ki*256, max=256)
        freq_q8        : in  unsigned(7 downto 0);
        -- Стан процесора (1=активний)
        proc_active    : in  std_logic;
        -- Тривалість активного стану cs
        cs             : in  unsigned(TIME_BITS-1 downto 0);
        -- Результат: динамічне та статичне споживання (Q8)
        energy_dyn     : out unsigned(ENERGY_BITS-1 downto 0);
        energy_stat    : out unsigned(ENERGY_BITS-1 downto 0);
        energy_total   : out unsigned(ENERGY_BITS-1 downto 0);
        valid_out      : out std_logic
    );
end entity energy_calc;

architecture rtl of energy_calc is
    signal dyn_stage  : unsigned(31 downto 0);
    signal stat_stage : unsigned(31 downto 0);
    signal calc_done  : std_logic;
begin
    process(clk, reset)
    begin
        if reset = '1' then
            energy_dyn   <= (others => '0');
            energy_stat  <= (others => '0');
            energy_total <= (others => '0');
            valid_out    <= '0';
            calc_done    <= '0';
        elsif rising_edge(clk) then
            if calc_done = '0' then
                -- Udyn = Ndyn_c * ki * Bij  (всі у Q8, результат
Q24→зберігаємо Q8)
                dyn_stage  <= ndyn_q8 * freq_q8 * B_ij;

                -- Ustat = Nstat_c * p * cs  (p=0 або 1)
                if proc_active = '1' then
                    stat_stage <= nstat_q8 * cs;
```

```
                else
                    stat_stage <= (others => '0');
                end if;

                calc_done <= '1';
                valid_out <= '0';
            else
                -- Зберігаємо старші біти (Q8 після скорочення)
                energy_dyn   <= dyn_stage(ENERGY_BITS+7 downto 8);
                energy_stat  <= stat_stage(ENERGY_BITS+7 downto 8);
                energy_total <= dyn_stage(ENERGY_BITS+7 downto 8) +
                                stat_stage(ENERGY_BITS+7 downto 8);
                valid_out    <= '1';
                calc_done    <= '0';
            end if;
        end if;
    end process;
end architecture rtl;

-- ============================================================
--  МОДУЛЬ 4: Перевірка готовності задачі
--  Задача готова, якщо всі її попередники завершені
-- ============================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.scheduler_pkg.all;

entity readiness_checker is
    port (
        clk          : in  std_logic;
        reset        : in  std_logic;
        -- Матриця суміжності DAG
        adj          : in  adj_matrix_t;
        -- Вектор стану завершення задач
        task_done    : in  std_logic_vector(MAX_TASKS-1 downto 0);
        -- ID задачі для перевірки
        task_id      : in  integer range 0 to MAX_TASKS-1;
        -- '1' якщо задача готова до виконання
        is_ready     : out std_logic;
        valid_out    : out std_logic
    );
end entity readiness_checker;

architecture rtl of readiness_checker is
begin
    process(clk, reset)
        variable all_pred_done : std_logic;
    begin
        if reset = '1' then
            is_ready  <= '0';
            valid_out <= '0';
        elsif rising_edge(clk) then
```

```
            all_pred_done := '1';
            -- Перевіряємо всіх попередників задачі task_id
            for pred in 0 to MAX_TASKS-1 loop
                if adj(pred, task_id) = '1' then
                    -- pred є попередником task_id
                    if task_done(pred) = '0' then
                        all_pred_done := '0';
                    end if;
                end if;
            end loop;
            is_ready  <= all_pred_done;
            valid_out <= '1';
        end if;
    end process;
end architecture rtl;

-- ============================================================
--  МОДУЛЬ 5: Пріоритетна черга задач (Priority Queue)
--  Реалізує сортування задач за показником надійності
--  (задачі з нижчою надійністю обробляються пріоритетно
--   для застосування динамічного резервування)
-- ============================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.scheduler_pkg.all;

entity priority_queue is
    port (
        clk          : in  std_logic;
        reset        : in  std_logic;
        -- Вхід: надійність задачі та її ID
        enqueue_en   : in  std_logic;
        task_id_in   : in  integer range 0 to MAX_TASKS-1;
        reliability  : in  unsigned(15 downto 0);  -- Q15
        -- Вихід: задача з найнижчою надійністю (для резервування)
        dequeue_en   : in  std_logic;
        task_id_out  : out integer range 0 to MAX_TASKS-1;
        rely_out     : out unsigned(15 downto 0);
        queue_empty  : out std_logic;
        queue_full   : out std_logic
    );
end entity priority_queue;

architecture rtl of priority_queue is
    type queue_id_t   is array(0 to MAX_TASKS-1) of integer range 0
to MAX_TASKS-1;
    type queue_rely_t is array(0 to MAX_TASKS-1) of unsigned(15
downto 0);

    signal q_ids   : queue_id_t;
    signal q_rely  : queue_rely_t;
    signal q_count : integer range 0 to MAX_TASKS;
```

```
begin
    process(clk, reset)
        variable min_rely  : unsigned(15 downto 0);
        variable min_idx   : integer range 0 to MAX_TASKS-1;
        variable temp_id   : integer range 0 to MAX_TASKS-1;
        variable temp_rely : unsigned(15 downto 0);
    begin
        if reset = '1' then
            q_count     <= 0;
            queue_empty <= '1';
            queue_full  <= '0';
            task_id_out <= 0;
            rely_out    <= (others => '0');
        elsif rising_edge(clk) then

            -- Вставка нової задачі в кінець черги
            if enqueue_en = '1' and q_count < MAX_TASKS then
                q_ids(q_count)  <= task_id_in;
                q_rely(q_count) <= reliability;
                q_count         <= q_count + 1;
            end if;

            -- Вилучення задачі з найнижчою надійністю (insertion-
sort крок)
            if dequeue_en = '1' and q_count > 0 then
                -- Знаходимо мінімум (lowest reliability = highest
backup priority)
                min_rely := q_rely(0);
                min_idx  := 0;
                for k in 1 to MAX_TASKS-1 loop
                    if k < q_count then
                        if q_rely(k) < min_rely then
                            min_rely := q_rely(k);
                            min_idx  := k;
                        end if;
                    end if;
                end loop;

                task_id_out <= q_ids(min_idx);
                rely_out    <= q_rely(min_idx);

                -- Видаляємо елемент min_idx зі зсувом масиву
                for k in 0 to MAX_TASKS-2 loop
                    if k >= min_idx and k < q_count-1 then
                        q_ids(k)  <= q_ids(k+1);
                        q_rely(k) <= q_rely(k+1);
                    end if;
                end loop;
                q_count <= q_count - 1;
            end if;

            -- Оновлення прапорців
```

```
            if q_count = 0 then
                queue_empty <= '1';
            else
                queue_empty <= '0';
            end if;
            if q_count = MAX_TASKS then
                queue_full <= '1';
            else
                queue_full <= '0';
            end if;
        end if;
    end process;
end architecture rtl;

-- ============================================================
--  МОДУЛЬ 6: Модуль динамічного резервування
--  Алгоритм 3.2 з роботи:
--    - Оцінює надійність кожної задачі
--    - Порівнює з порогом Rreq
--    - Створює резервні копії для задач із низькою надійністю
--    - Призначає резервні задачі на альтернативні процесори
-- ============================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.scheduler_pkg.all;

entity dynamic_redundancy is
    port (
        clk               : in  std_logic;
        reset             : in  std_logic;
        enable            : in  std_logic;
        -- Матриця надійностей Lji для всіх задач та процесорів
(Q15)
        rely_matrix       : in  rely_matrix_t;
        -- Матриця призначень (task -> proc)
        assignment        : in  std_logic_vector(MAX_TASKS*2-1
downto 0);
        -- Поріг надійності Rreq (Q15)
        rely_threshold    : in  unsigned(15 downto 0);
        -- Вихідні сигнали: які задачі потребують резервування
        backup_needed     : out std_logic_vector(MAX_TASKS-1 downto
0);
        -- Для кожної backup-задачі: альтернативний процесор
        backup_proc       : out std_logic_vector(MAX_TASKS*2-1
downto 0);
        -- Продукт загальної надійності (Q15 appr.)
        total_reliability : out unsigned(15 downto 0);
        -- '1' якщо вимоги до надійності виконані без резервування
        rely_ok           : out std_logic;
        done              : out std_logic
    );
end entity dynamic_redundancy;
```

```
architecture rtl of dynamic_redundancy is
    type state_t is (IDLE, CALC_RELY, CHECK_TOTAL, SORT_TASKS,
ASSIGN_BACKUP, COMPLETE);
    signal state        : state_t;
    signal task_idx     : integer range 0 to MAX_TASKS;
    signal total_rely   : unsigned(15 downto 0);
    signal proc_assign  : integer range 0 to MAX_PROC-1;
    signal backup_count : integer range 0 to MAX_TASKS;

    -- Масив надійностей задач (мінімум по процесорах)
    type rely_array_t is array(0 to MAX_TASKS-1) of unsigned(15
downto 0);
    signal task_rely : rely_array_t;

begin
    process(clk, reset)
        variable min_r   : unsigned(15 downto 0);
        variable proc_id : integer range 0 to MAX_PROC-1;
        variable alt_proc: integer range 0 to MAX_PROC-1;
        variable orig_proc: integer range 0 to MAX_PROC-1;
    begin
        if reset = '1' then
            state             <= IDLE;
            task_idx          <= 0;
            total_rely        <= to_unsigned(32767, 16); -- 1.0 y
Q15
            backup_needed     <= (others => '0');
            backup_proc       <= (others => '0');
            rely_ok           <= '0';
            done              <= '0';
            backup_count      <= 0;
        elsif rising_edge(clk) then
            case state is

                -- Очікування команди запуску
                when IDLE =>
                    done      <= '0';
                    if enable = '1' then
                        state     <= CALC_RELY;
                        task_idx  <= 0;
                        total_rely <= to_unsigned(32767, 16);
                        backup_needed <= (others => '0');
                    end if;

                -- Крок 1: Обчислення надійності кожної задачі
                -- Lji = rely_matrix(task, assigned_proc)
                when CALC_RELY =>
                    if task_idx < MAX_TASKS then
                        -- Отримуємо призначений процесор для задачі
task_idx
                        proc_id := to_integer(
```

```
                            unsigned(assignment(task_idx*2+1 downto
task_idx*2)));
                        task_rely(task_idx) <= rely_matrix(task_idx,
proc_id);

                        -- Накопичуємо добуток надійностей
(спрощено: min замість добутку)
                        -- В апаратурі добуток замінюємо мінімумом
для швидкості
                        if rely_matrix(task_idx, proc_id) <
total_rely then
                            total_rely <= rely_matrix(task_idx,
proc_id);
                        end if;

                        task_idx <= task_idx + 1;
                    else
                        state    <= CHECK_TOTAL;
                        task_idx <= 0;
                    end if;

                -- Крок 2: Перевірка загальної надійності
                when CHECK_TOTAL =>
                    total_reliability <= total_rely;
                    if total_rely >= rely_threshold then
                        rely_ok <= '1';
                    else
                        rely_ok <= '0';
                    end if;
                    state    <= SORT_TASKS;
                    task_idx <= 0;
                    backup_count <= 0;

                -- Крок 3: Виявлення задач, що потребують
резервування
                -- Задача потребує резервування якщо:
                --   - її надійність нижча за поріг
(rely_threshold), АБО
                --   - це критична задача (немає наступників →
термінальна)
                when SORT_TASKS =>
                    if task_idx < MAX_TASKS then
                        if task_rely(task_idx) < rely_threshold then
                            backup_needed(task_idx) <= '1';
                        else
                            backup_needed(task_idx) <= '0';
                        end if;
                        task_idx <= task_idx + 1;
                    else
                        state    <= ASSIGN_BACKUP;
                        task_idx <= 0;
                    end if;
```

```
                -- Крок 4: Призначення альтернативного процесора для
резервних задач
                -- Резервна задача НЕ повинна бути на тому ж
процесорі, що й оригінал
                when ASSIGN_BACKUP =>
                    if task_idx < MAX_TASKS then
                        if backup_needed(task_idx) = '1' then
                            orig_proc := to_integer(
                                unsigned(assignment(task_idx*2+1
downto task_idx*2)));

                            -- Вибір альтернативного процесора з
мінімальним Bij
                            min_r    := (others => '1');
                            alt_proc := 0;
                            for p in 0 to MAX_PROC-1 loop
                                if p /= orig_proc then
                                    -- Обираємо процесор з найкращою
надійністю
                                    if rely_matrix(task_idx, p) >
min_r or
                                       p = 0 then
                                        min_r    :=
rely_matrix(task_idx, p);
                                        alt_proc := p;
                                    end if;
                                end if;
                            end loop;

                            backup_proc(task_idx*2+1 downto
task_idx*2) <=

std_logic_vector(to_unsigned(alt_proc, 2));
                        end if;
                        task_idx <= task_idx + 1;
                    else
                        state <= COMPLETE;
                    end if;

                when COMPLETE =>
                    done  <= '1';
                    state <= IDLE;

            end case;
        end if;
    end process;
end architecture rtl;

-- ============================================================
--  МОДУЛЬ 7: Головний планувальник HRLHS
--  (Hybrid RL + Heuristic Scheduling)
--  Реалізує основний цикл планування:
--    1. Визначає готові задачі
```

```
--    2. Оцінює вартість призначення (час + енергія)
--    3. Вибирає оптимальний процесор
--    4. Диспетчеризує задачу
--    5. Контролює виконання та оновлює стан
-- ============================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.scheduler_pkg.all;

entity hrlhs_scheduler is
    port (
        clk              : in  std_logic;
        reset            : in  std_logic;
        start            : in  std_logic;

        -- Матриця часів виконання Bij
        B_matrix         : in  time_matrix_t;
        -- Матриця суміжності DAG
        adj              : in  adj_matrix_t;
        -- Матриця затримок зв'язку Aij
        A_comm           : in  comm_matrix_t;

        -- Параметри надійності λj для кожного процесора (Q8)
        lambda_proc      : in  std_logic_vector(MAX_PROC*8-1 downto
0);

        -- Параметри енергоспоживання
        ndyn_proc        : in  std_logic_vector(MAX_PROC*8-1 downto
0);   -- Q8
        nstat_proc       : in  std_logic_vector(MAX_PROC*8-1 downto
0);   -- Q8
        freq_proc        : in  std_logic_vector(MAX_PROC*8-1 downto
0);   -- Q8

        -- Поріг надійності (Q15)
        rely_threshold   : in  unsigned(15 downto 0);

        -- Вихід: вектор призначень (task -> proc)
        assignment       : out std_logic_vector(MAX_TASKS*2-1 downto
0);
        -- Вектор завершення задач
        task_done_vec    : out std_logic_vector(MAX_TASKS-1 downto
0);
        -- Загальний час завершення (makespan)
        makespan         : out unsigned(TIME_BITS-1 downto 0);
        -- Загальне енергоспоживання
        total_energy     : out unsigned(ENERGY_BITS-1 downto 0);
        -- Сигнал завершення планування
        schedule_done    : out std_logic;
        -- Поточний такт симуляції
        sim_time         : out unsigned(TIME_BITS-1 downto 0)
    );
```

```
end entity hrlhs_scheduler;

architecture rtl of hrlhs_scheduler is

    -- Стани головного автомата
    type fsm_state_t is (
        S_IDLE,
        S_INIT,
        S_FIND_READY,
        S_EVAL_PROC,
        S_ASSIGN_TASK,
        S_DISPATCH,
        S_MONITOR,
        S_CHECK_DONE,
        S_FINISH
    );
    signal state : fsm_state_t;

    -- Внутрішній стан задач та процесорів
    signal tasks       : task_array_t;
    signal proc_states : proc_state_array_t;

    -- Таймери виконання (що залишилось тактів)
    type timer_array_t is array(0 to MAX_PROC-1) of
unsigned(TIME_BITS-1 downto 0);
    signal proc_timers    : timer_array_t;
    signal proc_task_map  : std_logic_vector(MAX_PROC*4-1 downto 0);
-- proc->task

    -- Часові мітки фінішу кожної задачі
    type finish_time_t is array(0 to MAX_TASKS-1) of
unsigned(TIME_BITS-1 downto 0);
    signal task_finish    : finish_time_t;

    -- Вектор завершених задач
    signal done_vec       : std_logic_vector(MAX_TASKS-1 downto 0);

    -- Лічильник тактів симуляції
    signal sim_clk        : unsigned(TIME_BITS-1 downto 0);

    -- Поточна задача, що обробляється планувальником
    signal cur_task       : integer range 0 to MAX_TASKS-1;

    -- Найкращий процесор для поточної задачі
    signal best_proc      : integer range 0 to MAX_PROC-1;
    signal best_finish    : unsigned(TIME_BITS-1 downto 0);

    -- Накопичена енергія
    signal acc_energy     : unsigned(ENERGY_BITS-1 downto 0);

    -- Вектор призначень
    signal assign_vec     : std_logic_vector(MAX_TASKS*2-1 downto
0);
```

```
    -- Допоміжні функції
    function get_lambda(lv: std_logic_vector; p: integer) return
unsigned is
    begin
        return unsigned(lv(p*8+7 downto p*8));
    end function;

    function get_ndyn(ndv: std_logic_vector; p: integer) return
unsigned is
    begin
        return unsigned(ndv(p*8+7 downto p*8));
    end function;

    function get_freq(fv: std_logic_vector; p: integer) return
unsigned is
    begin
        return unsigned(fv(p*8+7 downto p*8));
    end function;

begin
    -- Підключення вихідних сигналів
    assignment    <= assign_vec;
    task_done_vec <= done_vec;
    sim_time      <= sim_clk;

    -- Головний процес планувальника
    process(clk, reset)
        variable pred_finish  : unsigned(TIME_BITS-1 downto 0);
        variable comm_delay   : unsigned(TIME_BITS-1 downto 0);
        variable earliest_start: unsigned(TIME_BITS-1 downto 0);
        variable candidate_finish: unsigned(TIME_BITS-1 downto 0);
        variable proc_free_at : unsigned(TIME_BITS-1 downto 0);
        variable energy_dyn_v : unsigned(ENERGY_BITS-1 downto 0);
        variable lambda_v     : unsigned(15 downto 0);
        variable rely_v       : unsigned(15 downto 0);
        variable all_done_v   : boolean;
        variable any_ready_v  : boolean;
        variable found_ready_v: boolean;
    begin
        if reset = '1' then
            state          <= S_IDLE;
            sim_clk        <= (others => '0');
            done_vec       <= (others => '0');
            assign_vec     <= (others => '0');
            acc_energy     <= (others => '0');
            makespan       <= (others => '0');
            total_energy   <= (others => '0');
            schedule_done  <= '0';
            cur_task       <= 0;
            best_proc      <= 0;
            best_finish    <= (others => '1');
```

```
            for i in 0 to MAX_TASKS-1 loop
                tasks(i).state       <= TASK_IDLE;
                tasks(i).assigned_to <= 0;
                tasks(i).start_time  <= (others => '0');
                tasks(i).end_time    <= (others => '0');
                tasks(i).priority    <= (others => '1');
                tasks(i).is_backup   <= '0';
                tasks(i).original_id <= 0;
                task_finish(i)       <= (others => '0');
            end loop;

            for p in 0 to MAX_PROC-1 loop
                proc_states(p) <= PROC_IDLE;
                proc_timers(p) <= (others => '0');
            end loop;

        elsif rising_edge(clk) then
            case state is

                -- ------------------------------------------------
                when S_IDLE =>
                    schedule_done <= '0';
                    if start = '1' then
                        state   <= S_INIT;
                        sim_clk <= (others => '0');
                    end if;

                -- ------------------------------------------------
                -- Ініціалізація: задачі без попередників → READY
                when S_INIT =>
                    for i in 0 to MAX_TASKS-1 loop
                        tasks(i).state <= TASK_IDLE;
                        done_vec(i)    <= '0';
                    end loop;
                    -- Задача 0 (F1) є кореневою — ставимо READY
                    tasks(0).state <= TASK_READY;
                    state <= S_FIND_READY;
                    cur_task <= 0;

                -- ------------------------------------------------
                -- Пошук наступної готової задачі для планування
                when S_FIND_READY =>
                    found_ready_v := false;
                    sim_clk <= sim_clk + 1;

                    -- Оновлення таймерів процесорів і перевірка
завершення
                    for p in 0 to MAX_PROC-1 loop
                        if proc_states(p) = PROC_BUSY then
                            if proc_timers(p) = to_unsigned(1,
TIME_BITS) then
                                -- Задача завершена
                                proc_states(p) <= PROC_IDLE;
```

```
                                proc_timers(p) <= (others => '0');
                                -- Відмічаємо відповідну задачу як
DONE
                                for t in 0 to MAX_TASKS-1 loop
                                    if tasks(t).assigned_to = p and
                                       tasks(t).state = TASK_RUNNING
then
                                        tasks(t).state <= TASK_DONE;
                                        done_vec(t)    <= '1';
                                        task_finish(t) <= sim_clk;
                                        -- Активуємо наступників
                                        for s in 0 to MAX_TASKS-1
loop
                                            if adj(t,s) = '1' and
                                               tasks(s).state =
TASK_IDLE then
                                                tasks(s).state <=
TASK_READY;
                                            end if;
                                        end loop;
                                    end if;
                                end loop;
                            elsif proc_timers(p) > to_unsigned(1,
TIME_BITS) then
                                proc_timers(p) <= proc_timers(p) -
1;
                            end if;
                        end if;
                    end loop;

                    -- Шукаємо першу задачу у стані READY
                    for t in 0 to MAX_TASKS-1 loop
                        if tasks(t).state = TASK_READY and not
found_ready_v then
                            cur_task     <= t;
                            found_ready_v := true;
                        end if;
                    end loop;

                    if found_ready_v then
                        best_finish <= (others => '1');
                        best_proc   <= 0;
                        state       <= S_EVAL_PROC;
                    else
                        -- Перевіряємо чи всі задачі завершено
                        all_done_v := true;
                        for t in 0 to MAX_TASKS-1 loop
                            if tasks(t).state /= TASK_DONE then
                                all_done_v := false;
                            end if;
                        end loop;
                        if all_done_v then
                            state <= S_CHECK_DONE;
```

```
                        end if;
                        -- інакше чекаємо, поки процесори не
звільняться
                    end if;

                -- ------------------------------------------------
                -- Оцінка кожного процесора для поточної задачі
                -- Критерій: мінімальний EFT (Earliest Finish Time)
                -- EFT(t,p) = max(proc_free_at(p), max_pred_finish +
comm) + B(t,p)
                when S_EVAL_PROC =>
                    for p in 0 to MAX_PROC-1 loop
                        if proc_states(p) /= PROC_OFFLINE then
                            -- Час, коли процесор звільниться
                            if proc_states(p) = PROC_BUSY then
                                proc_free_at := sim_clk +
proc_timers(p);
                            else
                                proc_free_at := sim_clk;
                            end if;

                            -- Найпізніший попередник + затримка
зв'язку
                            earliest_start := (others => '0');
                            for pred in 0 to MAX_TASKS-1 loop
                                if adj(pred, cur_task) = '1' then
                                    pred_finish :=
task_finish(pred);
                                    -- Якщо попередник на іншому
процесорі
                                    if tasks(pred).assigned_to /= p
then
                                        comm_delay := A_comm(pred,
cur_task);
                                    else
                                        comm_delay := (others =>
'0');
                                    end if;
                                    if pred_finish + comm_delay >
earliest_start then
                                        earliest_start :=
pred_finish + comm_delay;
                                    end if;
                                end if;
                            end loop;

                            if proc_free_at > earliest_start then
                                earliest_start := proc_free_at;
                            end if;

                            -- EFT = earliest_start + B(cur_task, p)
                            candidate_finish := earliest_start +
B_matrix(cur_task, p);
```

```
                        -- Зберігаємо найкращий варіант
                        if candidate_finish < best_finish then
                            best_finish <= candidate_finish;
                            best_proc   <= p;
                        end if;
                    end if;
                end loop;
                state <= S_ASSIGN_TASK;

            -- ------------------------------------------------
            -- Призначення задачі на найкращий процесор
            when S_ASSIGN_TASK =>
                tasks(cur_task).state       <= TASK_ASSIGNED;
                tasks(cur_task).assigned_to <= best_proc;
                -- Зберігаємо призначення у вектор
                assign_vec(cur_task*2+1 downto cur_task*2) <=
                    std_logic_vector(to_unsigned(best_proc, 2));
                state <= S_DISPATCH;

            -- ------------------------------------------------
            -- Запуск задачі на процесорі
            when S_DISPATCH =>
                tasks(cur_task).state      <= TASK_RUNNING;
                tasks(cur_task).start_time <= sim_clk;
                proc_states(best_proc)     <= PROC_BUSY;
                -- Завантажуємо таймер
                proc_timers(best_proc)     <= B_matrix(cur_task,
best_proc);

                -- Оновлюємо накопичену енергію (спрощений
розрахунок)
                -- Udyn = ndyn * freq * B  (беремо ndyn =
get_ndyn, freq=255)
                energy_dyn_v :=
                    resize(get_ndyn(ndyn_proc, best_proc) *
                           get_freq(freq_proc, best_proc) *
                           B_matrix(cur_task, best_proc),
ENERGY_BITS);
                acc_energy <= acc_energy +
energy_dyn_v(ENERGY_BITS-1 downto 0);

                state <= S_FIND_READY;

            -- ------------------------------------------------
            -- Перевірка чи всі задачі виконані
            when S_CHECK_DONE =>
                -- Знаходимо makespan = max finish time
                makespan <= (others => '0');
                for t in 0 to MAX_TASKS-1 loop
                    if task_finish(t) > makespan then
                        makespan <= task_finish(t);
                    end if;
```

```
                    end loop;
                    total_energy  <= acc_energy;
                    state         <= S_FINISH;

                -- ------------------------------------------------
                when S_FINISH =>
                    schedule_done <= '1';
                    state         <= S_IDLE;

            end case;
        end if;
    end process;
end architecture rtl;

-- ============================================================
--  МОДУЛЬ 8: Верхній рівень системи (Top-Level)
--  Інтегрує всі модулі та забезпечує взаємодію з AXI
-- ============================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.scheduler_pkg.all;

entity fpga_hetero_scheduler_top is
    port (
        -- Тактовий сигнал та скидання
        clk              : in  std_logic;
        reset_n          : in  std_logic;  -- активний LOW

        -- AXI-Lite інтерфейс (спрощено до регістрів керування)
        -- Запис конфігурації та зчитування результатів
        axi_awvalid      : in  std_logic;
        axi_awaddr       : in  std_logic_vector(7 downto 0);
        axi_wvalid       : in  std_logic;
        axi_wdata        : in  std_logic_vector(31 downto 0);
        axi_bready       : in  std_logic;
        axi_bvalid       : out std_logic;
        axi_arvalid      : in  std_logic;
        axi_araddr       : in  std_logic_vector(7 downto 0);
        axi_rvalid       : out std_logic;
        axi_rdata        : out std_logic_vector(31 downto 0);

        -- Сигнали стану
        led_schedule_done: out std_logic;
        led_rely_ok      : out std_logic;
        led_busy         : out std_logic
    );
end entity fpga_hetero_scheduler_top;

architecture rtl of fpga_hetero_scheduler_top is

    signal reset_sync    : std_logic;
```

```
    -- Сигнали DAG ініціалізатора
    signal B_matrix_s    : time_matrix_t;
    signal adj_s         : adj_matrix_t;
    signal A_comm_s      : comm_matrix_t;
    signal dag_ready_s   : std_logic;

    -- Сигнали планувальника
    signal sched_start   : std_logic;
    signal assign_out    : std_logic_vector(MAX_TASKS*2-1 downto 0);
    signal task_done_out : std_logic_vector(MAX_TASKS-1 downto 0);
    signal makespan_out  : unsigned(TIME_BITS-1 downto 0);
    signal energy_out    : unsigned(ENERGY_BITS-1 downto 0);
    signal sched_done    : std_logic;
    signal sim_time_out  : unsigned(TIME_BITS-1 downto 0);

    -- Сигнали динамічного резервування
    signal rely_mat_s    : rely_matrix_t;
    signal backup_needed_s : std_logic_vector(MAX_TASKS-1 downto 0);
    signal backup_proc_s   : std_logic_vector(MAX_TASKS*2-1 downto
0);
    signal total_rely_s    : unsigned(15 downto 0);
    signal rely_ok_s       : std_logic;
    signal redund_done_s   : std_logic;
    signal redund_enable_s : std_logic;

    -- Параметри процесорів (з Таблиці 2.1)
    -- P1: λ=0.01 Q8≈3, P2: λ=0.008 Q8≈2, P3: λ=0.012 Q8≈3
    signal lambda_proc_s : std_logic_vector(MAX_PROC*8-1 downto 0)
        := x"03_02_03";
    -- Ndyn: P1=0.8→Q8=205, P2=1.1→Q8=281, P3=1.0→Q8=256
    signal ndyn_proc_s   : std_logic_vector(MAX_PROC*8-1 downto 0)
        := x"CD_19_00";
    signal nstat_proc_s  : std_logic_vector(MAX_PROC*8-1 downto 0)
        := x"0D_0A_0F"; -- ~0.05,0.04,0.06 у Q8
    -- Частоти: всі = 1.0 → Q8=256=0xFF
    signal freq_proc_s   : std_logic_vector(MAX_PROC*8-1 downto 0)
        := x"FF_FF_FF";

    -- Поріг надійності Rreq = 0.8 → Q15 ≈ 26214
    signal rely_thresh_s : unsigned(15 downto 0)
        := to_unsigned(26214, 16);

    -- Ініціалізована матриця надійностей (спрощено: фіксовані
значення)
    -- В реальній системі обчислюється через reliability_calc
    function init_rely_matrix return rely_matrix_t is
        variable m : rely_matrix_t;
    begin
        -- Lji = Q15 appr. для кожної задачі та процесора
        -- exp(-λ*B): P1 λ≈0.01, P2 λ≈0.008, P3 λ≈0.012
        -- Наприклад, задача 0: B01=14, B02=6, B03=9
        -- Lji ≈ exp(-0.01*14)=0.869, exp(-0.008*6)=0.953, exp(-
0.012*9)=0.898
```

```
        m(0,0) := to_unsigned(28474, 16); -- 0.869
        m(0,1) := to_unsigned(31227, 16); -- 0.953
        m(0,2) := to_unsigned(29425, 16); -- 0.898
        m(1,0) := to_unsigned(28736, 16); -- 0.877
        m(1,1) := to_unsigned(31555, 16); -- 0.963
        m(1,2) := to_unsigned(29786, 16); -- 0.909
        m(2,0) := to_unsigned(30245, 16); -- 0.923
        m(2,1) := to_unsigned(30810, 16); -- 0.940
        m(2,2) := to_unsigned(29098, 16); -- 0.888
        m(3,0) := to_unsigned(30057, 16); -- 0.917
        m(3,1) := to_unsigned(27525, 16); -- 0.840
        m(3,2) := to_unsigned(30515, 16); -- 0.931
        m(4,0) := to_unsigned(29458, 16); -- 0.899
        m(4,1) := to_unsigned(27852, 16); -- 0.850
        m(4,2) := to_unsigned(29851, 16); -- 0.911
        m(5,0) := to_unsigned(27525, 16); -- 0.840
        m(5,1) <= to_unsigned(30057, 16); -- 0.917
        m(5,2) := to_unsigned(30810, 16); -- 0.940
        m(6,0) := to_unsigned(25690, 16); -- 0.784
        m(6,1) := to_unsigned(25903, 16); -- 0.790
        m(6,2) := to_unsigned(25477, 16); -- 0.777
        m(7,0) := to_unsigned(27525, 16); -- 0.840
        m(7,1) := to_unsigned(27131, 16); -- 0.828
        m(7,2) := to_unsigned(24412, 16); -- 0.745
        m(8,0) := to_unsigned(24773, 16); -- 0.756
        m(8,1) := to_unsigned(31555, 16); -- 0.963
        m(8,2) := to_unsigned(28212, 16); -- 0.861
        return m;
    end function;

    -- AXI регістри
    signal ctrl_reg    : std_logic_vector(31 downto 0);
    signal status_reg  : std_logic_vector(31 downto 0);

begin
    -- Синхронне скидання (активний LOW → перетворюємо)
    reset_sync <= not reset_n;

    -- ------------------------------------------------------
    -- Екземпляр DAG ініціалізатора
    -- ------------------------------------------------------
    U_DAG: entity work.dag_init
        port map(
            clk      => clk,
            reset    => reset_sync,
            B_matrix => B_matrix_s,
            adj      => adj_s,
            A_comm   => A_comm_s,
            ready    => dag_ready_s
        );

    -- ------------------------------------------------------
    -- Екземпляр головного планувальника HRLHS
```

```
-- ------------------------------------------------------
U_SCHED: entity work.hrlhs_scheduler
    port map(
        clk            => clk,
        reset          => reset_sync,
        start          => sched_start,
        B_matrix       => B_matrix_s,
        adj            => adj_s,
        A_comm         => A_comm_s,
        lambda_proc    => lambda_proc_s,
        ndyn_proc      => ndyn_proc_s,
        nstat_proc     => nstat_proc_s,
        freq_proc      => freq_proc_s,
        rely_threshold => rely_thresh_s,
        assignment     => assign_out,
        task_done_vec  => task_done_out,
        makespan       => makespan_out,
        total_energy   => energy_out,
        schedule_done  => sched_done,
        sim_time       => sim_time_out
    );

-- ------------------------------------------------------
-- Екземпляр модуля динамічного резервування
-- ------------------------------------------------------
rely_mat_s <= init_rely_matrix;

U_REDUNDANCY: entity work.dynamic_redundancy
    port map(
        clk              => clk,
        reset            => reset_sync,
        enable           => redund_enable_s,
        rely_matrix      => rely_mat_s,
        assignment       => assign_out,
        rely_threshold   => rely_thresh_s,
        backup_needed    => backup_needed_s,
        backup_proc      => backup_proc_s,
        total_reliability => total_rely_s,
        rely_ok          => rely_ok_s,
        done             => redund_done_s
    );

-- ------------------------------------------------------
-- AXI-Lite керування (спрощений регістровий інтерфейс)
-- Addr 0x00 [0]: start, [1]: enable_redundancy
-- Addr 0x04:     поріг надійності (rely_threshold)
-- Addr 0x10 RO:  status [0]=done,[1]=rely_ok,[2]=busy
-- Addr 0x14 RO:  makespan
-- Addr 0x18 RO:  total_energy
-- Addr 0x1C RO:  assignment vector (задачі 0-7)
-- ------------------------------------------------------
process(clk, reset_sync)
begin
```

```
        if reset_sync = '1' then
            axi_bvalid    <= '0';
            axi_rvalid    <= '0';
            axi_rdata     <= (others => '0');
            sched_start   <= '0';
            redund_enable_s <= '0';
        elsif rising_edge(clk) then
            sched_start     <= '0';
            redund_enable_s <= '0';

            -- Запис регістрів
            if axi_awvalid = '1' and axi_wvalid = '1' then
                case axi_awaddr is
                    when x"00" =>
                        ctrl_reg    <= axi_wdata;
                        sched_start <= axi_wdata(0);
                        redund_enable_s <= axi_wdata(1);
                    when others => null;
                end case;
                axi_bvalid <= '1';
            else
                axi_bvalid <= '0';
            end if;

            -- Читання регістрів
            if axi_arvalid = '1' then
                axi_rvalid <= '1';
                case axi_araddr is
                    when x"10" =>
                        axi_rdata <= (others => '0');
                        axi_rdata(0) <= sched_done;
                        axi_rdata(1) <= rely_ok_s;
                        axi_rdata(2) <= dag_ready_s;
                    when x"14" =>
                        axi_rdata <= std_logic_vector(
                            resize(makespan_out, 32));
                    when x"18" =>
                        axi_rdata <= std_logic_vector(
                            resize(energy_out, 32));
                    when x"1C" =>
                        axi_rdata <= assign_out(31 downto 0);
                    when x"20" =>
                        axi_rdata <= (others => '0');
                        axi_rdata(MAX_TASKS-1 downto 0) <=
task_done_out;
                    when x"24" =>
                        axi_rdata <= (others => '0');
                        axi_rdata(MAX_TASKS-1 downto 0) <=
backup_needed_s;
                    when x"28" =>
                        axi_rdata <= std_logic_vector(
                            resize(sim_time_out, 32));
                    when others =>
```

```
                        axi_rdata <= x"DEADBEEF";
                end case;
            else
                axi_rvalid <= '0';
            end if;
        end if;
    end process;

    -- LED індикація
    led_schedule_done <= sched_done;
    led_rely_ok       <= rely_ok_s;
    led_busy          <= not sched_done and dag_ready_s;

end architecture rtl;


-- ============================================================
--  TESTBENCH: Верифікація системи планування
-- ============================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity tb_fpga_hetero_scheduler is
end entity tb_fpga_hetero_scheduler;

architecture sim of tb_fpga_hetero_scheduler is

    signal clk      : std_logic := '0';
    signal reset_n  : std_logic := '0';

    signal axi_awvalid : std_logic := '0';
    signal axi_awaddr  : std_logic_vector(7 downto 0) := (others =>
'0');
    signal axi_wvalid  : std_logic := '0';
    signal axi_wdata   : std_logic_vector(31 downto 0) := (others =>
'0');
    signal axi_bready  : std_logic := '1';
    signal axi_bvalid  : std_logic;
    signal axi_arvalid : std_logic := '0';
    signal axi_araddr  : std_logic_vector(7 downto 0) := (others =>
'0');
    signal axi_rvalid  : std_logic;
    signal axi_rdata   : std_logic_vector(31 downto 0);

    signal led_done : std_logic;
    signal led_rely : std_logic;
    signal led_busy : std_logic;

    constant CLK_PERIOD : time := 10 ns; -- 100 MHz

begin
    -- Генератор тактового сигналу
```

```
    clk <= not clk after CLK_PERIOD/2;

    -- Екземпляр тестованого пристрою (DUT)
    DUT: entity work.fpga_hetero_scheduler_top
        port map(
            clk               => clk,
            reset_n           => reset_n,
            axi_awvalid       => axi_awvalid,
            axi_awaddr        => axi_awaddr,
            axi_wvalid        => axi_wvalid,
            axi_wdata         => axi_wdata,
            axi_bready        => axi_bready,
            axi_bvalid        => axi_bvalid,
            axi_arvalid       => axi_arvalid,
            axi_araddr        => axi_araddr,
            axi_rvalid        => axi_rvalid,
            axi_rdata         => axi_rdata,
            led_schedule_done => led_done,
            led_rely_ok       => led_rely,
            led_busy          => led_busy
        );

    -- Стимул процес
    process
    begin
        -- Скидання системи
        reset_n <= '0';
        wait for 5 * CLK_PERIOD;
        reset_n <= '1';
        wait for 10 * CLK_PERIOD;

        -- Запуск планувальника та активація динамічного
резервування
        -- Запис у регістр 0x00: start=1, enable_redundancy=1
        axi_awaddr  <= x"00";
        axi_awvalid <= '1';
        axi_wdata   <= x"00000003"; -- bit0=start,
bit1=redundancy_enable
        axi_wvalid  <= '1';
        wait for CLK_PERIOD;
        axi_awvalid <= '0';
        axi_wvalid  <= '0';
        wait for CLK_PERIOD;

        -- Очікуємо завершення планування (максимум 2000 тактів)
        wait for 2000 * CLK_PERIOD;

        -- Зчитуємо статус
        axi_araddr  <= x"10";
        axi_arvalid <= '1';
        wait for CLK_PERIOD;
        axi_arvalid <= '0';
        wait for CLK_PERIOD;
```

```
        -- Зчитуємо makespan
        axi_araddr  <= x"14";
        axi_arvalid <= '1';
        wait for CLK_PERIOD;
        axi_arvalid <= '0';
        wait for CLK_PERIOD;

        -- Зчитуємо енергоспоживання
        axi_araddr  <= x"18";
        axi_arvalid <= '1';
        wait for CLK_PERIOD;
        axi_arvalid <= '0';
        wait for CLK_PERIOD;

        -- Зчитуємо вектор призначень
        axi_araddr  <= x"1C";
        axi_arvalid <= '1';
        wait for CLK_PERIOD;
        axi_arvalid <= '0';
        wait for CLK_PERIOD;

        -- Зчитуємо задачі, що потребують резервування
        axi_araddr  <= x"24";
        axi_arvalid <= '1';
        wait for CLK_PERIOD;
        axi_arvalid <= '0';
        wait for CLK_PERIOD;

        report "=== Симуляція HRLHS планувальника завершена ==="
severity note;
        report "LED_DONE=" & std_logic'image(led_done)
               & " LED_RELY_OK=" & std_logic'image(led_rely)
severity note;

        wait;
    end process;
end architecture sim;
```

……

ДОДАТОК Б
(обов'язковий)

Тези доповіді та сертифікат учасника конференції


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КНУ
ім. Криворізький
XIX ВСЕУКРАЇНСЬКА НАУКОВО-ПРАКТИЧНА WEB-КОНФЕРЕНЦІЯ
АСПІРАНТІВ, СТУДЕНТІВ ТА МОЛОДИХ ВЧЕНИХ
МАТЕРІАЛИ
КОНФЕРЕНЦІЇ
CONFERENCE PROCEEDINGS
КОМП'ЮТЕРНІ ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ ТА МЕРЕЖІ
COMPUTER INTELLIGENT SYSTEMS AND NETWORKS
KICM-2026
CISN-2026
КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ
25-27 БЕРЕЗНЯ 2026
КРИВИЙ РІГ / KRYVYI RIH

ЗМІСТ

Н. В. Величко,
С.М. Лисенко, д.т.н., професор,
Хмельницький національний університет

МЕТОД ПЛАНУВАННЯ ТА РОЗПОДІЛУ ЗАДАЧ У ГЕТЕРОГЕННИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ

Запропоновано метод призначений для планування виконання взаємопов'язаних обчислювальних завдань у розподіленій системі з неоднорідними процесорами. Метод складається з двох взаємопов'язаних етапів: побудови енергоефективного розкладу виконання завдань та підвищення надійності системи за допомогою динамічного резервування. Основою методу є поєднання евристичного пошуку з адаптивним навчальним механізмом для уточнення розкладу.

Планування та розподіл задач у гетерогенних обчислювальних системах є актуальним через необхідність ефективного використання різнорідних обчислювальних ресурсів для підвищення продуктивності, енергоефективності та масштабованості сучасних обчислювальних платформ [1, 2]. Для вирішення такої задачі було запропоновано метод, корки якого подано нижче.

На початковому етапі виконується формалізація обчислювального процесу. Сукупність взаємопов'язаних завдань представляється у вигляді орієнтованого ациклічного графа, де вершини відповідають окремим обчислювальним задачам, а дуги визначають залежності між ними та необхідність передачі даних. Така модель дозволяє визначити порядок виконання операцій і забезпечує коректне врахування залежностей між завданнями. Кожне завдання може мати кілька попередників та кілька наступників, а його виконання можливе лише після завершення всіх попередніх операцій.

Одночасно з побудовою графової моделі формується опис обчислювальних ресурсів системи. Для кожного процесора задаються параметри продуктивності, час виконання окремих завдань та характеристики енергоспоживання. Оскільки система містить неоднорідні процесори, одне й те саме завдання може виконуватися на різних пристроях із різною швидкістю та різними витратами енергії. Урахування цих параметрів дає змогу оцінити ефективність призначення завдань конкретним процесорам.

Після формування моделі задачі виконується побудова початкового розкладу виконання завдань. На цьому етапі генерується множина можливих варіантів призначення завдань процесорам і визначається порядок їх виконання. Кожен такий варіант розглядається як окреме рішення, що характеризується певним значенням загального енергоспоживання та часу виконання. Мета цього етапу полягає у формуванні початкового набору перспективних рішень, які можуть бути використані для подальшої оптимізації.

Подальший етап передбачає оптимізацію сформованих розкладів. Для кожного варіанта розраховується загальне споживання енергії системою. Воно визначається як сума динамічної та статичної складових. Динамічна складова пов'язана з безпосереднім виконанням завдань на процесорах, тоді як статична складова відображає енергію, що споживається процесорами під час перебування у робочому стані. Такий підхід дозволяє врахувати як витрати енергії на обчислення, так і витрати, пов'язані з тривалістю роботи обладнання.

Під час оцінювання варіантів планування враховуються залежності між завданнями. Якщо два пов'язані завдання виконуються на різних процесорах, до часу виконання додається затримка передачі даних між ними. Якщо ж вони виконуються на одному процесорі, враховується лише час обчислення. Таким чином, вибір процесора для кожного завдання здійснюється з урахуванням не лише енергоспоживання, але й додаткових витрат часу на передавання даних між вузлами системи.

У процесі оптимізації параметри пошуку змінюються залежно від етапу виконання алгоритму. На початкових ітераціях основна увага приділяється дослідженню простору можливих рішень, що дозволяє виявити перспективні варіанти розподілу завдань. У подальшому алгоритм переходить до

детальнішого аналізу знайдених рішень і їх уточнення. Це забезпечує поєднання глобального пошуку оптимальних рішень із локальним вдосконаленням розкладу.

Для підвищення ефективності планування використовується механізм адаптивного уточнення розкладу. На основі результатів попередніх ітерацій система коригує стратегію призначення завдань процесорам. Рішення, що забезпечують менше споживання енергії та прийнятний час виконання, отримують вищу оцінку і використовуються як основа для формування наступних варіантів планування. Завдяки такому підходу алгоритм поступово наближається до оптимального розкладу виконання завдань.

Важливою складовою методу є перевірка дотримання часових обмежень. Для кожного сформованого розкладу визначається сумарний час виконання всіх завдань робочого процесу. Якщо цей час перевищує встановлений термін виконання, відповідний варіант планування відхиляється. У протилежному випадку він може бути використаний як допустиме рішення. Це дозволяє гарантувати своєчасне виконання обчислювального процесу.

Після завершення оптимізації формується остаточний розклад виконання завдань. Для кожного завдання визначаються процесор виконання, час початку обчислення та момент завершення операції. Отриманий розклад забезпечує виконання всіх залежностей між завданнями, дотримання часових обмежень та мінімізацію загального споживання енергії системою.

На наступному етапі здійснюється оцінювання надійності виконання завдань. Для кожного процесора визначається ймовірність його безвідмовної роботи протягом часу виконання завдання. На основі цих параметрів обчислюється надійність виконання кожної задачі, а також загальна надійність виконання всього робочого процесу.

Якщо отримане значення надійності є недостатнім, застосовується механізм динамічного резервування. Його сутність полягає у створенні резервних копій окремих завдань, які мають підвищений ризик відмови. Резервні завдання призначаються на інші процесори, відмінні від тих, на яких виконуються основні обчислення. У разі збою основного процесора резервне завдання може бути виконане альтернативним обчислювальним вузлом, що дозволяє зберегти безперервність роботи системи.

Під час формування резервного розкладу враховуються пріоритети завдань та доступність обчислювальних ресурсів. Завдання з високим пріоритетом або низькою надійністю отримують більший рівень захисту, тоді як для менш критичних завдань обсяг резервування може бути обмежений. Такий підхід дозволяє збалансувати витрати ресурсів та рівень відмовостійкості системи.

На рисунках 1 та 2 подано приклад діаграми Ганта для планування завдань робочого процесу та діаграму послідовності планування завдань.

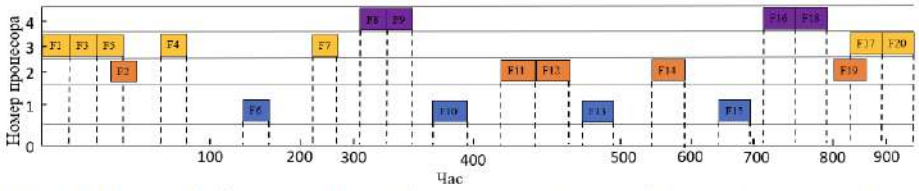


Рис. 1. Приклад діаграми Ганта для планування завдань робочого процесу

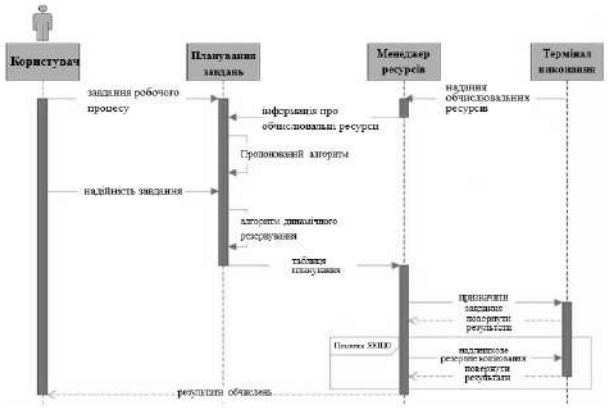


Рис. 2. Діаграма послідовності планування завдань

ВИСНОВКИ

Метод планування складається з послідовних етапів побудови графової моделі завдань, пошуку енергоефективного розкладу за допомогою поєднання евристичного пошуку та адаптивного навчання, перевірки часових обмежень і підвищення надійності системи шляхом динамічного резервування критичних завдань. Метод дозволяє ефективно розподіляти обчислювальні ресурси, зменшувати енергоспоживання системи та забезпечувати стабільність виконання обчислювальних процесів.

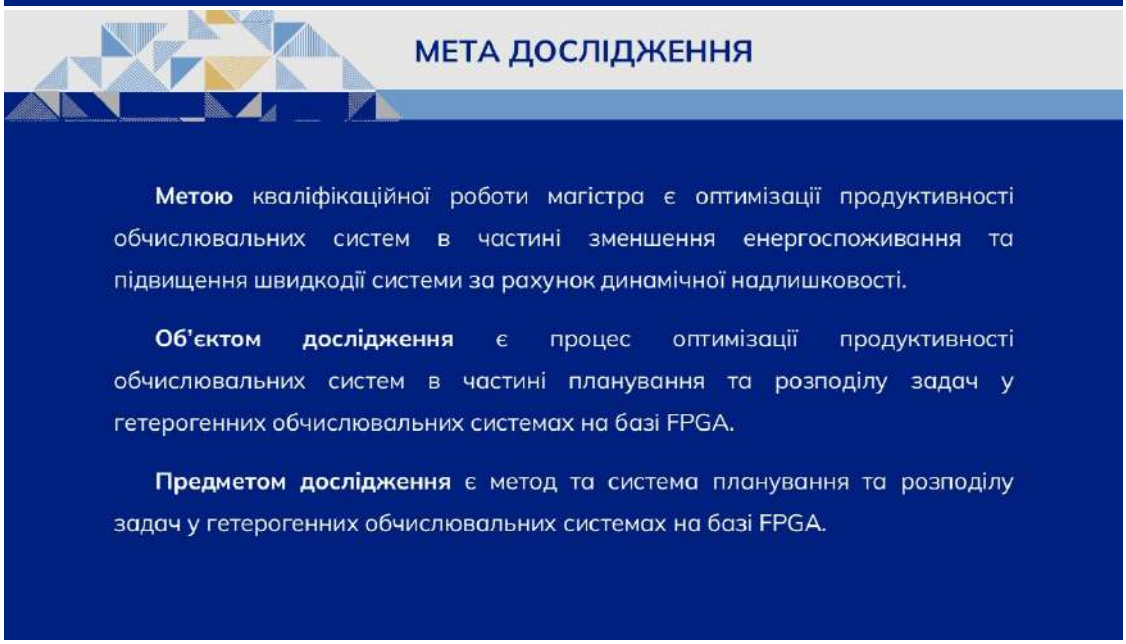
ЛІТЕРАТУРА

1. Ghorbian M., Ghobaei-Arani M., Esmaeili L. A survey on the scheduling mechanisms in serverless computing: a taxonomy, challenges, and trends. Cluster Computing. 2024. DOI: 10.1007/s10586-023-04264-8.

2. Aslani A., Ghobaei-Arani M. Machine learning inference serving models in serverless computing: a survey. Computing. Vol. 107. 2025. DOI: 10.1007/s00607-024-01377-9.

ДОДАТОК В
(обов'язковий)

Презентація



ЗАДАЧІ ДОСЛІДЖЕННЯ

Поставлена мета досягається розв'язанням таких основних завдань:

- проаналізувати відомі методи оптимізації продуктивності обчислювальних систем;
- розробити метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA;
- здійснити дослідження методу планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA;
- реалізувати метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA у вигляді системи.

НАУКОВА НОВИЗНА ТА ПРАКТИЧНА ЦІННІСТЬ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Наукова новизна отриманих результатів:

- удосконалено метод енергоефективного планування задач у гетерогенних паралельних системах, який на відміну від відомих поєднує евристичну оптимізацію та навчання з підкріпленням із урахуванням залежностей задач і неоднорідності ресурсів, і який дає змогу зменшити енергоспоживання та підвищити швидкодію системи за рахунок динамічної надлишковості;
- набула подальшого розвитку система планування задач у гетерогенних паралельних системах на основі FPGA.

АКТУАЛЬНІСТЬ ДОСЛІДЖЕННЯ

- В умовах зростання обсягів даних, ускладнення алгоритмів обробки та посилення вимог до швидкодії й енергоефективності, обчислювальні системи дедалі частіше будуються як гетерогенні платформи, що поєднують процесори загального призначення та спеціалізовані прискорювачі.
- Це дає змогу адаптувати виконання різних фрагментів задачі до різних моделей обчислень і до різних ресурсних обмежень, однак водночас ускладнює керування обчислювальним процесом.
- Ключовою проблемою стає планування та розподіл задач між різнорідними виконавцями з урахуванням обмежень пропускної здатності введення-виведення, доступності пам'яті, особливостей залежностей між задачами, а також сервісних вимог щодо затримки та пропускної здатності.

АНАЛІЗ ВІДОМИХ МЕТОДІВ

- Встановлено, що ефективність функціонування таких систем визначається не лише обчислювальною продуктивністю окремих компонентів, а й організацією пам'яті, характеристиками міжкомпонентного обміну даними, політиками диспетчеризації та здатністю системи адаптуватися до різних типів навантаження.
- FPGA займає особливе місце серед ресурсів гетерогенних систем, оскільки поєднує реконфігурованість, можливість глибокої конвеєризації та високий рівень паралелізму обробки даних.
- Аналіз сучасних методів планування засвідчив, що найбільш перспективними є адаптивні, багатокритеріальні та інтелектуальні підходи, які враховують динаміку стану ресурсів, структуру задач, вимоги до QoS, обмеження за затримкою, вартістю й надійністю.

МЕТОД ПЛАНУВАННЯ ТА РОЗПОДІЛУ ЗАДАЧ У ГЕТЕРОГЕННИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ

1. Формування початкового розкладу
- Моделювання задачі
- Урахування залежностей, ресурсів, часу та енергії
- Побудова базового розкладу

2. Глобальна оптимізація
- Ініціалізація популяції рішень
- Ітеративне оновлення позицій частинок
- Пошук мінімального енергоспоживання

3. Локальна адаптація
- Вибір ресурсів за політикою
- Оновлення Q-функції
- Адаптивне покращення розкладу

МЕТОД ПЛАНУВАННЯ ТА РОЗПОДІЛУ ЗАДАЧ У ГЕТЕРОГЕННИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ

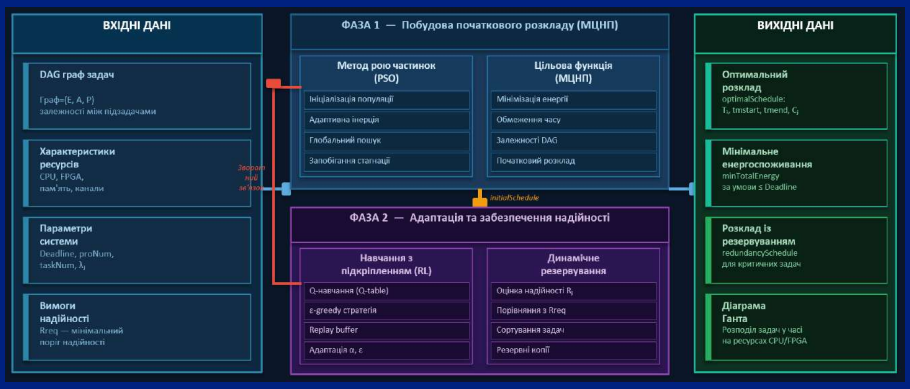
4. Оцінка надійності
- Аналіз ймовірності безвідмовного виконання задач
- Визначення критичних задач

5. Динамічне резервування
- Дублювання/перепризначення ненадійних задач
- Використання альтернативних ресурсів

6. Формування фінального розкладу
- Баланс: енергія – час – надійність
- Результат: оптимізований розклад виконання

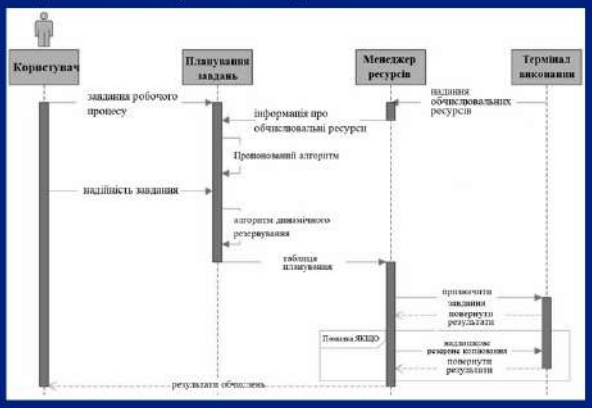
МЕТОД ПЛАНУВАННЯ ТА РОЗПОДІЛУ ЗАДАЧ У ГЕТЕРОГЕННИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ

Схема методу



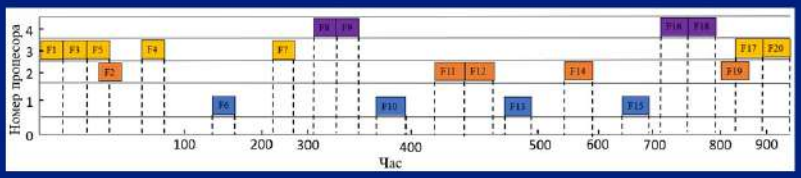
МЕТОД ПЛАНУВАННЯ ТА РОЗПОДІЛУ ЗАДАЧ У ГЕТЕРОГЕННИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ

Діаграма послідовності планування завдань



МЕТОД ПЛАНУВАННЯ ТА РОЗПОДІЛУ ЗАДАЧ У ГЕТЕРОГЕННИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ

Приклад діаграми Ганта для планування завдань робочого процесу



РЕАЛІЗАЦІЯ МЕТОДУ

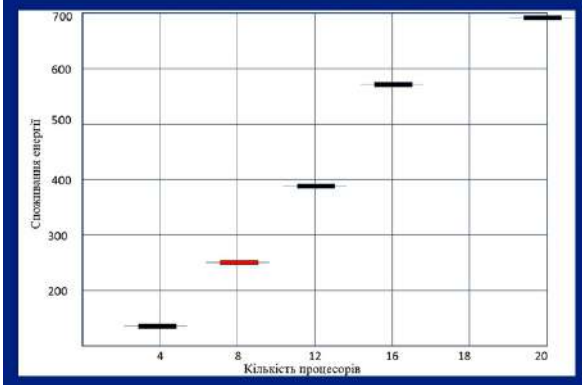
Загальна структурна схема системи реалізації методу на базі FPGA

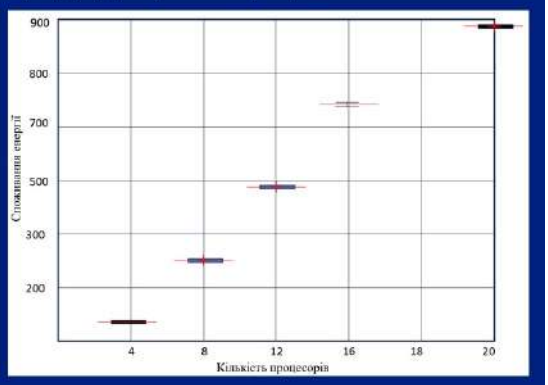
РЕАЛІЗАЦІЯ МЕТОДУ

Архітектура реалізації методу планування та розподілу задач у гетерогенній системі на базі FPGA

ЕКСПЕРИМЕНТИ

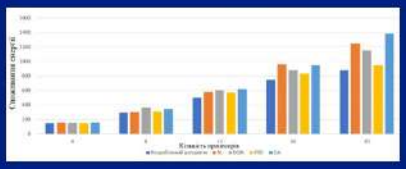
Статистичний аналіз споживання енергії для двох робочих процесів

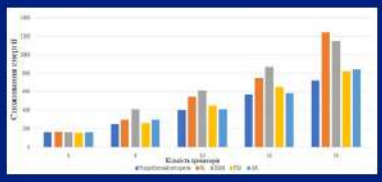


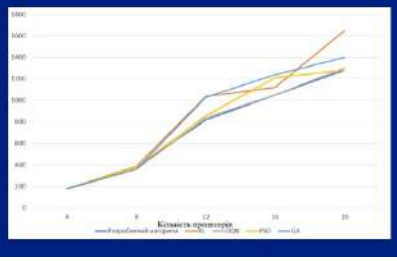


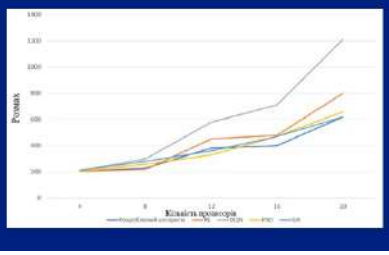
ЕКСПЕРИМЕНТИ

Порівняння споживання енергії та часу відгуку між запропонованим методом та традиційним методом за двох робочих процесів: а) робочий процес швидкого перетворення Фур'є»; б) робочий процес обробки графа





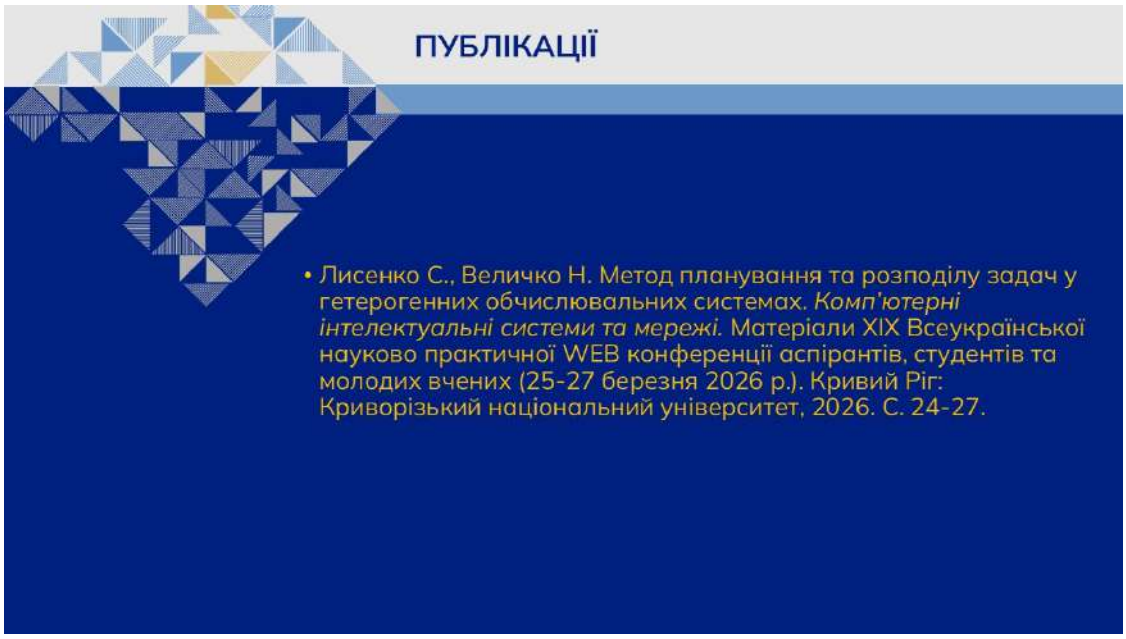




ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень:

- проаналізовано відомі методи оптимізації продуктивності обчислювальних систем;
- розроблено метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA;
- здійснено дослідження методу планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA;
- реалізовано метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA у вигляді системи.
- Запропонований метод забезпечує нижче енергоспоживання при збереженні прийнятного часу відгуку в діапазоні 4–20 процесорів. Підтверджено перевагу динамічного планування над статичними підходами. Досягнуто зниження енергоспоживання на 14,3%, 45,1% та 26,8% порівняно з відомими методами, що доводить ефективність запропонованого методу.


ПУБЛІКАЦІЇ
• Лисенко С., Величко Н. Метод планування та розподілу задач у гетерогенних обчислювальних системах. *Комп'ютерні інтелектуальні системи та мережі.* Матеріали XIX Всеукраїнської науково практичної WEB конференції аспірантів, студентів та молодих вчених (25-27 березня 2026 р.). Кривий Ріг: Криворізький національний університет, 2026. С. 24-27.

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Назарій ВЕЛИЧКО

Співавтор:

Назва: Метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA

Експерт: Сергій ЛИСЕНКО

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:12.2%

Коефіцієнт подібності 2:3.9%

Мікропробіли: 3

Заміна букв: 16

Інтервали: 0

Білі знаки: 6

Дата створення звіту: 2026-04-16 12:45:15.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2026-04-16 Доцент Андрій Нічепорук

Дата експерт

Anti-Plagiarism (http://ap.km.ua) v-15.701

Максимальне співпадіння з одним документом 41.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 11%**

	Документ		Сумарний збіг по Базі Даних	
ID: 270520 Назва: МКР Метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA Додано в БД: 2026-04-17 Автора: Назарій ВЕЛИЧКО Керівники: Сергій ЛИСЕНКО Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	179234	1104	74415 (42%)	455 (41%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
269814	Назва: Звіт з ПДП Метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA Додано в БД: 2026-03-13 Автора: Величка Н.В. Керівники: Аскеров В.В Консультанти: Опоненти:	73362 (41.0%)	462 (42.0%)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Здобувач: Назарій ВЕЛИЧКО

Тема: Метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи магістра:

Кількість листів креслень – ; кількість сторінок записки 96

1. Короткий зміст роботи та прийнятих рішень У роботі запропоновано систему, яка здійснює оптимізацію планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA

2. Висновок про відповідність роботи дипломному завданню Кваліфікаційна робота магістра відповідає виданому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі здійснено аналіз предметної області планування та розподілу задач у гетерогенних обчислювальних системах. У другому розділі здійснено формалізацію обчислювального процесу та побудовано комплексну модель системи планування задач у гетерогенних обчислювальних середовищах. У третьому розділі розроблено метод планування та розподілу задач у гетерогенних обчислювальних системах на базі програмованих логічних інтегральних схем. У четвертому розділі розроблено та описано програмно-технічний засіб реалізації методу планування та розподілу задач у гетерогенних обчислювальних системах на базі програмованих логічних інтегральних схем.

4. Позитивні сторони роботи: Удосконалено метод енергоефективного планування задач у гетерогенних паралельних системах, який на відміну від відомих поєднує евристичну оптимізацію та навчання з підкріпленням із урахуванням залежностей задач і неоднорідності ресурсів, і який дає змогу зменшити

енергоспоживання та підвищити швидкодію системи за рахунок динамічної надлишковості.

5. Негативні сторони роботи: В роботі присутні певні логічні помилки щодо опису моделі пропонованої системи

6. Оцінка графічного оформлення та пояснювальної записки роботи: –

7. Відгук про роботу в цілому: В загальному робота виконана на високому рівні.

8. Інші зауваження: –

9. Оцінка кваліфікаційної роботи магістра:

Розглянувши позитивні та негативні сторони представленої кваліфікаційної роботи магістра вважаю, що робота заслуговує оцінки «добре» 85.00 (B)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) д.т.н., професор, Бармак Олександр Володимирович, завідувач кафедри комп'ютерних наук

" 1 травня " 2026р.

Зав. кафедри КІІС
д-р. філософії Ользі ПАВЛОВІЙ

Назарій ВЕЛИЧКО

ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи КІ2м-24-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів академічної відповідальності, ознайомлений (а). Про використання спеціалізованих програмних засобів (СПЗ) StrikePlagiarism та Anti-Plagiarism для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а). Надаю університету право на передачу моєї роботи для обробки та збереження в базах даних СПЗ і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються СПЗ.

Також надаю свою згоду на обробку й збереження університетом моєї роботи в Інституційному репозитарії Хмельницького національного університету.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

1 травня 2026 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи Метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA
Автор Назарій ВЕЛИЧКО
Освітня програма Комп'ютерна інженерія та програмування
Рівень вищої освіти другий (магістерський)
Спеціальність 123 Комп'ютерна інженерія
Науковий керівник: д.т.н., професор Сергій ЛИСЕНКО

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	відповідає
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
2) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;
3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.
4) значна частина знайденого плагіату відноситься до списку використаних джерел
5) збіг зі своєю роботою: Звіт з переддипломної практики «Метод планування та розподілу задач у гетерогенних обчислювальних системах на базі FPGA», який додано в базу даних ХНУ 2026-03-13 автора Величка Н.В. (41%).

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 12.2% і адресується до 77 першоджерела; та системою Anti-Plagiarism складає 41%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

25.04.2026

Завідувач кафедри _____ Підпис — Ольга ПАВЛОВА (Ім'я, ПРІЗВИЩЕ)

Гарант освітньої програми _____ Підпис — Олег САВЕНКО (Ім'я, ПРІЗВИЩЕ)

Керівник кваліфікаційної роботи _____ Підпис — Сергій ЛИСЕНКО (Ім'я, ПРІЗВИЩЕ)