МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

На правах рукопису

КЛИМЕНКО Ірина Анатоліївна

УДК 004.274

МЕТОДИ ТА ЗАСОБИ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ОБРОБКИ ІНФОРМАЦІЇ В РЕКОНФІГУРОВНИХ КОМП'ЮТЕРНИХ СИСТЕМАХ НА БАЗІ ПЛІС

Спеціальність 05.13.05 – комп'ютерні системи та компоненти
Дисертація на здобуття наукового ступеня
доктора технічних наук

Науковий консультант
Луцький Георгій Михайлович
доктор технічних наук,
професор

Киев – 2017

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БКД	–	бібліотека конфігураційних даних
БКФК	–	блок керування формуванням команд
БМК	–	блок мікропрограмного керування
БПК	–	блок буферної пам’яті команд
БФК	–	блок формування команди
ГА	–	граф алгоритму
КС	–	комутаційне середовище
ЛПК	–	локальна пам’ять конфігурацій
ОМ	–	обчислювальний модуль
ПАРС	–	пристрій автоматичного розподілу завдань і синхронізації
ПДД	–	пам’ять з довільним доступом
ПКС	–	пам’ять керувальних слів
ПЛІС	–	програмовні логічні інтегральні схеми
ПСФК	–	паралельне середовище формування команд
ПФМ	–	потоковий функціональний модуль
РК	–	регістр команди
РКС	–	реконфігуровна комп’ютерна система
САПР	–	система автоматизації проектування
СЛАР	–	система лінійних алгебричних рівнянь
СФК	–	середовище формування команд
ФБ	–	функціональний блок
ЯПФ	–	ярусно-паралельна форма

ВСТУП

Дисертаційну роботу присвячено напряму створення високопродуктивних паралельних комп'ютерних систем, які забезпечують високу користувацьку ефективність під час розв'язання широкого класу задач керування великої розмірності, зокрема в режимі реального часу. Функціональним і технологічним підґрунтям для створення таких систем є сучасна динамічно перепрограмовна елементна база – програмовні логічні інтегральні схеми (ПЛІС), що підтримують технологію часткової динамічної реконфігурації.

Актуальність теми. Широко розповсюджені класи високопродуктивних паралельних комп'ютерних систем характеризуються фіксованою структурою обчислювального середовища. Це не дозволяє ефективно відображувати алгоритми сильнозв'язаних задач, структура яких не відповідає структурі комп'ютерної системи.

Традиційними напрямами підвищення ефективності розв'язання задач з високою кількістю інформаційних зв'язків є побудова паралельних комп'ютерних систем з реконфігуровними каналами зв'язків між процесорами, таких як *NUMA* (*Non-Uniform Memory Access*) системи, трансп'ютери, мультипроцесори. Сучасними тенденціями підвищення ефективності є збільшення «зерна обчислень» і використання мультиядерних процесорів у вузлах паралельних обчислювальних систем (високопродуктивні кластери, суперкомп'ютери). Але більшість відомих методів та засобів підвищення ефективності паралельних комп'ютерних систем сприяють логарифмічному приросту користувацької ефективності для розв'язання задач з високою кількістю інформаційних обмінів.

Актуальним напрямом підвищення ефективності паралельної обробки інформації є побудова реконфігуровних комп'ютерних систем на програмовній елементній базі ПЛІС. Основною концепцією підвищення ефективності таких систем є сумісний програмно-апаратний підхід, який ґрунтується на апаратному пришвидшенні певних функцій, виконання яких

неефективне засобами універсальних процесорів. Застосування технології ПЛІС дозволяє реалізувати в одному кристалі мікросхеми обчислювальну систему будь-якої складності з гнучкою архітектурою. Можливість перепрограмування ПЛІС на фізичному рівні забезпечує швидке налаштування обчислювальної структури для реалізації довільних алгоритмів обчислювальних задач. Така концепція здатна забезпечити баланс лінійного приросту користувацької ефективності, підтримання ефективних технологій паралельного програмування, широку функціональність і економічну привабливість.

Відомі дотепер реконфігуровні обчислювальні системи ґрунтуються на статичній реконфігурації кристалів ПЛІС, що зумовлює широке використання методів та засобів статичного планування обчислювального процесу і розподілу задач на обчислювальне середовище. Статична структура дозволяє ефективно використовувати в межах цих систем відомі методи паралельного програмування та методи і засоби підвищення ефективності, але в більшості випадків значні накладні витрати часу на статичну перебудову обчислювальної структури і непродуктивні енергетичні й апаратні витрати можуть бути сумірними з апаратним пришвидшенням, що здатне зневілювати ефект від використання реконфігуровних обчислювальних систем. Зокрема ця проблема стає актуальною для виконання обчислювальних задач, на час розв’язання яких накладаються жорсткі часові обмеження.

В останні роки поява ПЛІС, що динамічно реконфігуруються, створила передумови і нові можливості для підвищення ефективності паралельних комп’ютерних систем за рахунок значного зменшення накладних витрат часу на перепрограмування кристала ПЛІС, а також можливості реконфігурації обчислювальної структури в динамічному режимі (*Run Time*).

Ефективність функціонування динамічно реконфігуровних комп’ютерних систем, зокрема в межах розв’язання задач з великою кількістю інформаційних обмінів, великої і надвеликої розмірності, на розв’язання яких накладаються часові обмеження, пов’язана з рядом проблем. Проблеми

функціональних обмежень спричинені високими непродуктивними витратами часу і продуктивності у процесі динамічної реконфігурації обчислювальної структури. Накладні витрати зумовлені затримками передавання даних для налаштування ПЛІС і невисокою швидкодією їх інтерфейсів. Важливою є проблема апаратурних обмежень реконфігуровних обчислювальних ресурсів ПЛІС, що критично для реалізації мультизадачного режиму функціонування незважаючи на надвисокий ступінь інтеграції сучасних ПЛІС. Для вирішення проблеми мінімізації непродуктивних витрат, забезпечення апаратного прискорення і ефективного використання обчислювального простору ПЛІС в умовах часових обмежень функціонального процесу важливою стає проблема оптимізації процесу обробки інформації, що зводиться до визначення оптимального співвідношення вимог розв’язуваних задач і структури реконфігуровного обчислювального середовища. Вирішенню означених проблем присвячено дисертаційну роботу.

Традиційні технології паралельних обчислень орієнтовані на фіксоване обчислювальне середовище без будь-яких просторових і функціональних обмежень. Використання існуючих технологій паралельного програмування як і традиційних методів та засобів підвищення ефективності паралельних обчислень не є ефективним для вирішення поставленої проблеми підвищення ефективності динамічно реконфігуровних комп’ютерних систем.

У зв'язку з цим дисертаційна робота становить науковий і практичний інтерес. Вирішується актуальна проблема розвитку теорії організації динамічно адаптивних паралельних обчислень у комп’ютерних системах, що мають обмеження за своїми функціональними й апаратурними можливостями, зумовлені використанням динамічно програмовної елементної бази.

Зв’язок роботи з науковими програмами, планами, темами. Дисертаційна робота виконувалася відповідно до планів науково-дослідних робіт кафедри обчислювальної техніки НТУУ «КПІ»: «Методи та засоби підвищення ефективності рішення задач на основі перестроюваних обчислювальних засобів на ПЛІС», № ДР 0114U000547, 2014 – 2015 рр. та

«Створення засобів проектування та розробка на їх основі високопродуктивних процесорів систем технічного зору», № ДР 0115U002326, 2015 – 2016 рр.

Мета і завдання дослідження. Метою роботи є розвиток теорії організації комп'ютерних систем на ПЛІС за рахунок підвищення ефективності процесу відображення задач на динамічно реконфігуровне обчислювальне середовище і оптимізації процесу обробки інформації з урахуванням співвідношення вимог розв'язуваних задач та структури обчислювального середовища.

Для досягнення поставленої мети в роботі вирішуються такі завдання:

– аналіз структурної і функціональної організації реконфігуровних комп'ютерних систем з метою визначення вимог до методів і засобів підвищення ефективності обробки інформації;

– визначення критеріїв ефективності функціонування динамічно реконфігуровних комп'ютерних систем та критеріїв ефективної реалізації алгоритмів обчислювальних задач;

– розроблення математичних моделей оцінювання значень критеріїв ефективності динамічно реконфігуровних комп'ютерних систем та оцінювання часу виконання функціональних процесів обробки інформації;

– розроблення концепції структурної організації динамічно реконфігуровних комп'ютерних систем з метою зменшення затримок передавання конфігураційних даних і затримок, які вносить операційна система у процесі реалізації динамічного відображення задач на реконфігуровне обчислювальне середовище;

– розроблення методів і засобів зменшення накладних витрат процесу реконфігурації обчислювального середовища з метою оптимізації процесу динамічного відображення задач на реконфігуровне обчислювальне середовище в комп'ютерних системах, що мають функціональні та апаратурні обмеження;

– розроблення способу визначення оптимального співвідношення параметрів розв’язуваних задач і структури реконфігуровного обчислювального середовища з метою підвищення ефективності паралельної обробки інформації в комп’ютерних системах, що мають функціональні та апаратурні обмеження;

– розроблення нових та вдосконалення існуючих методів і засобів взаємодії основних складових елементів структурного рівня та комунікаційних процесів, підвищення надійності та відмовостійкості з метою підвищення ефективності обробки інформації у динамічно реконфігуровних комп’ютерних системах.

Об'єктом дослідження є процес взаємної адаптації розв’язуваних задач і обчислювального середовища, який породжує проблеми підвищення ефективності обробки інформації в динамічно реконфігуровних комп’ютерних системах шляхом зменшення накладних витрат процесу відображення задач на реконфігуровне обчислювальне середовище і визначення зернистості обчислень.

Предмет дослідження: методи та засоби підвищення ефективності обробки інформації в динамічно реконфігуровних комп’ютерних системах за рахунок зменшення накладних витрат процесу реконфігурації обчислювального середовища і оптимізації процесу обробки інформації з урахуванням функціональних і апаратурних обмежень реконфігуровних комп’ютерних систем.

Методи досліджень. Для розроблення і дослідження структури та методів і засобів організації обробки інформації у динамічно реконфігуровних комп’ютерних системах використовуються положення теорії обчислювальних систем, теорії керування, теорії нечітких множин, а також елементи теорії інформації, графів та алгоритмів, методів моделювання. Аналіз паралельних обчислювальних процесів та похибок обчислень виконується із застосуванням елементів математичного аналізу, теорії числових методів, імітаційного та математичного моделювання.

Наукова новизна одержаних результатів полягає у вирішенні актуальної проблеми розвитку теорії організації обробки інформації в комп'ютерних системах з урахуванням їх функціональних та апаратурних обмежень, що включає в себе взаємозв'язані вирішення завдань оптимізації процесу обробки інформації з урахуванням співвідношення параметрів розв'язуваних задач і структури обчислювального середовища та зменшення накладних витрат процесу відображення задач на реконфігуровне обчислювальне середовище.

Основні наукові результати, отримані особисто автором:

– уперше розроблено математичні моделі функціональних процесів, що на відміну від відомих ураховують багаторівневу структуру динамічно реконфігуровних комп'ютерних систем і непродуктивні витрати на всіх рівнях їх функціонування, які дозволяють оцінити процес обробки інформації з метою його оптимізації за запропонованим інтегральним критерієм, що ґрунтується на співвідношенні непродуктивного часу, параметрів розв'язуваних задач і структури обчислювального середовища з урахуванням його апаратурних обмежень;

– запропоновано новий спосіб організації віртуальної пам'яті в реконфігуровних комп'ютерних системах, який відрізняється від відомих багаторівневою багатофункціональною структурою для зберігання, пошуку і керування конфігураційними даними, що забезпечує реалізацію різних стратегій обслуговування завдань, які відрізняються частотою їх виконання, непродуктивним часом та обсягом використання апаратурних ресурсів ПЛІС.

– запропоновано новий спосіб скорочення критичного часу виконання обчислювальних задач, який на відміну від відомих інтегрує статичний і динамічний підходи для трансформації графів обчислювальних задач з використанням модифікованого методу гілок і границь, що забезпечує інтенсивне скорочення непродуктивних витрат процесу відображення задач на динамічно реконфігуровне обчислювальне середовище;

– запропоновано й обґрунтовано новий метод адаптивного відображення задач на реконфігуровне обчислювальне середовище, який відрізняється від відомих скороченням критичного часу виконання обчислювальних задач на базі багаторівневого кешування конфігураційних даних, що дає змогу реалізувати ефективні стратегії організації процесу відображення задач за обсягом непродуктивного часу і апаратурних витрат з урахуванням несталих умов відображення;

– запропоновано й обґрунтовано новий метод оптимізації процесу відображення задач на реконфігуровне обчислювальне середовище, що відрізняється від відомих визначенням стратегії обслуговування кожного завдання на підставі аналізу показника його апаратного пришвидшення, який шляхом оптимізації непродуктивного часу за розробленим у роботі інтегральним критерієм забезпечує потрібний час для виконання обчислювальних задач, підвищує ефективність використання апаратурних ресурсів ПЛІС і скорочує кількість відхилень виконання завдань;

– уперше визначено й обґрунтовано критерій швидкодії паралельного обчислювального середовища на ПЛІС, який на відміну від відомих ґрунтується на визначенні співвідношення часу виконання завдань на апаратурі ПЛІС і обсягу корисних даних для обчислення з урахуванням затримок поширення сигналів на фізичному рівні функціональних блоків і просторових обмежень обчислювального середовища, що дає змогу оцінити ефективність паралельної обробки інформації на фізичному рівні реконфігуровних комп’ютерних систем для розв’язання задач великої розмірності;

– запропоновано нову стратегію організації процесу обробки інформації, яка відрізняється від відомих взаємною адаптацією розв’язуваних задач і обчислювального середовища на підставі визначення оптимальної зернистості обчислень за критерієм максимальної швидкодії паралельного обчислювального середовища на ПЛІС, що дає змогу забезпечити екстремуми цільових функцій оптимізації обробки даних на всіх рівнях динамічно

реконфігуровних комп’ютерних систем, що в цілому призводить до підвищення їх користувацької ефективності;

– удосконалено структурну організацію бібліотеки функціональних блоків на ПЛІС, яка відрізняється від відомих формуванням набору функціональних блоків з оптимальними характеристиками, що дає змогу варіювати зернистістю обчислювального середовища під час розв’язання задач великої розмірності на підставі запропонованого критерію швидкодії паралельного обчислювального середовища на ПЛІС, що забезпечує підвищення ефективності паралельної обробки інформації у динамічно реконфігуровних обчислювальних системах і розширення їх функціональних можливостей;

– запропоновано й обґрунтовано нові рівні абстракцій розгляду архітектури реконфігуровних комп’ютерних систем, які на відміну від відомих локалізують процес реконфігурації обчислювального середовища на структурному рівні обчислювальних модулів, що забезпечує підвищення ефективності обробки інформації через застосування на кожному рівні ефективних методів та засобів організації обчислень, зменшення складності та скорочення накладних витрат процесу відображення задач на реконфігуровне обчислювальне середовище;

– модифіковано метод автоматичного розподілу завдань і синхронізації процесів на базі моделі обчислень, керованих потоком даних, який відрізняються від відомих механізмом завчасної реконфігурації обчислювального середовища, дворівневою організацією синхронізації процесів і вдосконаленим способом підвищення відмовостійкості керувального ядра, що забезпечує підвищення ефективності обробки даних на локальному рівні обчислювального модуля за рахунок автоматичного скорочення накладних витрат, зменшення складності й апаратного пришвидшення функціональних процесів, а також підвищення надійності реконфігуровних комп’ютерних систем.

Практичне значення одержаних результатів. Розроблено нові методи і засоби організації структури та процесів обробки інформації в реконфігуровних комп'ютерних системах, побудованих на ПЛІС.

Практичне значення результатів роботи полягає у тому, що запропонована структурна організація реконфігуровних комп'ютерних систем та нова стратегія взаємної адаптації розв'язуваних задач і структури обчислювального середовища дозволяють підвищити користувацьку ефективність паралельних комп'ютерних систем і реалізувати масштабовані високопродуктивні паралельні комп'ютерні системи з можливістю динамічної адаптації до вимог широких класів задач. Математичні моделі, методи й засоби доведені до практичної реалізації у вигляді програмних продуктів і промислових зразків. Результати роботи можуть бути використані для розроблення нових і вдосконалення наявних високопродуктивних паралельних комп'ютерних систем.

Отримані практичні результати полягають в такому:

– децентралізація і локалізація засобів керування реконфігуровними обчислювальними ресурсами на структурному рівні однотипних обчислювальних модулів дозволяють пришвидшити процес обробки інформації, розвантажити операційну систему від розв'язання неспецифічних задач, зменшити рух даних на міжмодульному рівні, а також забезпечити мобільність і універсальність технологій, спростити масштабування обчислювальної потужності;

– оптимізація процесу відображення задач на структуру реконфігуровних комп'ютерних систем розширює їх функціональні можливості через забезпечення широкого класу задач керування ефективною цільовою обчислювальною структурою;

– багаторівнева структура пам'яті та апаратні засоби для реалізації пошуку і підтримання даних дають змогу реалізовувати апаратні багатовимірні бази даних і апаратні системи керування базами даними, що сприяє підвищенню ефективності збереження й обробки інформації в

спеціалізованих обчислювальних системах для розв’язання задач керування в обмеженому режимі часу;

– використання поверхні динамічної ділянки ПЛІС для кешування функціональних блоків дозволяє вирішити проблему зберігання ресурсів внутрішньої пам’яті ПЛІС у широких класах реконфігуровних комп’ютерних систем;

– використання імітаційної моделі реконфігуровної комп’ютерної системи дає змогу виділяти реконфігуровні обчислювальні ресурси в режимі часу, наближеному до реального, що є зручним інструментом для моделювання і дослідження часових характеристик функціональних процесів обробки інформації у динамічно реконфігуровних комп’ютерних системах;

– багаторівнева архітектура реконфігуровних комп’ютерних систем розширює їх функціональні можливості через реалізацію розподілених неоднорідних обчислювальних структур, а також забезпечує розв’язання обчислювальних задач, що мають самоподібний характер зокрема через реалізацію циклічних і фрактальних моделей динамічних процесів і структур;

– запропоновані засоби автоматичної синхронізації процесів на структурному рівні обчислювального модуля дозволяють удосконалити традиційні технології паралельного програмування, які пропонують виробники ПЛІС, та підвищити ефективність синхронізації процесів у реконфігуровних комп’ютерних системах класу системи-на-кристалі.

Теоретичні і практичні результати дисертаційної роботи впроваджено у ТОВ «Науково-виробничий комплекс «Головне підприємство обробки польотної інформації «АВІАЦІЙНІ ТЕХНОЛОГІЇ» для підвищення ефективності обробки інформації в розподілених комп’ютерних системах під час виконання завдань керування в режимі реального часу та Національному технічному університеті «Київський політехнічний інститут імені Ігоря Сікорського» на кафедрах обчислювальної техніки, автоматизованих систем обробки інформації і управління, системного програмування і спеціалізованих комп'ютерних систем та в аспірантурі КПІ імені Ігоря Сікорського в

навчальний процес (використовуються під час проведення лекційних і лабораторних занять, курсового і дипломного проектування). Зазначені впровадження підтверджено відповідними актами.

Особистий внесок здобувача. Основні результати дисертації отримано здобувачем самостійно. У роботах, що опубліковані у співавторстві, здобувачеві належать: [1] – спосіб адаптивного скорочення критичного часу виконання обчислювальних задач у реконфігуровних обчислювальних системах, що керуються потоком даних; [2] – архітектурна концепція побудови процесорного ядра на ПЛІС з динамічно реконфігуровною системою команд для реалізації моделі обчислень, керованих потоком даних; [3, 34] – стратегія взаємної адаптації розв'язуваних задач і обчислювального середовища в реконфігуровних комп'ютерних системах, спосіб визначення оптимальної зернистості під час розв'язання задач великої розмірності, критерії ефективності обчислювального середовища на ПЛІС; [4] – новий підхід до організації процесу обробки даних, що ґрунтується на застосуванні завчасної реконфігурації, та спосіб його реалізації для реконфігуровних комп'ютерних систем, що керуються потоком даних; [6] – метод пришвидшення реконфігурації та концепція розроблення його програмної моделі та структури емулятора динамічно реконфігуровної комп'ютерної системи; [7, 35, 36] – методи пришвидшення реконфігурації, способи їх реалізації та концепція розроблення програмно-апаратних моделей реконфігуровних обчислювальних систем; [10] – математичні моделі визначення часу реконфігурації при повторному використанні апаратних ресурсів; [11] – метод оптимізації структури графу задачі та спосіб трансформації ярусно-паралельної форми графу з урахуванням апаратурних обмежень ПЛІС; [12] – багаторівнева структура віртуального адресного простору і спосіб зберігання та підтримання конфігурацій апаратних задач на поверхні реконфігуровної ділянки ПЛІС; [13] – сучасна класифікація реконфігуровних комп'ютерних систем, напрями підвищення їх ефективності, нова концепція побудови архітектури реконфігуровних комп'ютерних систем,

що ґрунтується на парадигмі динамічної реконфігурації; [15, 40] – концепція вдосконалення базових технологій проектування систем-на-кристалі, апаратні засоби підвищення ефективності мультипроцесорних обчислювальних систем, створених на базі стандартних технологій; [17, 46, 47] – апаратна реалізація на ПЛІС бібліотечних модулів апаратних задач та функціональних елементів структурного рівня для виконання реконфігуровних обчислень; [19, 26] – математичні та апаратні моделі багатовимірних обчислювальних задач для ефективної реалізації засобами реконфігуровних комп'ютерних систем; [20, 44] – метод динамічного аналізу графічної інформації та апаратні засоби автоматизації введення в обчислювальну систему вихідних графів алгоритмів обчислювальних задач; [18, 22] – метод реконфігурації та відновлення системи у разі відмови функціональних блоків для паралельних обчислювальних систем, керованих потоком даних, та її апаратна модель на ПЛІС; [23, 27 – 32, 43, 45] – методи та апаратні засоби вдосконалення структури та принципів взаємодії функціональних елементів структурного рівня реконфігуровних комп'ютерних систем; [24, 25] – концепція та спосіб вдосконалення технологій *GRID* через реалізацію розподілених неоднорідних обчислювальних структур на ПЛІС.

Апробація результатів дисертації. Матеріали дисертації доповідались та обговорювались на: Міжнародній конференції «*Infocom Advanced Solution* 2015» (м. Київ, 2015 р.); III і IX Міжнародних науково-технічних конференціях «Комп'ютерні системи і мережні технології (*CSNT* 2016)» (м. Київ, 2010, 2016 рр.); VII Міжнародній науковій конференції «Сучасні проблеми математичного моделювання, прогнозування та оптимізації (ОПТІМА 2016)» (м. Кам'янець-Подільський, 2016 р.); Науковій конференції студентів, магістрантів та аспірантів «Інформатика та обчислювальна техніка, ІОТ-2016» (м. Київ, 2016 р.); XXII Міжнародній конференції з автоматичного управління «Автоматика – 2015» (м. Одеса, 2015 р.); Всеукраїнській науково-практичній конференції «Перспективні напрямки сучасної електроніки, інформаційних та комп'ютерних систем (MEICS-2015)» (м. Дніпропетровськ, 2015 р.); Другій і

третій Міжнародних конференціях «Високопродуктивні обчислення (*HPC_UA*'2013)» (м. Київ, 2012 – 2013 рр.); VI Міжнародній конференції молодих вчених «Комп'ютерні науки та інженерія (*CSE*-2013)» (м. Львів, 2013 р.); Міжнародній науково-практичній конференції «Інформаційні технології в освіті, науці і техніці, (ІТОНТ-2012)» (м. Черкаси, 2012 р.); Ювілейній міжнародній науково-практичній конференції «Проектування комп'ютерних систем» (м. Київ, 2010 р.); 13-й Всеукраїнській (8-й Міжнародній) студентській науковій конференції з прикладної математики та інформатики (СНКПМІ-2010) (м. Львів, 2010 р.); Другій науковій конференції «Прикладна математика та комп'ютинг» (м. Київ, 2010 р.); Міжнародній науково-технічній конференції «Інтелектуальні технології лінгвістичного аналізу» (м. Київ, 2009 р.).

Публікації. Основні положення дисертаційної роботи опубліковано в 47 наукових працях, серед яких 26 статей у наукових фахових виданнях (у тому числі 11 статей, що реферуються міжнародними наукометричними базами даних), 1 патент на винахід, 5 патентів на корисну модель, 15 публікацій у збірниках тез доповідей науково-технічних конференцій.

Структура й обсяг дисертації. Дисертаційна робота складається зі вступу, 6 розділів, висновків і 6 додатків. Загальний обсяг роботи становить 287 сторінок друкованого тексту, 126 рисунків, 20 таблиць. Список використаної літератури містить 204 найменування.

РОЗДІЛ 1
АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ РЕКОНФІГУРОВНИХ КОМП'ЮТЕРНИХ СИСТЕМ НА БАЗІ ПЛІС

1.1. Актуальність проблеми підвищення ефективності паралельних комп'ютерних систем

Традиційні методи та засоби підвищення ефективності паралельних комп'ютерних систем засновані на інтенсивних засобах вдосконалення архітектурних рішень та програмного забезпечення та екстенсивних засобах збільшення кількості і підвищення складності обчислювальних вузлів, поліпшення їх часових характеристик, удосконалення технологічних характеристик елементної бази та мережевих засобів.

У результаті застосування інтенсивних засобів підвищення ефективності паралельні обчислювальні системи еволюціонували як універсальні високопродуктивні обчислювальні кластери з підтриманням стандартизованих операційних систем, що сприяло їх високій практичній й економічній привабливості. Застосування екстенсивних засобів підвищення ефективності забезпечило високу продуктивність сучасних паралельних обчислювальних систем за рахунок використання процесорів з багатоядерною архітектурою [1, 2] і високошвидкісних комунікаційних середовищ різних технологій та архітектур *Ethernet* [3], *InfiniBand* [4, 5].

Таким чином, сучасні високопродуктивні обчислювальні системи характеризуються надвисокою декларованою піковою продуктивністю і застосовуються для виконання складних трудомістких обчислювальних задач у різних предметних галузях. Це породжує актуальну проблему підвищення користувацької ефективності сучасних паралельних обчислювальних систем в широкому класі задач. Проблема полягає в тому, що висока ефективність обчислень досягається в основному під час вирішення слабкозв'язаних задач. Під час розв'язання задач з великою кількістю інформаційних обмінів

продуктивність обчислень знижується до 10% – 15% від декларованої пікової продуктивності, а дрібнозернистих задач – до 1%. [6]. Це пов’язано з жорстко фіксованою архітектурою універсальних паралельних обчислювальних систем, яка неадекватна до класів сильнозв’язаних задач.

Традиційні екстенсивні засоби підвищення ефективності паралельних обчислювальних систем зазвичай сприяють логарифмічному приросту користувацької ефективності в широкому класі задач, що обґрунтовано законом Амдала [7, 8]. Окрім того, в умовах сучасного прогресу комп'ютерних технологій та напівпровідникової техніки, який забезпечує близькі до граничних характеристики потужності процесорів і швидкості каналів передавання даних, застосування екстенсивних засобів має тенденцію до зменшення їх ефекту. Цю тенденцію натепер доведено вповільненням виконання закону Мура, яке спостерігається з 2010 р. [2, 8 – 10].

Таким чином, проблема застосування традиційних технологій підвищення ефективності паралельних обчислень на базі фіксованих обчислювальних структур є актуальною проблемою сьогодення. Це підтверджується чисельними публікаціям у літературі [8, 11 – 17], які доводять актуальність напряму підвищення ефективності паралельних комп’ютерних систем шляхом побудови гнучких обчислювальних середовищ, які адаптуються до вимог розв’язуваних класів задач. Стрімкий розвиток цього напряму з середини 2000 років зумовлений черговим технологічним проривом у галузі інтегральних схем, який забезпечив технологічну основу для створення реконфігуровних комп'ютерних систем на базі ПЛІС [13, 14, 18 – 22]. Для побудови обчислювального середовища замість універсальних процесорів використовуються ПЛІС, що дозволяє налаштовувати на апаратному рівні обчислювальної системи будь-яку функціональну структуру [13, 23]. Під реконфігурацією в цьому контексті розуміють фізичне налаштування обчислювальної структури через її перепрограмування на апаратному рівні обчислювальної системи [13, 14, 17, 23].

Теорія побудови реконфігуровних комп’ютерних систем поєднує в собі багато знань та досвід, що нагромадились останніми десятиріччями: принципи організації однорідних задачезорієнтованих обчислювальних структур, однопроцесорних і мультипроцесорних обчислень загального призначення, методи та засоби розпаралелювання процесів на рівні алгоритмів і структур, технології створення замовних інтегральних мікросхем, перепрограмованих цифрових пристроїв та систем, цифрової обробки сигналів, систем автоматизації проектування. У витоках реконфігуровних обчислень відомі праці В.М. Глушкова, В.І. Варшавського, Є.В. Євреїнова, А.В. Каляєва, Дж. Єстріна та ін. Серед сучасних учених в галузі проектування та розроблення реконфігуровних комп’ютерних систем відомі праці О.В. Палагіна, В.М. Опанасенка, О.О. Баркалова, А.О. Мельника, Р.Б. Дунця, В.Ф. Євдокимова, В.І. Жабіна, А.М. Сергієнка та ін.

Актуальність та перспективність напряму підвищення ефективності паралельних обчислювальних систем підтверджується також упровадженням реконфігуровних обчислювальних структур в обчислювальні системи та вузли відомими виробниками *Intel*, *Microsoft*, *Atmel* [24 – 26].

1.2. Огляд особливостей реалізації відомих реконфігуровних комп'ютерних систем на базі ПЛІС

1.2.1. Функціональна класифікація реконфігуровних комп’ютерних систем на базі ПЛІС. У працях [13, 17, 24, 27, 28] наведено основні визначення з предметної галузі реконфігуровних обчислювальних систем. З’ясовано, що *реконфігуровні обчислювальні системи* характеризуються наявністю обчислювальних вузлів, структура яких може перебудовуватись [29 – 31]. У сучасних реконфігуровних обчислювальних системах обчислювальні вузли зі змінною структурою називаються *реконфігуровними вузлами.* Реконфігуровні вузли реалізують на базі ПЛІС. *Реконфігурація* – це фізичне налаштування,

прошивання або програмування апаратної структури, створеної на базі ПЛІС, для виконання певної функції [17, 24].

У праці [14] наведено класифікацію реконфігуровних обчислювальних систем, основні положення якої наведено далі.

За станом системи під час реконфігурації обчислювального середовища розрізняють [14]:

– *реконфігурацію часу зупинки* (*Shutdown Reconfiguration, SD*), що відповідає зупинці системи під час реконфігурації обчислювального середовища та наявності зовнішніх засобів керування реконфігурацією обчислювального середовища;

– *реконфігурацію часу виконання* (*Run Time Reconfiguration, RTR*), що відповідає збереженню працездатності системи під час реконфігурації обчислювального середовища та реалізації керування як функціональної складової системи.

За способом виконання реконфігурації обчислювального середовища розрізняють [14]:

– *повну реконфігурацію* обчислювального середовища *(Full Reconfiguration, FR), що є* реконфігурацією кристала ПЛІС у повному обсязі;

– *часткову реконфігурацію* обчислювального середовища (*Partial Reconfiguration, PR*), що є реконфігурацією частини кристала ПЛІС.

Відома класифікація [14] всебічно відображує ситуацію на момент її створення у 2006 р., але потребує вдосконалення згідно із сучасними технічними можливостями мікросхем ПЛІС і методології проектування реконфігуровного обчислювального середовища. У праці [20] вдосконалено класифікацію, яка враховує сучасний рівень розвитку ПЛІС.

Насамперед узагальнено два широкі класи:

– реконфігуровні комп’ютерні системи (РКС) на ПЛІС [8, 19, 32], які є сучасними комп’ютерними системами, призначеними для виконання великого обсягу обчислень відповідно до загальноприйнятої теорії високопродуктивних обчислювальних систем *HPC* (*Height Performance Computer*). Натепер це

найбільш актуальна і конкурентоспроможна галузь досліджень, яка претендує на створення систем загального застосування шляхом побудови обчислювальної структури, що адаптується до класів розв'язуваних задач [20, 22, 33];

– реконфігуровні обчислювальні системи та пристрої, до яких належить широкий клас спеціалізованих обчислювальних систем, зокрема, вбудованих систем (*Embedded Systems*) [25, 28, 34] та систем-на-кристалі *SoC* (*System on a Chip*) [27]. Обчислювальні системи цього класу є задачезорієнтованими системами, що забезпечують необхідну користувачу функціональність з максимальною продуктивністю.

Основні характеристики сучасних класів реконфігуровних комп'ютерних систем з урахуванням положень відомої класифікації [14] узагальнено на рис. 1.1.

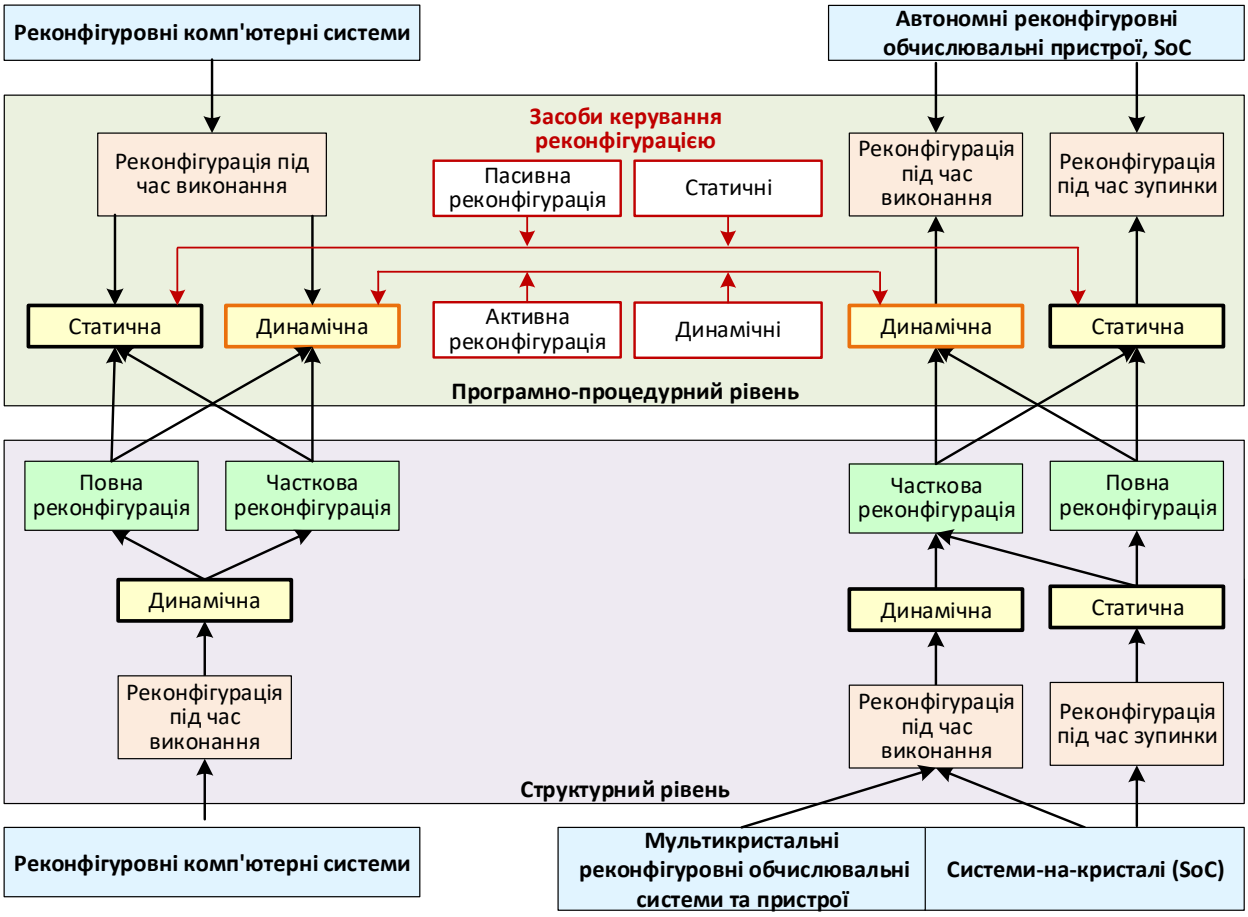


Рис. 1.1. Сучасна класифікація реконфігуровних комп'ютерних систем

У працях [13, 14, 16] подано типову організацію реконфігуровних обчислювальних систем, яка розглядається як сукупність реконфігуровного обчислювального середовища, побудованого на базі одного або декількох кристалів ПЛІС, та програмно-апаратного оточення (рис. 1.2). Реконфігурація обчислювального середовища і обчислення розглядаються як взаємопов'язані процеси, що характерно для класу реконфігуровних комп'ютерних систем. У цьому контексті для визначення поняття динамічної реконфігурації важливим аспектом є організація керування в реконфігуровних обчислювальних системах. За цим фактором класифікація [14] описує *пасивну* реконфігурацію, коли керувальний вплив на реконфігурацію відбувається з боку зовнішніх факторів, і *активну* реконфігурацію, коли реконфігурація здійснюється без будь яких зовнішніх впливів (див. рис. 1.1).

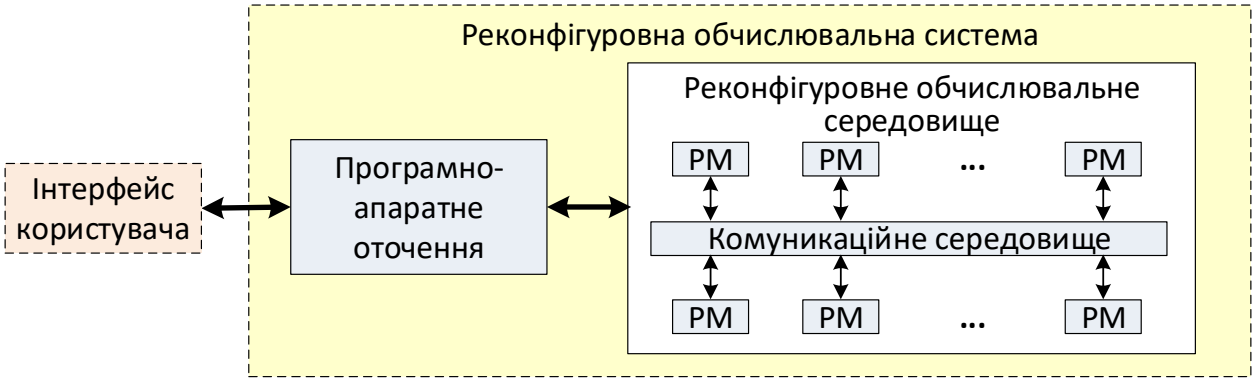


Рис. 1.2. Узагальнена структура реконфігуровної обчислювальної системи [14, 16]: РМ – реконфігуровний модуль

В удосконаленій класифікації [20] враховано сучасну технологію проектування цифрових пристроїв на ПЛІС – *часткову динамічну реконфігурацію* (*Dynamic Partial Reconfiguration, DPR*) (див. рис. 1.1). Це реконфігурація частини фізичного кристала ПЛІС у режимі часу виконання *RTR*, що забезпечує високі показники ефективності під час реконфігурації обчислювальної структури [15, 17, 35 – 39] і лежить в основі реалізації динамічної реконфігурації обчислювального середовища.

Під час аналізу літературних джерел виявлено, що відомі визначення понять статичної і динамічної реконфігурації характеризують їх на рівні фізичного процесу перепрограмування окремого кристала ПЛІС (див. рис. 1.1). Таким чином, у працях [15, 40] поняття динамічної реконфігурації визначається як реконфігурація *часу виконання Run Time*. Праці [40, 41] визначають поняття статичної реконфігурації як *реконфігурацію часу компіляції* (*Compile Time Reconfiguration, CTR*), за якої конфігурація завантажується у ПЛІС лише один раз під час ввімкнення пристрою і весь цикл роботи додатка не змінюється. Деякі автори [42] ототожнюють статичну реконфігурацію з поняттям реконфігурації часу зупинки. Повна реконфігурація у працях [40, 41] визначається як статична реконфігурація. У працях [15, 43] розглядається повна динамічна реконфігурація як *реконфігурація реального часу* (*Run Time Reconfiguration, RTR*). У працях [44 – 46] поняття часткової реконфігурації ототожнюється з поняттям динамічної реконфігурації. Наведені визначення не враховують особливості реалізації функціональних процесів у сучасному класі реконфігуровних комп'ютерних систем, що пов'язані із застосуванням технології часткової динамічної реконфігурації.

Узагальнимо тепер такі класи реконфігуровних обчислювальних систем:

– статично або динамічно реконфігуровні обчислювальні системи та пристрої з монокристальним реконфігуровним обчислювальним середовищем, наприклад, для *SoC,* коли реконфігурація часу зупинки та часу виконання стосується фактично всієї реконфігуровної системи;

– статично реконфігуровні комп'ютерні системи, для яких характерна активна або пасивна реконфігурація часу зупинки (див. рис. 1.1);

– динамічно реконфігуровні комп'ютерні системи, для яких характерна активна реконфігурація часу виконання (див. рис. 1.1).

На підставі сучасної класифікації (див. рис. 1.1) сформульовані визначення статичної та динамічної реконфігурації обчислювального середовища в реконфігуровних обчислювальних системах:

– *статична реконфігурація* (*Static Reconfiguration, SR*) – процес, що ініціалізується запитом ззовні та виконується під дією зовнішніх алгоритмів керування, належить до пасивного типу і є повністю суб'єктивним процесом, що відбувається на вимогу користувача для розв'язання певної користувацької задачі;

– *динамічна реконфігурація* (*Dynamic Reconfiguration, DR*) – процес, що характеризується автоматичним режимом здійснення незалежно від будь-яких зовнішніх впливів, коли ініціалізація і сам процес реконфігурації відбуваються під час роботи системи під дією засобів керування, що є частиною системи.

У статично реконфігуровних комп'ютерних системах зазвичай застосовуються традиційні технології паралельних обчислень, які базуються на статичних методах та засобах організації обчислювального процесу, включаючи статичне планування обчислень і розподіл задач на фіксовані обчислювальні ресурси. При цьому запуск процесів на підставі заздалегідь спланованих розкладів характерний для проблемно зорієнтованих обчислень, що залишає статично реконфігуровні комп'ютерні системи в цьому класі.

Динамічна реконфігурація – це автоматичний процес динамічного налаштування апаратної структури обчислювального середовища для виконання певних задач під час роботи системи в режимі «*RunTime*». Процес динамічної реконфігурації визначається внутрішніми факторами, що породжуються станом системи, або внутрішньою логікою керування. Відповідно до цього виникає необхідність у певній підсистемі керування, що може ініціювати реконфігурацію незалежно від зовнішніх факторів.

Згідно із класичним визначенням [13] термін «реконфігурація» поєднує як процес реконфігурації апаратної структури обчислювального середовища, так і процес обробки інформації, який виконує комп'ютер. Тому процес керування реконфігурацією обчислювального середовища зазвичай

розглядають сумісно на структурному і програмно-процедурному (абстрактному рівні операційної системи) рівнях [33, 46]. Одним із завдань програмно-процедурного рівня є *динамічне відображення задач* на обчислювальне середовище, що реконфігурується в режимі «*RunTime*». Таке обчислювальне середовище називається *реконфігуровним обчислювальним середовищем*.

Для динамічно реконфігуровних комп'ютерних систем характерні динамічні технології керування обробкою інформації, коли рішення приймаються на основі аналізу поточної інформації в процесі виконання обчислень. Це створює передумови для розширення функціональних можливостей реконфігуровних комп'ютерних систем шляхом динамічної адаптації до широких класів задач.

1.2.2. Аналіз функціональних та структурних особливостей основних класів реконфігуровних комп'ютерних систем. На підставі огляду основних класів реконфігуровних комп'ютерних систем (див. рис. 1.1) наведемо основні тенденції їх розвитку та підвищення ефективності.

Розроблення та використання спеціалізованих та вбудованих реконфігуровних обчислювальних систем і пристроїв актуальні протягом усього часу використання технологій ПЛІС. Їх розвиток починається від цифрових схем нестандартної логіки до динамічно реконфігуровних систем-на-кристалі з розширеними функціональними можливостями [25, 28, 48, 49]. Технологія часткової динамічної реконфігурації забезпечила розміщення більш складних пристроїв на меншій площині кристала, економію апаратурних ресурсів ПЛІС, динамічну адаптацію до різних класів задач, мультизадачний режим функціонування.

Розвиток реконфігуровних комп'ютерних систем на базі ПЛІС протягом тривалого часу зводився до спільно спроектованого програмно-апаратного (*HW*/*SW*) підходу [47, 50], коли пришвидшення обчислень досягається апаратною реалізацією критичних функцій. При цьому ПЛІС

використовується як обчислювальні поля для створення різного роду прискорювачів для дискретних процесорів. Такий підхід залишає реконфігуровні комп’ютерні системи в межах об’єктно-орієнтованої парадигми. Проблеми та ризики такої перспективи розглянуто в праці [48].

Реконфігуровні комп’ютерні системи та суперкомп’ютери історично пройшли шлях від повністю реконфігуровних (апаратних) обчислювальних систем до сучасних реконфігуровних комп’ютерних систем та реконфігуровних суперкомп’ютерів, які претендують на належність до класу високопродуктивних систем загального застосування [20, 22, 33, 44, 46, 47].

У працях [33, 44, 46, 47] реконфігуровні комп’ютерні системи називають реконфігуровними суперкомп'ютерами *RC* (*Reconfigurable Computers*) та високопродуктивними реконфігуровними комп'ютерами *HPRC* (*High Performance Reconfigurable Computers*).

За тенденціями розвитку та підвищення ефективності подамо відомі реконфігуровні комп’ютерні системи наступними класами і розглянемо основні особливості їх реалізації:

- комп’ютерні системи з реконфігуровними каналами зв’язку;
- апаратні реконфігуровні комп’ютерні системи;
- статично реконфігуровні комп’ютерні системи;
- динамічно реконфігуровні комп’ютерні системи.

Комп’ютерні *системи, що реконфігуруються на рівні комутації каналів зв’язку.* Традиційними напрямами підвищення ефективності розв’язання задач з високою кількістю інформаційних зв’язків є побудова паралельних комп’ютерних систем з реконфігуровними каналами зв’язку між процесорами, до яких належать системи *NUMA*, трансп’ютери, мультипроцесори [18, 29 – 31, 51].

Реконфігурація в цьому класі реконфігуровних комп’ютерів відповідає класичному уявленню про реконфігуровні обчислення, коли процес реконфігурації відбувається на рівні комутації зв’язків. У працях [8, 18, 54, 55]

описано однорідні масштабовані обчислювальні структури, що являють собою постійні обчислювальні поля на ПЛІС і динамічну систему комунікацій. Сучасні системи такого класу будують на базі ПЛІС, що дозволяє використовувати всі переваги програмовної елементної бази і забезпечити підвищення ефективності порівняно з історичними попередниками. Відомі реалізації систем означеного класу забезпечують продуктивність від 0,025 до 6 Тфлопс [55]. Сучасні сімейства комплектуються мікросхемами від *Virtex* 5 до *Virtex* 7 компанії *Xilinx* [55].

Фіксоване обчислювальне середовище залишається основною невирішеною проблемою в системах розглянутого класу. Це зумовлює необхідність зведення задачі до вимог обчислювальної системи, що потребує попереднього статичного аналізу графу обчислювального алгоритму. За вдосконаленою класифікацією [20] системи цього класу належать до статично РКС.

Апаратні реконфігуровні комп'ютерні системи на ПЛІС. Створення апаратних РКС зумовлено стрімким розвитком програмовної елементної бази середини 2000 років та ідеєю підвищення швидкодії за рахунок апаратної реалізації всього функціонала та програмного забезпечення без будь-яких жорстких архітектурних рішень [11, 56, 57]. Це означає відсутність дискретних мікропроцесорів, централізованого керування та функціонального рівня операційної системи.

Як найбільш характерний приклад у працях [11] описано реконфігуровний обчислювальний кластер *SPIRIT* (2006 р.). Розробниками вирішено такі проблеми: ефективне керування системою [11] та доступом до програмовного устаткування з боку різної кількості користувачів [59], синхронізація роботи функціональних елементів, реалізація пам'яті та файлової системи за рахунок рефакторингу файлової системи UNIX [60], рефакторинг системних бібліотек *MPI* в апаратні ядра [61], ефективна маршрутизація [62]. Результати випробувань показали лінійне збільшення

швидкодії для різних наборів даних порівняно з реалізацією на класичному обчислювальному кластері [12].

Для вирішення проблеми зручності програмування та користування апаратними РКС у праці [51] описано дослідження, присвячені рефакторингу уніфікованих широкозастосовуваних операційних систем на апаратне забезпечення. У працях [57, 63] запропоновано розширення базової операційної системи *LINUX* для підтримання апаратних ресурсів ПЛІС. Запропоновані засоби дозволяють значно спростити розроблення прикладних програм для РКС, але не забезпечують динамічної реконфігурації, оскільки операційна система *LINUX* не має власних засобів для динамічного розгалуження програм.

У результаті аналізу апаратних РКС визначено їх позитивну особливість: спеціалізація та апаратна реалізація керувальних структур і ефективних зв'язків між ними сприяє збільшенню швидкодії системи. Негативно впливають на ефективність обробки інформації [11, 48]:

– відсутність операційної системи та централізованої системи управління;

– висока структурна компонента програмування, що особливо суттєва під час програмування додатків високого рівня;

– унікальність кожної реконфігуровної обчислювальної системи потребує специфічної підготовки програмістів.

Концепція побудови апаратних реконфігуровних обчислювальних систем залишається актуальною для побудови об'єктнозорієнтованих обчислювальних систем та пристроїв [49].

Статично реконфігуровні комп'ютерні системи. Недоліки повністю реконфігуровних комп'ютерних систем усунено шляхом реалізації спільно спроектованого програмно-апаратного (*HW*/*SW*) підходу [47, 50] до організації обчислень, що ґрунтується на апаратному пришвидшенні в системі загального призначення. Принцип функціонування таких реконфігуровних систем полягає в тому, що певні секції програми розглядаються як функціональні ядра,

які замінюються оптимізованими реалізаціями на ПЛІС. Реконфігуровне обчислювальне середовище являє собою віртуальну структуру, побудовану на певній кількості кристалів ПЛІС. Технологічні можливості ПЛІС того часу зумовлюють лише повну статичну реконфігурацію за вимогою користувача [14]. Такі системи зазвичай функціонують у мультизадачному режимі, при цьому віртуальне реконфігуровне обчислювальне середовище розподіляється між користувачами для розв'язання користувацьких задач. Кожна мікросхема ПЛІС у статичному режимі [20] програмується для реалізації користувацької функції за вимогою користувача або системних засобів керування обчисленнями.

Як типові представники цього класу РКС наведемо суперкомп'ютери *MAXWELL* (2007 – 2012 р.) [19], *NOVO G* (2009 – 2012 рр.) [22], *RIVYERA* (2008 рік) [64], СКИФ (2012 р.) [4], *SRC*-7 і *SRC*-6 (2012 р.), *SGI Altix*/*RASC* (2012 р.), *Cray XT*5*h* і *Cray XD*1 (2012) [21]. Застосування цих РКС актуальне і дотепер.

У літературних джерелах подаються успішні реалізації прикладних додатків на відомих РКС [19]: моделювання цін активів у галузі фінансового інжинірингу на базі стохастичного моделювання Монте-Карло забезпечило пришвидшення більш ніж у 300 разів порівняно з використанням серійних мікропроцесорів; реалізація арифметики подвійної точності, ітераційних числових методів, операцій лінійної алгебри на великих матрицях і векторах – пришвидшення щонайменше в п'ять разів порівняно з програмною реалізацією [66]; реалізація три- і чотиривимірних візуалізацій – пришвидшення більш ніж у шість разів; високопродуктивна реалізація молекулярно-динамічного моделювання [67].

У результаті аналізу статично РКС виявлено можливість реалізації реконфігуровних комп'ютерних систем загального призначення на базі ПЛІС на відміну від загальноприйнятої думки про виключно об'єктно-орієнтовану спрямованість технологій ПЛІС. Визначено також характерні особливості РКС, що породжують проблему підвищення ефективності обробки інформації:

– апаратні обмеження обчислювальних вузлів не дозволяють

реалізовувати потужні обчислювальні структури;

– жорстка архітектура знижує користувацьку ефективність у процесі розв’язання задач з великою кількістю інформаційних обмінів;

– накладні видатки на передавання конфігураційних даних і здійснення комунікаційних з’єднань значно більші за обсяг корисних обчислень;

– недосконала мережа передавання інформації «ПЛІС-ПЛІС» не дозволяє ефективно розвантажити рівень операційної системи у процесі обміну даними між обчислювальними вузлами [19, 64]; задовільні результати дає реалізація програмованих комутаторів, таких як *Infiniband* [22, 4] або спеціалізованих комутаторів [11, 61];

– реалізовані технології не підтримують часткову динамічну реконфігурацію;

– не існує жодних автоматичних засобів структурної реалізації функцій з динамічним відображенням їх на архітектуру реконфігуровного обчислювального середовища;

– теоретично наявність універсальної операційної системи з підтриманням ефективних технологій паралельного програмування і мультикристальна структура реконфігуровного обчислювального середовища забезпечують передумови для динамічного відображення задач на реконфігуровну обчислювальну структуру [22], але особливості функціональної організації і апаратурні обмеження ПЛІС не дозволяють ефективно реалізувати цей процес;

– досі у класі РКС використовується застаріла елементна база (2008 – 2012 рр.), її оновлення неефективне.

Динамічно реконфігуровні комп’ютерні системи. У класі статично РКС систем залишається невирішеною проблема динамічного керування реконфігурацією обчислювального середовища і динамічного відображення потоків задач на динамічно реконфігуровне обчислювальне середовище. Найважливішою причиною цього є апаратурні та функціональні обмеження

використовуваних технологій ПЛІС, що підтримують лише реконфігурацію часу зупинення [14, 19, 20, 22].

Виникнення технології часткової динамічної реконфігурації відкрило нові можливості та перспективи для створення динамічно реконфігуровних обчислювальних систем. У працях [15, 68, 69] доведено, що технологія часткової динамічної реконфігурації дає змогу не тільки програмувати частину мікросхеми в режимі *Run Time*, а і значно зменшувати кількість конфігураційних даних, що передаються для налаштування обчислювального середовища. Технологія часткової динамічної реконфігурації дозволяє створювати проекти ефективні за часом реконфігурації та споживання логічних ресурсів та електроенергії, що висвітлюється в літературі дослідниками як України [14 – 16, 68, 70, 71], так і інших країн світу [40, 72, 73]. У праці [74] розглянуто питання підвищення надійності РКС на базі застосування часткової динамічної реконфігурації, а у [45, 73] – підвищення ефективності реалізації мультизадачного режиму роботи, який у РКС ґрунтується саме на застосуванні часткової динамічної реконфігурації. Це дозволяє відображувати множину задач на один і той самий пристрій динамічно, визначаючи залежності відображення в режимі *Run Time*. Динамічне перепрограмування і мультизадачний режим роботи породжують проблему динамічного планування та розподілу реконфігуровних обчислювальних ресурсів [38, 45, 69, 73].

У працях [16, 42, 71, 75, 76] описано реконфігуровні комп’ютерні системи на базі часткової динамічної реконфігурації, які змінюють власне апаратне забезпечення в динамічному режимі під дією внутрішніх алгоритмів керування – *самореконфігуровні SR* (*Self-Reconfiguration*) обчислювальні системи. Це надає їм надвисокої ефективності, гнучкості та автономності.

1.3. Аналіз відомих способів підвищення ефективності динамічно реконфігуровних комп'ютерних систем на базі ПЛІС

1.3.1. Аналіз способів організації реконфігуровного обчислювального середовища на ПЛІС. Особливості методології часткової динамічної реконфігурації описано в працях [20, 72], яка полягає в розгляді обчислювального поля ПЛІС як *статичної ділянки* для розміщення статичних компонентів та *динамічної ділянки* для розміщення реконфігуровних компонентів. Статичні компоненти ПЛІС завантажуються одноразово під час ініціалізації пристрою. Динамічні компоненти ПЛІС можуть бути реконфігуровні на різних рівнях деталізації. У працях [15, 68] розглядаються реконфігурація на рівні логічних блоків ПЛІС і модульна реконфігурація, коли реконфігурується частина ПЛІС з використанням створених окремо апаратних модулів, що додаються або замінюються під час реконфігурації. Такі структури на ПЛІС називаються *реконфігуровними модулями*.

Для конфігурації ПЛІС створюється *конфігураційний бітовий потік*. Розглядається [15, 36, 68] повний бітовий потік, створюваний для конфігурації всієї мікросхеми, та частковий бітовий потік, що відповідає тільки тій ділянці мікросхеми, яка змінюється під час реконфігурації – часткової реконфігурації.

Методологія часткової реконфігурації полягає у виділенні площини динамічної ділянці ПЛІС для розміщення реконфігуровних модулів та розподіленні зв'язків між статичними і динамічними модулями системи. Розглянемо способи реалізації часткової реконфігурації.

Реконфігурація стовпцями. Реконфігуровна ділянка складається з певної кількості стовпців, які розміщуються на всю висоту пристрою [36]. На розмір часткового бітового потоку впливає тип, тобто розмір пристрою: час реконфігурації пропорційний довжині бітового потоку і кратний висоті стовпця мікросхеми. Наприклад, конфігураційний фрейм мікросхеми *Virtex* II *XC*2V8000 становить 286 біт, а *Virtex* II *XC*2*V*40 – 26 біт [36]. Недоліком

розглянутої технології є непродуктивні витрати апаратурних ресурсів ПЛІС і неефективне розміщення внутрішніх зв’язків між модулями.

Реконфігурація блоками. Організація бітового потоку *Virtex* 4 ґрунтується на *CLB*-фреймах (*Configurable Logic Block*), структура яких має стандартний розмір – 16 логічних блоків *CLB* у висоту і 1 *CLB* у ширину – і не залежить від висоти пристрою [77]. Відповідна реконфігуровна ділянка має прямокутну форму і розмір, кратний одному конфігураційному *CLB*-фрейму. В інших сімействах мікросхем Virtex 5 збільшено кількість логічних блоків у фреймі до 20 логічних блоків *CLB* у висоту, а у *Virtex* 6 – до 40 *CLB* у висоту [20, 36]. Порівняно з реконфігурацією стовпцями це зменшує розмір бітового потоку, а відповідно і час здійснення реконфігурації та ефективність використання апаратурних ресурсів ПЛІС [78, 79].

Реконфігурація розділами. Існує найбільш гнучкий спосіб розміщення реконфігуровних модулів у динамічній ділянці ПЛІС, коли кожен апаратний модуль реалізується прямокутною групою вузлів, розміри якої не обмежені заздалегідь визначеними розмірами реконфігуровного модуля [15]. На застосуванні реконфігурації розділами (*Partitions*) [35, 70] заснована технологія часткової динамічної реконфігурації в сучасних сімействах мікросхем *Virtex* 6 і *Virtex* 7. Реконфігурація розділами підтримується на рівні сучасних інструментальних засобів розроблення [17, 20, 25, 35, 80, 81], із застосуванням яких можна вирішувати проблеми створення однорідних вузлів, установлення зв’язків між ними, оптимізації площини та співвідношення сторін відповідно до внутрішніх зв’язків модуля [17]. Розглянута технологія забезпечує найефективніше використання апаратурних ресурсів та розподілення зв’язків, а також мінімальні розміри бітової послідовності і, відповідно, зменшення часу реконфігурації [81].

Аналіз тенденцій розвитку технології часткової реконфігурації в різних сімействах мікросхем ПЛІС ілюструє табл. 1.1. Сучасні сімейства ПЛІС компанії *Xilinx* підтримують можливість часткової реконфігурації не тільки на рівні апаратних засобів [35, 70, 79, 80], а і на рівні інструментального

підтримання: часткова динамічна реконфігурація підтримується у програмному пакеті *Xilinx ISE Design Suite* 12 [36, 80].

Таблиця 1.1

Підтримання часткової динамічної реконфігурації в ПЛІС компанії *Xilinx*

Тип реконфігурації	Рік	Сімейство ПЛІС	Підтримання САПР
1	2	3	4
Перше покоління ПЛІС підтримує часткову реконфігурацію (знято з виробництва)	1996	*XC*6200	–
Друге покоління ПЛІС – реконфігурація стовпцями	1999	*Virtex*, *Virtex*II (*Pro*), *Virtex E*	*Xilinx ISE Design Suite* 10.X (2008 р.) 11.X (2009 р.)
Сімейства невисокої ціни без офіційного підтримання часткової реконфігурації. Групами ентузіастів успішно перевірено її спроможність на деяких пристроях	2003	*Spartan* 3, *Spartan* 3*E*, *Spartan* 3*A*	*Xilinx ISE Design Suite* 10.*X* (2008 р.) – інтегрований *Plan Ahead*
Третє покоління ПЛІС підтримує часткову динамічну реконфігурацію фреймами розміром 16 – 20 CLB, а з 2010 р. – розділами	2006	*Virtex* 4	*Xilinx ISE Design Suite* 10.1 (2008 р.) – інтегрований *Plan Ahead*; 11.*X* (2009 р.) – інтегрований *Plan Ahead*; 12.*X* (2010 р.) – автономний *Plan Ahead*
	2006	*Virtex* 5	

Продовження табл. 1.1

1	2	3	4
Сімейство четвертого покоління підтримує часткову динамічну реконфігурацію фреймами розміром 40 CLB, а з 2010 р. – розділами	2009 – 2010	*Virtex* 6, *Spartan* 6	*Xilinx ISE Design Suite* 11.3 (2009 р.) – інтегрований *Plan Ahead*, підтримання порту *ICAP*; 12.*X* (2010 р.) – вбудоване підтримання часткової реконфігурації (розділами); підтримання автономної версії *Plan Ahead* (модульна); 13.*X* (2011 р.).
Сучасні сімейства підтримують часткову динамічну реконфігурацію на базі розділів	2011 – 2013	*Virtex* 7, *Artix* 7, *Kintex* 7, *Zynq* 7000	*Xilinx ISE Design Suite* 13.X (2011 р.) 14.X (2013 р.); підтримання порту *ICAP*

Мікросхеми компанії *Altera* традиційно не підтримують часткову динамічну реконфігурацію [22]. Ситуація змінилась лише з появою останніх модифікацій мікросхем [82, 83].

Практичну реалізацію реконфігуровних комп’ютерних систем на базі часткової динамічної реконфігурації розглянуто у працях [70, 73, 84]. Модуль *ICAP* (*Internal Configuration Access Port*) є основним компонентом, що забезпечує технологічну можливість часткової динамічної реконфігурації [35]. Він є апаратним засобом ПЛІС, який дає змогу динамічно завантажувати в пам’ять потік бітів без застосування порту *JTAG* (*Joint Test Action Group*) [85], якщо конфігураційна інформація зберігається на одному і тому самому кристалі. Якщо часткові конфігурації зберігаються в зовнішній енергонезалежній пам’яті, то для модифікації алгоритму роботи

реконфігуровних модулів необхідно підключати зовнішні засоби завантаження нових бітових послідовностей через порт *JTAG* [68]. Це менш швидкодійна технологія, але з урахуванням обмеження ємності внутрішньої пам'яті ПЛІС і високої ціни їх використання, у працях [20, 70] визначено, що за певних умов зберігання неактивних бітових потоків у зовнішній енергонезалежній пам'яті вона є більш ефективною.

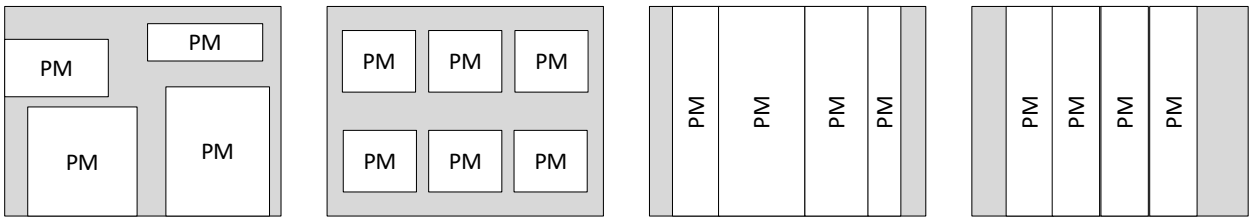
У працях [15, 70] оцінено час виконання часткової реконфігурації на сімействі мікросхем *Virtex* 5 з максимальною швидкодією 3.2 Гбіт/с. Установлено, що мінімальний час реконфігурації для ПЛІС сімейства *Virtex* 5 дорівнює 0,016 мс для найменшого модуля, який займає 0,36% ПЛІС і складається з 160 логічних елементів. Час реконфігурації великого модуля, який займає 10 % логічних елементів ПЛІС, становить 0,42 мс.

У працях [15, 20, 70] установлено, що для ефективної реалізації динамічної реконфігурації обчислювального середовища на ПЛІС доцільно використовувати сучасні пристрої, які підтримують ефективні технології проектування з підтриманням часткової динамічної реконфігурації і забезпечують необхідне інструментальне підтримання для спрощення проектування (табл. 1.1). Це зумовлено негативним впливом, що здійснює політика компаній виробників щодо оновлення своїх продуктів, яка ускладнює оновлення і модифікацію готових обчислювальних систем.

Для планування розкладу виконання обчислень та виділення реконфігуровних обчислювальних ресурсів необхідно враховувати розмір та структуру реконфігуровної ділянки. Розмір цілком залежить від сімейства ПЛІС, що використовується. Структура визначається моделлю розміщення реконфігуровних модулів. У працях [17, 44, 68] описано чотири моделі розміщення задач: 1*D* і 2*D* моделі з фіксованою і гнучкою структурою (рис. 1.3 *а – г*).

Гнучка 2*D* модель (рис. 1.3, *а*) дозволяє виділити прямокутні ділянки для розміщення задач у довільному місці на поверхні реконфігуровної ділянки ПЛІС. Це збільшує щільність розміщення задач і, як наслідок, ефективність

використання динамічної ділянки ПЛІС, але значно ускладнює проектування і реалізацію алгоритмів розміщення. Фіксована 2*D* модель являє собою розділення динамічної області на фіксовану кількість реконфігуровних модулів (рис. 1.3, *б*). Такий спосіб актуальний в сучасних реконфігуровних технологіях завдяки спрощенню планування і розміщення апаратних задач. Фіксоване розміщення реалізовує ефективні способи взаємодії реконфігуровних модулів і статичних елементів системи за рахунок реалізації спеціалізованих комунікаційних середовищ [17]. Моделі 1*D* (рис. 1.3, *в*, *г*) відповідають технології розміщення стовбцями.



а *б* *в* *г*

Рис. 1.3. Моделі розміщення реконфігуровних модулів (РМ): *а* – гнучка 2D; *б* – фіксована 2*D*; *в* – гнучка 1*D*; *г* – фіксована 1*D* – моделі [44, 68];

Проблемі ефективного використання динамічної ділянки ПЛІС присвячено праці [17, 33, 69], у яких запропоновано методи оптимізації розміщення реконфігуровних модулів на ПЛІС. Часто запропоновані методи реалізовані на фізичному рівні кристалів ПЛІС і абстраговані від проблем оптимізації продуктивності на вищих рівнях обчислювальної системи, що дозволяє їх використовувати в комплексі з різноманітними технологіями пришвидшення реконфігурації.

1.3.2. Аналіз особливостей архітектури динамічно реконфігуровних комп’ютерних систем на базі ПЛІС. Оскільки основною проблемою, яка вирішується в дисертаційній роботі, є організація процесу обробки інформації

в РКС, основним критерієм для розгляду архітектур РКС є організація засобів керування обчислювальним процесом.

На підставі запропонованої у праці [20] сучасної класифікації визначено два основні способи організації процесу керування у РКС, які узагальнюють дві базові архітектури реконфігуровних комп'ютерних систем: *системи з апаратними* [46, 88] *та програмними* [44, 86, 87] *засобами керування обчисленнями* (рис. 1.4). У працях [43, 44] наведено аналогічні класифікації РКС, де вони поділяються на апаратні реконфігуровні обчислювальні системи (*RH, Reconfigurable Hardware Systems*) і РКС (*RC, Reconfigurable Computing Systems*).

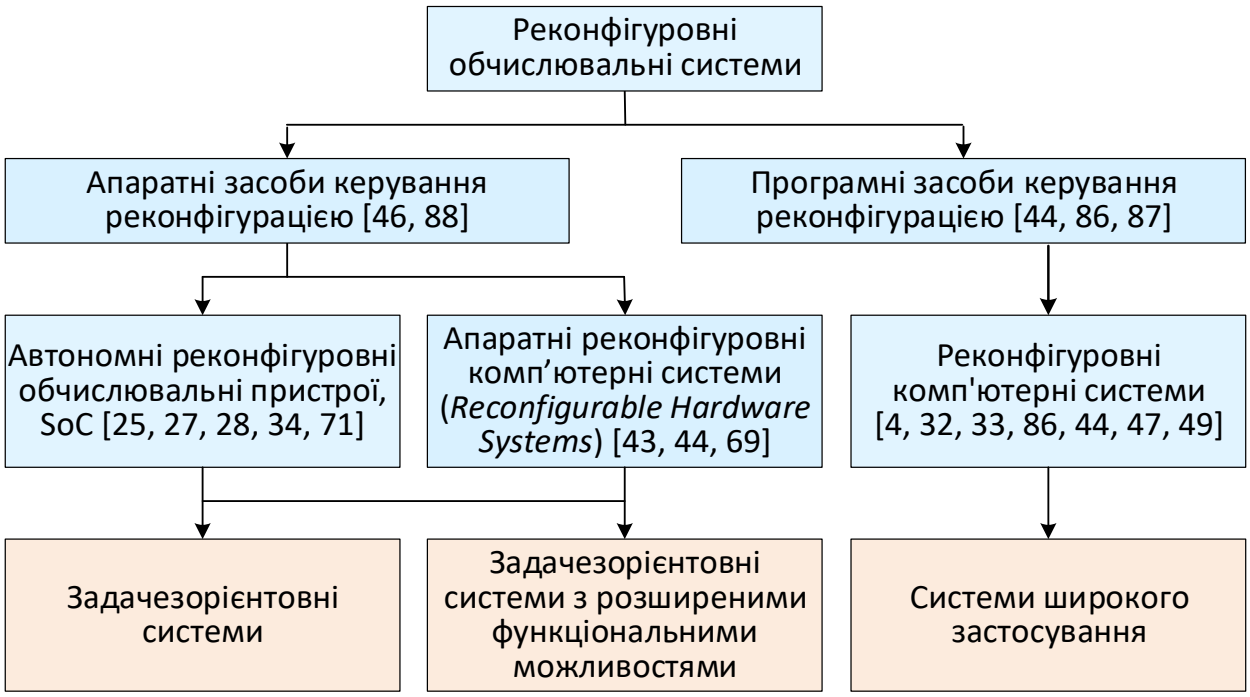


Рис. 1.4. Способи організації засобів керування реконфігурацією

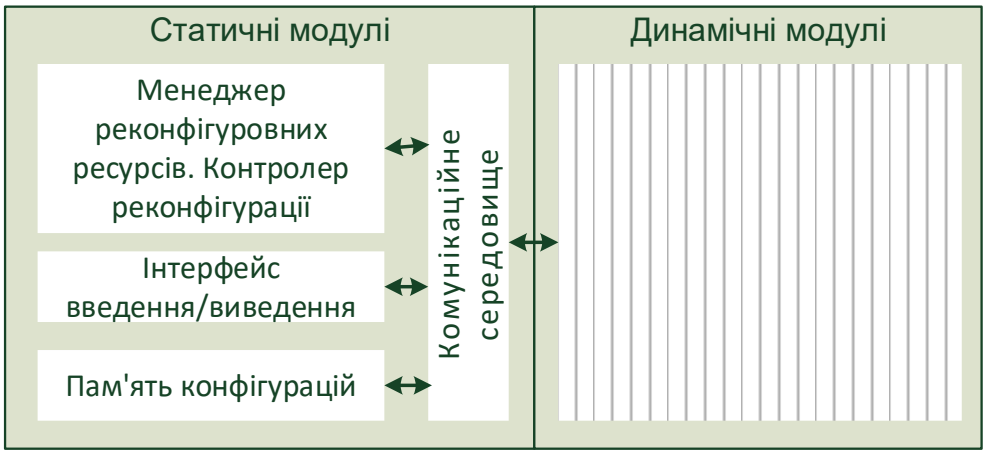
У праці [8] узагальнено методи та засоби керування обчисленнями в РКС, на підставі чого з'ясовано, що завданням керуючих засобів є організація процесів планування розкладу виконання обчислювальних задач, виділення реконфігуровних обчислювальних ресурсів та керування обчисленнями. Основна функціональна відмінність від організації традиційних керуючих засобів полягає в налаштуванні обчислювального середовища, що передує розподіленню на нього потоку обчислювальних задач. Таким чином, процес

керування обробкою інформації в РКС включає реалізацію логічної і фізичної послідовностей реконфігурації обчислювального середовища [14]. Логічна послідовність включає планування і виділення реконфігуровних обчислювальних ресурсів, фізична – розміщення, зупинення, призупинення, відновлення і всі види реорганізації та переміщення завантажених, виконуваних та закінчених задач на поверхні ПЛІС. При цьому з погляду реалізації процесу керування обчисленнями в реконфігуровних обчислювальних системах логічна й фізична послідовність реконфігурації взаємопов’язані.

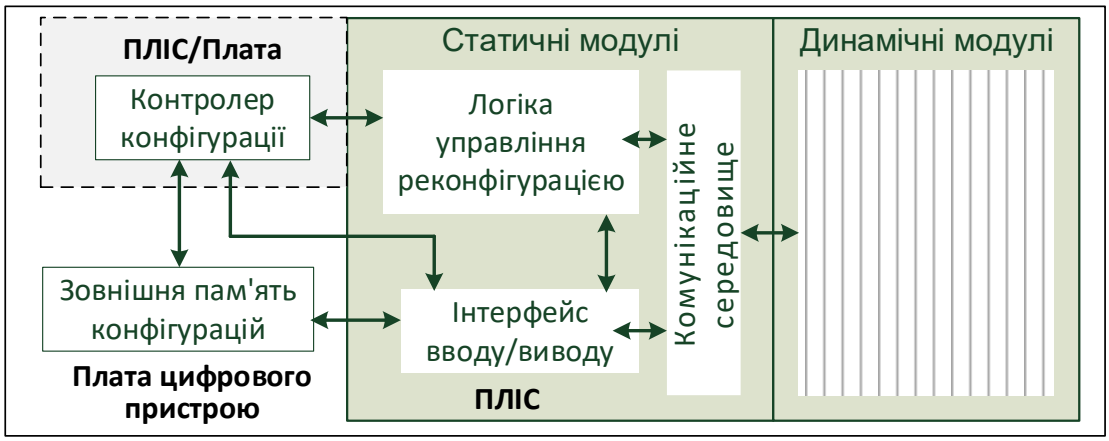
На підставі аналізу основних класів відомих реалізацій реконфігуровних обчислювальних систем, що виконаний вище (див. рис. 1.1 і 1.4), визначено їх типові структурні та функціональні особливості.

Апаратні реконфігуровні обчислювальні системи. Апаратна операційна система розміщується в статичній області ПЛІС і є основною складовою вбудованих автономних реконфігуровних обчислювальних систем та пристроїв (рис. 1.5) [69]. Найбільш характерна функціональна особливість таких систем – автономність від будь-яких зовнішніх впливів та засобів керування. Загальний підхід до побудови апаратних реконфігуровних комп’ютерів зображено на рис. 1.6 [16, 43, 44].

Реконфігуровні комп’ютерні системи. Керування обчисленнями у РКС зазвичай реалізоване на рівні програмної надбудови операційної системи засобами системних процесорів або дискретних процесорних ядер. У сучасних реалізаціях РКС застосовуються стандартизовані операційні системи загального призначення [43, 44] з метою спрощення процесів розроблення, підтримання та експлуатації. Це також дозволяє використовувати традиційні методи планування та розподілу задач на реконфігуровне обчислювальне середовище, а також розширює функціональні можливості РКС.



а



б

Рис. 1.5. Архітектура вбудованих та автономних реконфігуровних систем і пристроїв: а – система-на кристалі; б – система на платі

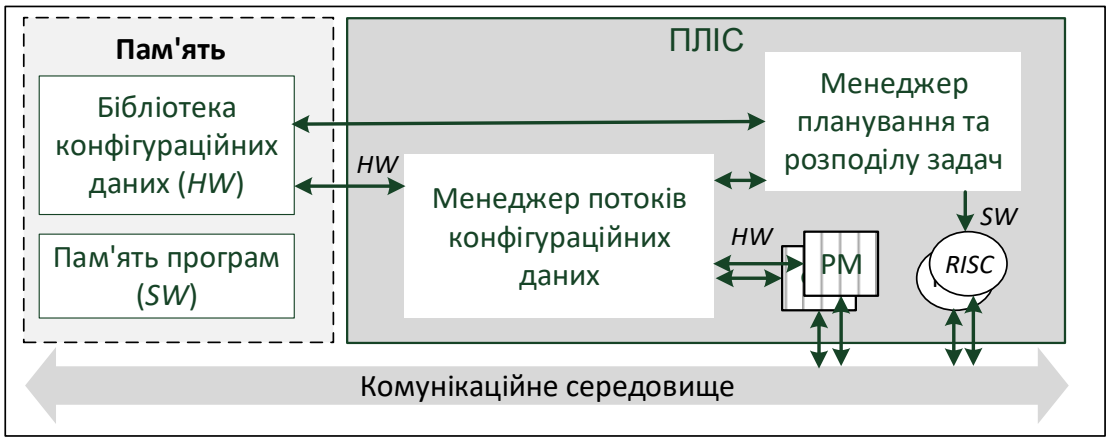


Рис. 1.6. Архітектура апаратних реконфігуровних комп'ютерів: *SW* – програми; *HW* – апаратні задачі; *RISC* – програмовні процесорні ядра

Основні структурні особливості РКС узагальнено на структурній схемі на рис. 1.7, до яких належать:

– відкрита архітектура, модульний принцип побудови, наявність дискретних процесорів, які поєднані високошвидкісними засобами обміну даними;

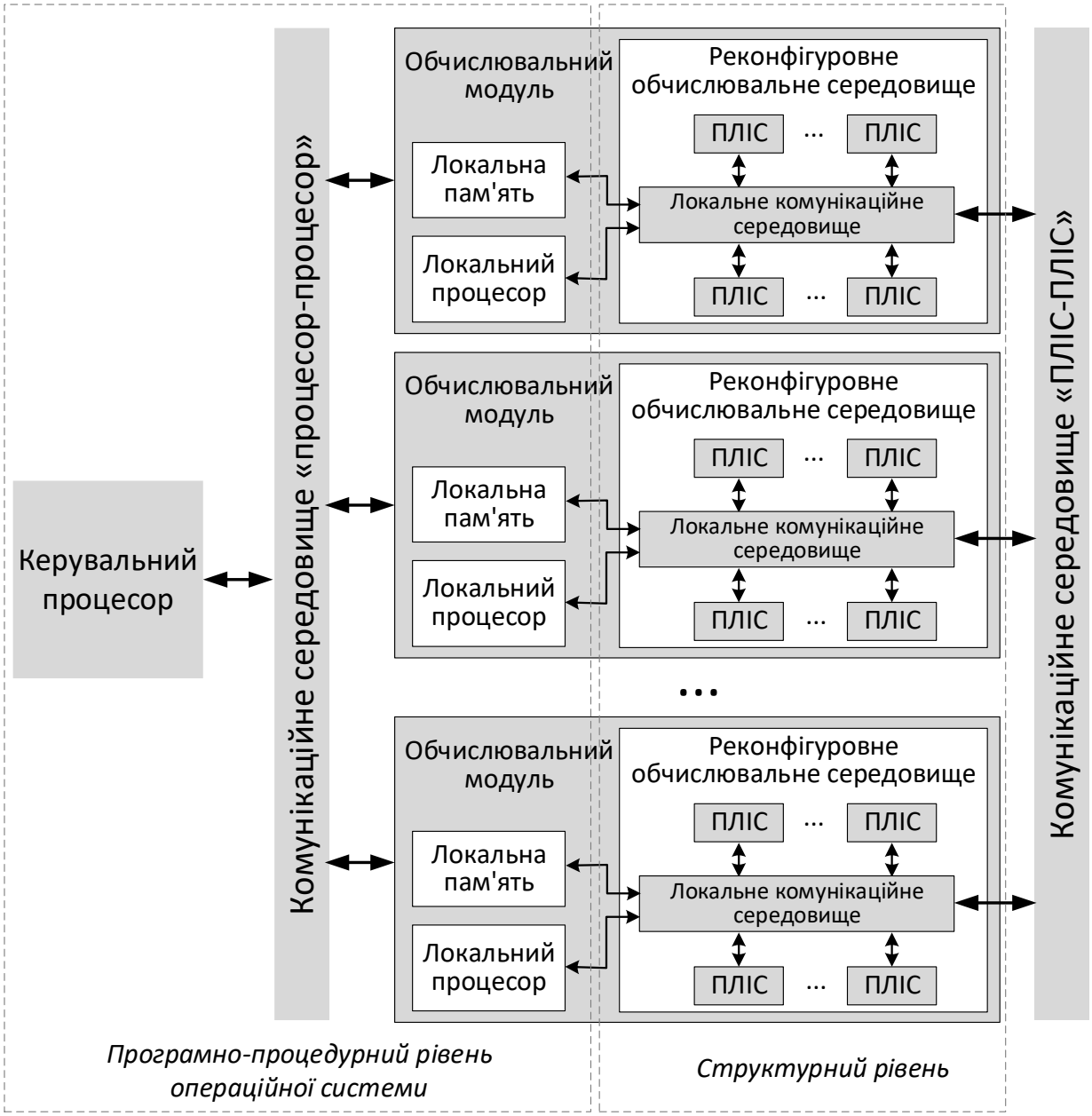


Рис. 1.7. Загальний підхід до організації архітектури реконфігуровних комп’ютерних систем

– у структуру кожного обчислювального модуля (ОМ) входять дискретний процесор, який виконує функції внутрішньомодульного та міжмодульного керування, і розв’язувальне поле (обчислювальне середовище), яке побудоване на базі кристалів ПЛІС;

– обчислювальне середовище має монокристальну або мультикристальну структуру, в тому числі актуальною є гібридна структура обчислювального середовища;

– для підвищення ефективності обміну даними і збільшення розмірів розв’язувального поля у РКС реалізовані мережі передавання даних ПЛІС-ПЛІС з різною топологією та можливостями;

– обчислювальне середовище побудоване на базі ПЛІС, які підтримують технології повної або часткової реконфігурації, сучасні ПЛІС підтримують технологію часткової динамічної реконфігурації;

– конфігураційні дані для налаштування обчислювального середовища зберігаються в централізованих бібліотеках конфігураційних даних (БКД);

– сховища конфігураційних даних у різних реалізаціях по-різному віддалені від поверхні реконфігуровної ділянки ПЛІС.

На підставі наведених вище особливостей структурної організації РКС зробимо такі висновки:

– загальною особливістю всіх класів сучасних РКС є наявність динамічно реконфігуровного обчислювального середовища, яке реалізоване на базі технології часткової динамічної реконфігурації; на ефективність функціонування РКС впливає розмір цієї динамічної області, який обмежує кількість і складність розміщуваних реконфігуровних модулів;

– розташування сховищ конфігураційних даних для налаштування обчислювальних ресурсів у різних класах систем по-різному віддалене від цільової поверхні динамічної області ПЛІС, що є критичною характеристикою щодо виникнення затримань завантаження конфігураційних даних.

Порівняльний аналіз основних класів реконфігуровних комп’ютерних систем наведено в табл. 1.2.

Таблиця 1.2

Порівняльний аналіз основних класів реконфігуровних обчислювальних систем

Класи реконфігуровних обчислювальних систем	Керування обчисленнями	Керування реконфігурацією обчислювального середовища			Зберігання конфігурацій-них даних	Функціональні можливості	
	Операційна система	Реалізація	Розташування	Складність		Функціональність	Клас систем
1	2	3	4	5	6	7	8
Системи-на-кристалі	Апаратна усічена	Апаратна	Вбудовано на кристалі	+	Вбудована пам'ять на кристалі	+	Задачезорієнтовані
Спеціалізовані реконфігуровні обчислювальні пристрої	Апаратна усічена	Апаратна	Локально на кристалі або на платі	+	Локальна пам'ять на платі	±	Задачезорієнтовані

Продовження табл. 1.2

1	2	3	4	5	6	7	8
Апаратні реконфігуровні комп’ютерні системи	Апаратна спеціалізована	Апаратна надбудова	Локально на платі	++	Локальна пам'ять на платі	+++	Задачезорієнтовані з розширеними можливостями
Реконфігуровні комп’ютерні системи	Програмна універсальна	Програмна надбудова	Віддалено або локально	+++++	Віддалена централізована пам'ять	+++++	Загального призначення

Примітка: + – низький ступень, +++++ – високий ступень.

1.3.3. Аналіз методології проектування динамічно реконфігуровних комп’ютерних систем на базі ПЛІС. Методологію проектування та розроблення додатків для РКС описано у працях [19, 22]. Найбільш трудомісткий і специфічний етап проектування – розроблення апаратних ядер для розв’язання обчислювальних задач, за якого прагнення отримати максимальне пришвидшення обчислень ускладнює розроблення. Для програмування застосовуються спеціальні мови апаратного опису апаратури (*VHDL*, *Verilog*) та спеціалізовані інструментальні засоби. Розроблені компоненти завантажуються в бібліотеку готових компонентів. Структурне програмування потребує специфічної кваліфікації програміста.

Незважаючи на ряд перспективних технічних досягнень, реконфігуровні комп’ютерні системи мають проблему через складність структурної компоненти програмування. Істотною ця проблема виявляється під час програмування додатків високого рівня. У праці [57] зазначено, що унікальність кожної реконфігуровної обчислювальної системи ускладнює програмування у зв’язку з концептуальними відмінностями обчислювальних моделей програмного забезпечення та апаратних засобів, відсутністю мови високого рівня для проектування апаратури і широковживаних інструментальних засобів розроблення, єдиного інтерфейсу для зв’язку програміста із ПЛІС.

1.3.4. Аналіз ефективності засобів керування обробкою інформації в динамічно реконфігуровних комп’ютерних системах. У праці [8] розроблено систематизацію відомих РКС за способами реалізації засобів керування обробкою інформації, показано на рис. 1.8. Досліджено співвідношення програмної та апаратної складової для розв’язання задачі реконфігурації у РКС. Визначено місце сучасних реконфігуровних комп’ютерних систем, які ґрунтуються на використанні технології часткової динамічної реконфігурації (рис. 1.8, II). Систематизація також упорядковує відомі реалізації РКС за історичними тенденціями збільшення продуктивності обчислень.

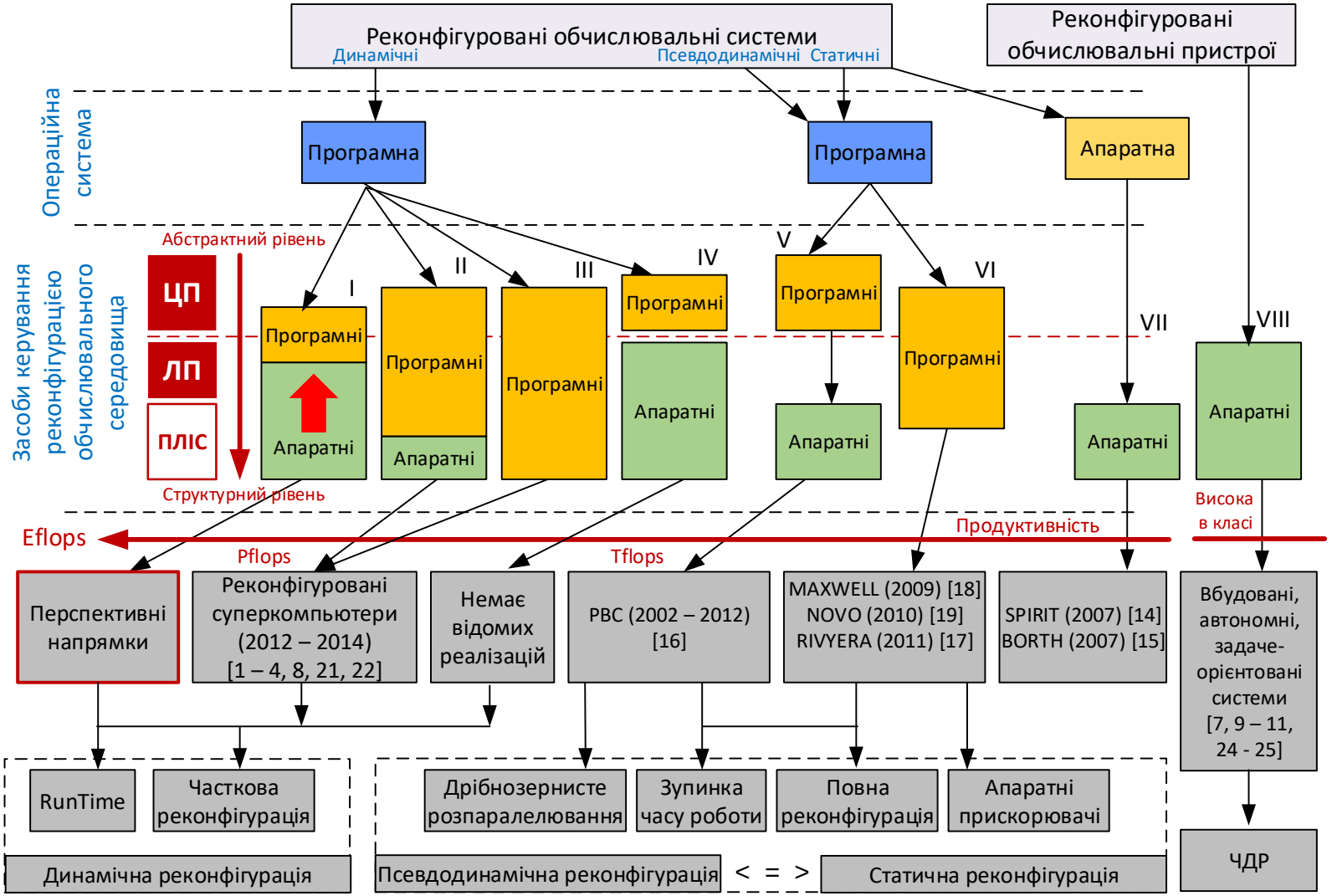


Рис. 1.8. Систематизація засобів керування в реконфігуровних комп’ютерних системах

У роботі визначено, що на ефективність засобів керування реконфігурацією впливає структура обчислювального модуля. У працях [44, 46, 86] розглянуто сучасні гібридні архітектури обчислювальних модулів, які ґрунтуються на наявності як програмного, так і апаратного обчислювального ресурсу, що сприяє збільшенню користувацької ефективності в широкому класі задач (рис. 1.8, II, III). У працях [46, 86] розглянуто гібридні архітектури з дискретним процесором у структурі обчислювального модуля. У працях [33, 44] реалізовані декілька процесорів, між якими розподіляються функції керування та обчислення (рис. 1.8, II).

Описані розв'язки [46, 89], коли фізичний процес керування реконфігурацією покладається на спеціальні засоби апаратних контролерів реконфігурації, що звільняє локальний процесор для розв'язання задач більш високого рівня (рис. 1.8, II). У працях [43, 44] подано класичну автономну апаратну РКС з гібридною архітектурою (рис. 1.8, VIII). Теоретично така архітектура може бути масштабована для розв'язання широкого класу задач (рис. 1.8, IV). У працях [11, 56, 57] відзначено, що розроблення повнофункціональної апаратної операційної системи для реконфігуровних комп'ютерів не є ефективним (рис. 1.8, VII). Відомі реконфігуровні комп'ютерні системи, що описані у працях [19, 22, 64], належать до класу статично РКС (рис. 1.8, V, VI). Ці системи натепер підтримуються.

На підставі систематизації визначено, що співвідношення програмної та апаратної складових реалізації засобів керування обробкою інформації є важливим параметром оцінювання ефективності РКС.

У праці [8] визначено, що у відомих РКС засоби керування реконфігурацією розглядаються на абстрактному та структурному рівнях абстракції обчислювальної системи. Рівні абстракції у РКС характеризуються ступенем абстракції засобів керування обробкою інформації від фізичного процесу реконфігурації обчислювального середовища. Розгляд засобів керування реконфігурацією на двох рівнях абстракцій узагальнено на рис. 1.9. У працях [8, 90] рівні абстракції визначено як багаторівневе подання

архітектури комп’ютера, коли деталі нижнього рівня приховані від верхнього рівня для забезпечення більш простих моделей функціонування верхніх рівнів абстракції архітектури.

Рис. 1.9. Рівні абстракції розгляду засобів керування реконфігурацією

На зовнішньому прошарку абстрактного рівня засобами операційної системи вирішуються загальні проблеми керування РКС, у тому числі: надання доступу користувачам, організація інтерфейсів уведення програм та даних, доступ до загальних ресурсів, зокрема, бібліотеки конфігураційних даних, розроблення й керування паралельними додатками, зокрема, планування та розподіл задач між вузлами зовнішнього прошарку абстрактного рівня. Цей прошарок має найвищий ступінь абстракції від

безпосереднього керування процесом реконфігурації, але його може не бути, а його задачі виконуватиме наступний прошарок абстрактного рівня, що характерно для спеціалізованих убудованих реконфігуровних обчислювальних систем і пристроїв.

Внутрішній прошарок абстрактного рівня керується операційною системою і являє собою деяку розподілену надбудову операційної системи, яка більшою чи меншою мірою враховує вплив фізичного процесу реконфігурації.

Планувальники та розподільники задач виконують класичне завдання створення розкладів та розподілу задач між обчислювальними вузлами на абстрактному рівні. Але на абстрактному рівні РКС можна лише до деякої міри розглядати збільшені структурні об’єкти. Реконфігуровні обчислювальні вузли не мають фіксованої архітектури і явно виражених обчислювальних параметрів, які налаштовуються сумісно з процесом планування та розподілу задач. При цьому враховується ряд обмежень та умов, які накладає фізична та просторова структури реконфігуровної ділянки.

Таким чином, на абстрактному рівні операційної системи необхідно враховувати технологічні особливості елементної бази, способи розміщення апаратних задач, апаратні та просторові обмеження реконфігуровної структури, фізичні параметри апаратних задач, часові та енергоспоживні параметри фізичного процесу реконфігурації.

Усі рівні абстракції у РКС тісно взаємозв’язані у функціональному процесі, що зумовлює необхідність постійної взаємодії найвищого шару реалізації програмного забезпечення з найнижчим шаром фізичної організації обчислювального середовища, оскільки специфіка організації реконфігуровних обчислень потребує врахування фізичних параметрів кристалів ПЛІС на всіх етапах процесу обробки даних. Це є одним з важливих функціональних обмежень реконфігуровних обчислювальних середовищ, що впливає на їх ефективність і не дозволяє ефективно використовувати традиційні технології паралельного програмування.

У праці [8] наведено оцінку ефективності засобів керування обробкою інформації в РКС на різних рівнях абстракції. Для оцінювання вибрано такі параметри відомих РКС:

1. забезпечення апаратного пришвидшення обчислень;
2. складність реалізації структурної складової керування;
3. перехід на вищий рівень для реалізації структурної складової керування, що потребує додаткових витрат продуктивності;
4. ступінь гнучкості керувальних засобів, що зумовлює складність зміни алгоритмів керування відповідно вдосконаленням структури реконфігуровного обчислювального середовища;
5. обчислювальна складність реалізації алгоритмів керування реконфігурацією;
6. використання ресурсу операційної системи;
7. складність, трудомісткість розроблення та експлуатації;
8. специфічність розроблення.

Розроблено п’ятибальну абстрактну шкалу оцінювання ефективності реалізації структурної складової: 0 – абсолютно ефективно (100%), з практичного погляду недосяжно; 1 – ефективно, досягається незначними зусиллями або надає значний приріст швидкодії (90% – 70%); 2 – допустимо (60% – 40%); 3 – нижче від допустимого, досягається значними зусиллями або витратами часу (30% – 20%); 4 –неможливо або абсолютно неефективно (0%).

Оцінювання І. Виконана на підставі аналізу відомих реалізацій реконфігуровних комп’ютерних систем з програмними та апаратними засобами керування обчислювальним процесом, особливості якого ілюструє рис. 1.10.

Підхід апаратного пришвидшення засобів керування при суміщенні абстрактного та структурного рівнів тягне за собою значні ускладнення реалізації, підтримання, експлуатації, зумовлені, зокрема, необхідністю апаратної реалізації певних функцій операційної системи. Програмна реалізація характеризується значною обчислювальною складністю реалізації

керувальних алгоритмів, непродуктивно витрачає продуктивність системи, базується на застосуванні традиційних технологій планування та розподілу задач, що розроблені для фіксованих обчислювальних ресурсів. Із діаграми видно, що для реалізації процесу керування обробкою інформації в динамічно РКС розглянута реалізація неефективна.

З огляду на вагомі переваги апаратного пришвидшення й безпосередньої близькості апаратних засобів до фізичного рівня системи розглянуто рівні абстракції керувальних засобів, які характерні для класу автономних убудованих та спеціалізованих реконфігуровних обчислювальних систем.

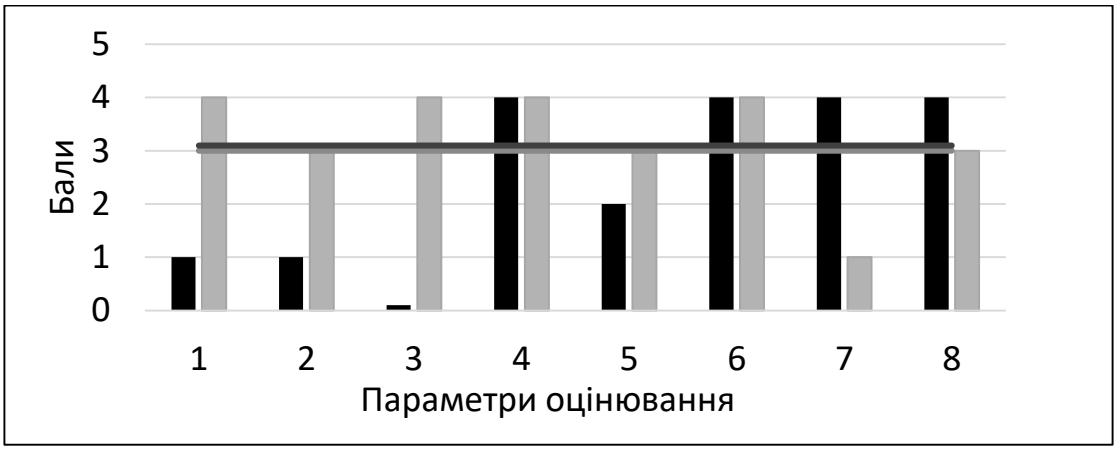


Рис. 1.10. Оцінка ефективності поширеної реалізації керувальних засобів: ■ – апаратна реалізація; ■ – програмна реалізація; ━, ━ – усереднені показники відповідно

Оцінювання II. В означених РКС зазвичай реалізований принцип відокремленої реалізації рівнів керування реконфігурацією [91 – 94]. Тоді абстрактний рівень виконується суто засобами операційної системи і не розглядається в контексті реалізації процесу керування реконфігурацією. Для цього використовуються спеціалізовані апаратні або програмі засоби керування процесом реконфігурації на структурному рівні обчислювального модуля РКС. Як видно з рис. 1.11, локалізація програмних засобів керування реконфігурацією неафективна, у той час, як локалізація апаратних засобів керування на структурному рівні сприяє збільшенню ефективності.

Визначено, що апаратна реалізація забезпечує додатковий приріст продуктивності, а розміщення апаратних засобів керування на одному рівні з об’єктами керування реконфігурацією максимально зменшує витрати продуктивності на їх реалізацію, тобто забезпечення структурної складової керування досягається мінімальними обчислювальними витратами.

Апаратні засоби керування функціонують на структурному рівні обчислювального середовища, що дає змогу з більшою ефективністю враховувати фізичні параметри ПЛІС. Утім всі відомі реалізації апаратних РКС характеризуються специфічністю, складністю підтримання та експлуатації, обмеженими функціональними можливостями, що не дозволяє розглядати їх як самостійну перспективну технологію для реалізації РКС.

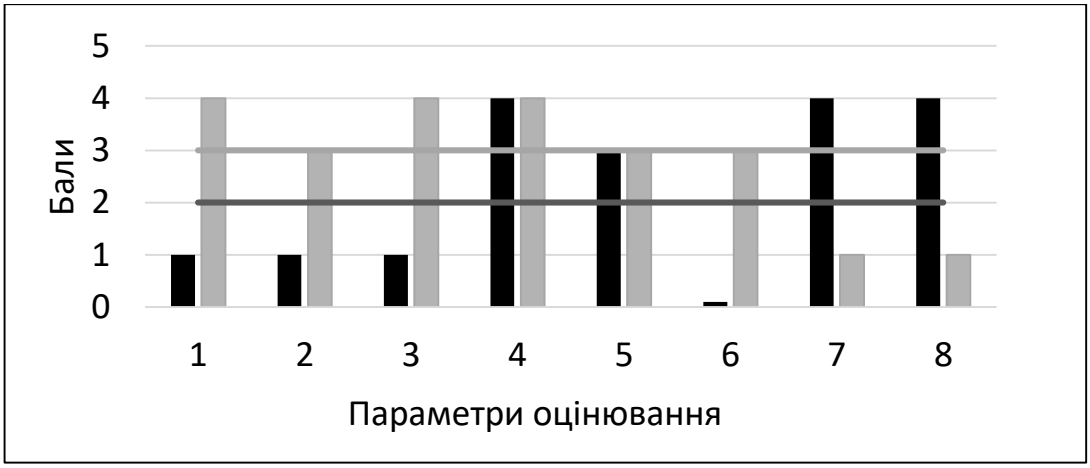


Рис. 1.11. Оцінка ефективності розділеної реалізації абстрактного та структурного рівнів керувальних засобів: ■ – апаратна реалізація; ■ – програмна реалізація; ━, ━ – усереднені показники відповідно

Таким чином, визначено, що для вирішення проблеми динамічного планування паралельних обчислень і розподілу задач на динамічно реконфігуровне обчислювальне середовище існуюча організація засобів керування неефективна, і отже, потребує перегляду існуючих рівнів абстракції.

1.3.5. Аналіз ефективності методів та засобів відображення задач на реконфігуровне обчислювальне середовище. Процес відображення задач у РКС включає процес планування розкладу виконання завдань і виділення реконфігуровних обчислювальних ресурсів для виконання завдань. Для визначених у праці [20] класів РКС актуальні: *Off-Line* режим (статичний) під час зупинення роботи системи або під час компіляції і *On-Line* режим (динамічний) під час роботи РКС.

Off-Line режим відображення характерний для статично РКС, коли вихідний граф задачі відомий заздалегідь і є достатньо часу для знаходження оптимального розв'язку. У працях [44, 46 95] зазначено, що застосування традиційних статичних методів планування задач, таких як цілочислове лінійне програмування *ILP* (*Integer Linear Programming*), еволюційні алгоритми тощо, не є ефективними в динамічно РКС у зв'язку зі значними часовими витратами.

У працях [15, 96] описано використання *Off-Line* відображення в режимі реального часу, коли елементна база не підтримує механізми часткової динамічної реконфігурації, наприклад, під час використання пристроїв компанії *Altera* до 2010 р. випуску. *Off-Line* режими використовуються у статично РКС [19, 22, 64].

У праці [97] досліджено статичний спосіб розв'язання пакувальної задачі за допомогою методу гілок та границь, за яким визначається схема найменшого розміру для розміщення всіх задач, максимально допустимого часу виконання за заздалегідь відомим розкладом, схеми розміщення для заданих розмірів мікросхеми і заданого розкладу.

У праці [95] показано, що використанням статичних методів можна ефективно зменшити непродуктивні витрати часу реконфігурації, якщо реконфігурацію виконувати завчасно. Пришвидшення реконфігурації досягається за рахунок суміщення в часі процесів реконфігурації і обчислення, при цьому розклад виконання складається на етапі планування.

On-Line сценарії працюють під час роботи системи в режимі *RunTime* в процесі надходження нових задач. Для планування розкладу виконання можуть бути задіяні традиційні алгоритми динамічного планування [42], але це неефективно, оскільки планувальник і розподільник сильно залежать один від одного. Алгоритми *On-Line* сценаріїв вимагають поєднання етапів планування і розподілу задач на реконфігуровне обчислювальне середовище з урахуванням параметрів задач, фізичних властивостей та обмежень реконфігуровного обчислювального середовища, часових витрат на здійснення реконфігурації.

Під час застосування *On-Line* відображення часто виникають проблеми відхилення задач у випадку неможливості надання їм необхідного часу для виконання або розміщення [44. 98]. На час обчислень впливає необхідність дефрагментації динамічної ділянки ПЛІС та вивантаження задач з різних причин: зменшення статичного енергоспоживання, кешування конфігурацій, призупинення некритичних задач тощо [99]. Планування та розподіл задач у гібридних реконфігуровних системах ускладнюється необхідністю врахування різної природи програмних і апаратних задач. Є задачі, що більш ефективно реалізовуватимуться апаратно і навпаки – не можуть бути розміщені на апаратурі [100].

1.3.6. Аналіз методів та засобів зменшення накладних витрат процесу реконфігурації обчислювального середовища на ПЛІС. Відомі засоби підвищення ефективності функціонування РКС в основному спрямовані на розроблення алгоритмів планування обчислень і розміщення задач на реконфігуровне обчислювальне середовище, що водночас реалізують різні методи зменшення накладних витрат реконфігурації. Найпоширенішими методами є: попереднє завантаження конфігурацій [47, 91, 95], повторне використання обчислювальних ресурсів [44, 46, 101], кешування конфігурацій і пошук ізоморфних структур з метою зменшення комунікаційних витрат і руху даних [33, 47]. Зазвичай методи планування обчислень і розподілу задач

у РКС засновані на підтриманні списку вільного або зайнятого простору на поверхні ПЛІС [44]. Завчасна реконфігурація є окремим випадком повторного використання реконфігуровних ресурсів. Підтримання на ПЛІС заздалегідь сконфігурованих функціональних блоків потребує постійних статичних енерговитрат; ці проблеми досліджуються у праці [102]. Автори пропонують зменшення «струму витоку» на транзисторах шляхом використання часткової реконфігурації для тимчасового вивантажування блоків прискорювачів. Але зважаючи на те, що процес прошивання кристала ПЛІС призводить до значних непродуктивних витрат як часу, так і енергії, запропоновано підхід ефективний за умови збереження вивантажених даних у внутрішній пам’яті ПЛІС.

Для знаходження компромісу між зменшенням статичних енерговитрат на підтримання попередньої реконфігурації і зменшенням накладних витрат у праці [95] розглядають комплексне застосування двох підходів: попередню реконфігурацію «якомога раніше» і реконфігурацію «якомога пізніше».

Оптимізація просторового розміщення задач розглядається у працях [44, 98], у яких дослідження спрямовані на зменшення коефіцієнта відмов розміщення задач та зменшення фрагментації реконфігуровної області. У праці [54] розглянуто метод планування та розподілу задач на реконфігуровну структуру із забезпеченням якості обслуговування, що вимагає виконуваний додаток, та врахуванням просторових обмежень ПЛІС. У працях [100, 103] зменшення коефіцієнта відмов розміщення задач забезпечують динамічні алгоритми сумісного планування та розміщення з витісненням задач у гетерогенних системах: апаратну задачу може бути витіснено і перезапущено як програмну і навпаки.

1.3.7. Аналіз технологій реконфігурації обчислювального середовища із врахуванням апаратурних обмежень ПЛІС. Завантаження конфігураційних даних затримується через порівняно повільні інтерфейси зовнішньої конфігурації ПЛІС [68]. Можливість конфігурування мікросхеми зсередини

засобами *ICAP* [85] сприяє значному зменшенню часу реконфігурації, якщо бітові потоки зберігаються у внутрішній пам’яті мікросхеми [102]. Таким чином, засоби попередньої вибірки конфігурацій, повторного використання обчислювальних ресурсів, динамічної самореконфігурації, а також дефрагментації реконфігуровного обчислювального середовища і витіснення задач максимально збільшують швидкість завантаження конфігурацій, зберігаючи їх у внутрішній пам’яті ПЛІС.

Кількісну оцінку використання внутрішньої пам’яті для підтримання часткової динамічної реконфігурації виконано в праці [70]. Виявлено, що обмежені ресурси внутрішньої пам’яті ПЛІС є основною перешкодою, яка не дозволяє зберігати велику кількість конфігураційних даних. Ємність внутрішньої блокової пам’яті поширених реконфігуровних пристроїв сімейства Virtex 4 становить 108 – 756 кбайт, а сімейства Virtex E – 4 – 16 кбайт. Розмір повного конфігураційного потоку для мікросхем сімейства Virtex E досягає 68 – 748 кбайт. За даними праць [15, 68] ємність конфігураційних файлів реконфігуровних лічильників, що відрізняються довжиною лічильного регістра, досягає 24 – 131 кбайт. При цьому найменший модуль займає 0,36% логічного ресурсу мікросхеми Virtex-5 FX70, найбільший – 10%.

Таким чином, для ефективної реалізації технології часткової динамічної реконфігурації необхідно знаходити компроміс між обмеженими ресурсами внутрішньої пам’яті мікросхеми ПЛІС і накладними витратами часу реконфігурації обчислювального середовища. Для вирішення цієї проблеми потрібно залучати зовнішню пам’ять [33, 46].

Ряд праць, які пропонують архітектурні та технічні вдосконалення базових технологій часткової реконфігурації, ґрунтуються на апаратних методах кешування та віртуальних моделях пам'яті, засобах збільшення пропускної здатності інтерфейсів реконфігурації. Так, у праці [46] поряд з існуванням двох процесорів для керування реконфігурацією і виконання програмних функцій передбачено окремий апаратний контролер

реконфігурації, який звільняє центральний керувальний процесор від витрат на реконфігурацію і збільшує пропускну здатність реконфігурації, сприяючи пришвидшенню обчислень. У працях [92, 93] для підвищення швидкості завантаження конфігураційних потоків запропоновано підхід прямого доступу до пам'яті конфігурацій. Розміщення блока кеш-пам'яті поряд з портом *ICAP* сприяє подальшому збільшенню продуктивності [93]. Однак розміщення пам'яті великої ємності на ПЛІС виявляється неефективним.

Беручи до уваги, що максимальна пропускна здатність *SRAM* дорівнює 400 Мбайт/с, у праці [89] запропоновано спеціальні засоби синхронізації роботи порту *ICAP* і зовнішньої пам'яті, які спрямовані на пришвидшення передавання конфігураційних даних. У праці [93] за режиму прямого доступу до пам'яті досягнута пропускна здатність 82 Мбайт/с, а в разі розміщення блока кеш-пам'яті на базі внутрішньої пам'яті *RAM* (*BRAM*) мікросхеми поряд з інтерфейсом *ICAP* відбувається подальше пришвидшення до 378 Мбайт/с. Але розміщення кеш-пам'яті великої ємності на ПЛІС обмежує ефективність запропонованих засобів. У праці [92] між *ICAP* і зовнішньою *SRAM* досягнута пропускна здатність 295 Мбайт/с у режимі прямого доступу. У праці [102] реконфігурація пришвидшується за рахунок реалізації пристрою реконфігурації для зв'язку *ICAP* та зовнішньої *SRAM* у режимі прямого доступу, що дозволяє забезпечити швидкість передавання конфігураційного потоку близьку до ідеальної – 400 Мбайт/с. Для нових поколінь ПЛІС автори очікують досягнення пропускної здатності до декількох Гбайт/с [92].

На підставі викладеного зробимо висновок, що підтримання засобів зменшення накладних витрат шляхом вбудовування в традиційні алгоритми планування та розміщення задач, які зазвичай покладаються на рівень програмної або програмно-апаратної надбудови операційної системи, характеризується значними витратами продуктивності.

Переважна кількість відомих засобів удосконалення структурного рівня відокремлені від процесу планування та розміщення задач на реконфігуровне ОС, що неефективно для динамічно реконфігуровних комп'ютерних систем,

де всі засоби функціональні процеси сильно залежать один від одного і у більшості випадків становлять єдиний механізм.

1.3.8. Аналіз методів та засобів підвищення ефективності паралельної обробки інформації в динамічно реконфігуровних комп'ютерних системах. Як ефективні рішення в галузі паралельних обчислень зазвичай розглядають варіювання зернистістю обчислень [104 – 106]. Можливість динамічної реконфігурації обчислювального середовища створює передумови для підвищення користувацької ефективності паралельних обчислювальних систем через варіювання зернистістю обчислювального середовища відповідно до вимог обчислювальних задач. Однак відомі методи та засоби варіювання зернистістю обчислень розроблені для фіксованих обчислювальних структур [105, 106] і не можуть бути ефективно використані в динамічно РКС.

У відомих реконфігуровних комп'ютерних системах проблеми варіювання зернистістю вирішуються шляхом виділення необхідного обсягу обчислювальних ресурсів із загального реконфігуровного обчислювального середовища. У праці [54] реалізовано реконфігуровну систему команд для пришвидшення функціональних ядер (*Instruction Set Extensions ISEs*) [107, 108] через використання великоозернистих реконфігуровних архітектур (*Coarse-grained reconfigurable architectures, CGRAs*) [109]. Виконання відображення задач ґрунтуються на визначенні мінімальної кількості апаратури для забезпечення необхідного часу виконання алгоритму через реконфігурацію каналів зв'язків між зумовленими обчислювальними структурами. Варіювання зернистістю обмежується зведенням обчислювального алгоритму до визначеної форми, зумовленої структурою системи.

Незважаючи на те, що надвисокий ступінь інтеграції сучасних ПЛІС дозволяє абстрагуватись від апаратурних обмежень кристалів ПЛІС, просторові, технологічні та фізичні характеристики кристалів ПЛІС мають певний вплив на розмір «зерна» обчислювального середовища під час розв'язання задач великої і надвеликої розмірності. На розмір «зерна»

обчислень, окрім традиційних параметрів складності паралельної реалізації алгоритмів і просторових параметрів ПЛІС, значно впливають затримання внутрішніх каналів передавання даних [16].

1.3.9. Узагальнення проблем функціональної та структурної організації динамічно реконфігуровних комп'ютерних систем, що впливають на ефективність обробки інформації. Використання ПЛІС, що динамічно реконфігуруються, створює передумови і нові можливості для підвищення ефективності паралельних обчислювальних систем за рахунок значного зменшення накладних витрат часу на перепрограмування кристалів ПЛІС, а також можливості їх реконфігурації в динамічному режимі.

На підставі виконаного вище огляду відомих реалізацій РКС та особливостей їх структурної і функціональної організації визначено, що ці системи мають певні функціональні та апаратні обмеження, що породжують проблему підвищення ефективності обробки інформації в РКС.

Функціональні та апаратні обмеження РКС полягають у такому:

1. Особливістю архітектури динамічно реконфігуровних комп'ютерних систем є наявність реконфігуровного обчислювального середовища, структура якого може перебудовуватись у режимі роботи системи. Реконфігуровне обчислювальне середовище з огляду на абстрактний рівень операційної системи являє собою віртуальну масштабовану обчислювальну структуру, яка з урахуванням структурного рівня є взаємозв'язаним обчислювальним середовищем з певної кількості кристалів ПЛІС. На ефективність процесу відображення задач на реконфігуровне обчислювальне середовище впливають апаратурні обмеження кристалів ПЛІС, які обмежують складність створюваних обчислювальних структур та кількість розміщуваних ФБ.

2. Розміщення централізованих бібліотек конфігураційних даних для перепрограмування обчислювальних ресурсів зазвичай віддалено від цільової поверхні ПЛІС, що спричинює значні затримання під час завантаження конфігураційних даних.

3. Забезпечення процесу реконфігурації характеризується значними накладними витратами. Накладні витрати вимірюються непродуктивними витратами часу, обчислювальної потужності та енергоспоживання.

4. Відомі методи та засоби відображення задач на реконфігуровне обчислювальне середовище дозволяють відображувати задачі на доступні апаратні ресурси реконфігуровного обчислювального середовища за критерієм мінімізації непродуктивних витрат часу. При цьому проблеми обмежень обчислювального простору вирішуються «на льоту» традиційними способами дефрагментації простору ПЛІС, призупинення та вивантаження некритичних задач, відхилення виконання. Це заважає ефективно відображувати задачі на нестале обчислювальне середовище через додаткові неконтрольовані витрати часу, що можуть зневілювати весь ефект від апаратного пришвидшення.

5. Для вирішення проблеми мінімізації непродуктивних витрат, зменшення часу апаратних обчислень і забезпечення ефективного використання вирішального простору ПЛІС в умовах часових обмежень важливою стає проблема оптимізації процесу обробки інформації, що зводиться до визначення оптимального співвідношення вимог розв’язуваних задач і структури реконфігуровного обчислювального середовища.

6. Усі відомі методи та засоби зменшення накладних витрат убудовані в алгоритми планування обчислень та розподілу ресурсів і зазвичай реалізовані на рівні надбудови операційної системи. Це знижує ефективність обробки даних і завантажує операційну систему розв’язанням неспецифічних задач, що спричиняється необхідністю врахування параметрів фізичного рівня на абстрактному рівні операційної системи.

7. Сучасні технології ПЛІС пропонують механізми зменшення накладних витрат з використанням часткової динамічної реконфігурації. Часткова реконфігурація природно і ефективно сприяє зменшенню затримань реконфігурації, але накладає специфічні вимоги на реалізацію засобів керування паралельними додатками та реконфігуровними обчислювальними

ресурсами. Специфіка організації процесу реконфігурації потребує врахування фізичних параметрів ПЛІС на всіх етапах від процесу планування виконання паралельних додатків до процесу розподілу задач на новостворену обчислювальну структуру. Засоби керування реконфігурацією функціонують на всіх рівнях абстракції обчислювальної системи. При цьому способи їх реалізації є критичною характеристикою. Для апаратних систем характерна реалізація засобів керування на структурному рівні, що дозволяє найбільш ефективно враховувати фізичні параметри реконфігуровного обчислювального середовища на ПЛІС і забезпечувати ефективні алгоритми логічного та фізичного процесів реконфігурації. У класі реконфігуровних комп'ютерів централізовані засоби керування реконфігурацією є програмною надбудовою операційної системи, що зумовлює високу обчислювальну та логічну складність урахування фізичних параметрів розподіленого обчислювального середовища на ПЛІС і збільшує накладні витрати.

8. Рівні абстракції, на яких розглядаються засоби керування обчисленнями у відомих РКС, визначені як неефективні. Існуючі рівні абстракції зумовлюють надмірну складність реалізації процесу керування обчисленнями і дедалі збільшують накладні видатки реконфігурації, що негативно впливає на загальну ефективність обробки даних в РКС.

9. У статично реконфігуровних обчислювальних системах технології паралельного програмування реалізуються на етапі розроблення програм. Це не дозволяє виявляти та реалізовувати неявний паралелізм, а також ефективно організовувати багатозадачний режим обчислень, який характерний для систем керування, коли стратегія керування змінюється залежно від зовнішніх факторів і має обслуговуватися відповідною програмою. Статичним аналізом графів алгоритмів обчислювальних задач виявити паралельні процеси для різних поєднань програм з урахуванням взаємного зсуву в часі їх запуску зазвичай не можливо [51]. Це обмежує застосування статично реконфігуровних комп'ютерних систем, зокрема для розв'язання задач в обмеженому режимі часу.

10. Відомі засоби динамічного планування для виконання дрібнозернистих обчислень на слабкозв’язаних обчислювальних системах, які хоча і дають змогу виявляти паралельні гілки програм, що виникають безпосередньо у процесі обчислень, однак їх застосування для розв’язання задач динамічного відображення завдань на реконфігуровне обчислювальне середовище на ПЛІС реалізовується на рівні операційної системи. У РКС це призводить до додаткових витрат часу, що критично для функціонування таких систем.

ПОСТАНОВКА ЗАВДАННЯ ДИСЕРТАЦІЙНОЇ РОБОТИ

Виконаний аналіз дозволив оцінити і підсумувати проблеми, що впливають на ефективність функціонування реконфігуровних комп’ютерних систем:

– значні непродуктивні часові витрати під час реконфігурації обчислювальної структури, що зумовлені затриманнями передавання великих обсягів конфігураційних даних із віддалених централізованих бібліотек;

– апаратні обмеження ПЛІС, зокрема просторові, та обмежені апаратні ресурси внутрішньої пам’яті ПЛІС і низька швидкодія інтерфейсів конфігурації;

– специфічні вимоги до засобів керування паралельним обчислювальним процесом;

– складність модифікації бібліотечних реалізацій функціональних ядер у динамічному режимі.

Визначені проблеми не ефективно вирішувати засобами традиційних технологій паралельних обчислень та методів і засобів підвищення їх ефективності. Традиційні технології паралельних обчислень орієнтовані на фіксоване обчислювальне середовище без будь-яких функціональних обмежень або на статичні технології реконфігурації, коли розглянуті обмеження не є критичними.

У зв'язку з цим виникає необхідність вирішення проблеми ефективної організації процесу обробки інформації в системах, що мають певні обмеження за своїми функціональними і апаратними можливостями, які зумовлені використанням динамічно програмовної елементної бази.

Таким чином, у розділі поставлено завдання, вирішення яких сприятиме вирішенню важливої науково-технічної проблеми – організації адаптивних паралельних обчислень у межах систем, що мають певні обмеження функціональних і апаратних можливостей і зумовлені використанням динамічно перепрограмовної елементної бази. Основні з них такі:

1. Переглянути існуючі рівні абстракції розгляду засобів керування обробкою інформації в РКС для:

– забезпечення можливості використання на кожному рівні найефективніших технологій паралельного програмування;

– децентралізації і локалізації процесу керування реконфігурацією обчислювального середовища якомога ближче до фізичного рівня його реалізації;

– забезпечення ефективного врахування фізичних параметрів ПЛІС;

– зменшення руху даних на рівні операційної системи, спрощення масштабування і мобільності технологій.

2. Забезпечити ефективне керування паралельним обчислювальним процесом та реконфігуровними обчислювальними ресурсами на структурному рівні через:

– зменшення накладних видатків реконфігурації;

– оптимізацію процесу обробки інформації за критеріями співвідношення непродуктивних витрат часу та функціональних і апаратних обмежень РКС;

– зменшення витрат продуктивності на реалізацію алгоритмів керування процесом реконфігурації обчислювального середовища;

– підвищення ефективності механізмів розпаралелювання та розподілу динамічного потоку задач на реконфігуровне обчислювальне середовище.

3. Збільшити користувацьку ефективність обчислень у широких класах задач керування через:

– оптимізацію часу виконання обчислень на рівні апаратури ПЛІС за співвідношенням параметрів обчислювальних задач і продуктивності реконфігуровного обчислювального середовища з урахуванням просторових обмежень кристалів ПЛІС;

– реалізацію бібліотеки апаратних функціональних ядер з можливістю варіювання зернистістю обчислень.

РОЗДІЛ 2
МАТЕМАТИЧНІ МОДЕЛІ АДАПТИВНОГО ВІДОБРАЖЕННЯ ЗАДАЧ НА РЕКОНФІГУРОВНЕ ОБЧИСЛЮВАЛЬНЕ СЕРЕДОВИЩЕ

2.1. Визначення та обґрунтування класів задач для ефективного розв’язання в динамічно реконфігуровних комп’ютерних системах

Сучасні системи оброблення та передавання даних характеризуються властивістю багатовимірності. У таких системах виникає потреба у розв’язанні специфічних обчислювальних задач, що характеризуються

- надвеликою розмірністю,
- умовами невизначеності,
- режимом реального часу.

Інформаційний [18, 110] і багатовимірний [111, 112] характер сучасних задач робить їх критичними до часу виконання і специфіки обчислювального середовища сучасних високопродуктивних обчислювальних систем. На підставі виконаного в дисертаційній роботі аналізу структурної та функціональної організації динамічно РКС визначено передумови їх застосування для підвищення ефективності розв’язання сучасних задач інформаційного типу [6, 18]. У працях [13, 16 – 20, 22, 25, 27, 34, 51, 66, 67, 70, 71, 111 – 119] визначено й обґрунтовано такі класи задач для ефективного розв’язання засобами динамічно реконфігуровних комп’ютерних систем:

- задачі керування в режимі реального часу;
- задачі керування в невизначеному базисі;
- оброблення великих обсягів інформації;
- багатовимірні обчислення;
- алгебра багатовимірних матриць;
- лінійна алгебра;
- тензорна алгебра.

2.1.1. Особливості розв’язання задач керування в умовах невизначеності. Проблеми умов невизначеності висвітлено у працях [111, 120 – 125]. Зазначено, що в переважній більшості задач керування, пов'язаних із прийняттям рішень, має бути врахована складна інформаційна ситуація, особливістю якої є висока апріорна невизначеність про властивості об'єкта і вплив навколишнього середовища, неможливість безпосереднього спостереження і виміру основних параметрів об'єкта або велика неточність і неповнота апріорної інформації.

Розв’язання задач в умовах невизначеності стосується таких актуальних сучасних галузей:

– багатовимірних інформаційних систем,
– розпізнання образів,
– штучного інтелекту,
– економічного аналізу,
– аналізу багатовимірного трафіку в комп’ютерних мережах, тощо.

У праці [126] зазначено, що як ефективну та перспективну модель реалізації обчислень у системах означеного класу широко застосовують методи тензорної алгебри, які зводяться до великої кількості паралельних матричних обчислень великої розмірності та обчислень надвеликих систем лінійних алгебричних рівнянь (СЛАР). У роботі [127] відзначено, що подання нечітких змінних у вигляді тензор-змінної дозволяє використовувати матричне і множинне подання об'єкта, з розкладанням матриці тензор-змінної по рядках або по стовпцях. Методи подання нечітких змінних у вигляді тензор-змінної розглянуто у працях [111, 112], у яких визначено передумови створення високопродуктивних паралельних обчислювальних структур для розв’язання задач аналізу й оброблення багатовимірної інформації.

Проблема керування за умов невизначеності належить до таких проблем теорії керування, у межах яких розв’язання навіть окремих задач має велике теоретичне та прикладне значення. Такі задачі описано у праці [128], де як один з прикладів наведено задачу розв’язання СЛАР за умов невизначеності.

Такі СЛАР називаються системами лінійних алгебричних рівнянь з нечіткими змінними (НСЛАР), коли коефіцієнти матриці та її вільні члени подаються інтервалами значень або нечіткими змінними.

У прийнятті рішень в умовах невизначеності потрібно враховувати ту обставину, що наближена постановка задачі усуває раціональність і необхідність застосування точних методів для розв'язання вихідної задачі. У цьому контексті у праці [127] для розв'язання НСЛАР розглянуто спосіб подання вихідної нечіткої задачі у вигляді сукупності чітких задач. У цьому випадку подання нечіткої змінної як тензора дає змогу формулювати чітку блокову СЛАР у вигляді, коли всі її параметри (матриця коефіцієнтів і вільні члени) є матрицями.

Для розв'язання прикладних задач у системах керування, що функціонують в умовах невизначеності, у праці [128] запропоновано методологію розв'язання СЛАР з нечіткими змінними з матричною формою відображення параметрів моделі, яка полягає у формуванні блокової чіткої СЛАР, яка розв'язується стандартними методами. Етапи розв'язання НСЛАР, що виконуються протягом основного циклу керування, такі.

Етап 1. Вирішення проблеми прийняття рішення в умовах невизначеності.

Етап 2. Визначення необхідності розв'язання НСЛАР.

Етап 3. Подання коефіцієнтів матриці та вільних членів нечіткими змінними.

Нечіткі СЛАР являють собою системи рівнянь:

$$\begin{cases} \tilde{a}_{11} \otimes \tilde{x}_1 \oplus \cdots \oplus \tilde{a}_{1m} \otimes \tilde{x}_n \mathrel{\tilde{=}} \tilde{b}_1; \\ \vdots \\ \tilde{a}_{m1} \otimes \tilde{x}_1 \oplus \cdots \oplus \tilde{a}_{mm} \otimes \tilde{x}_n \mathrel{\tilde{=}} \tilde{b}_m. \end{cases} \tag{2.1}$$

$$\tilde{A} \otimes \tilde{x} \mathrel{\tilde{=}} \tilde{b}$$

де $\tilde{a}_{ij}$, $\tilde{x}_i$ і $\tilde{b}_j$ – нечіткі змінні; $\otimes$ і $\oplus$ – знаки скалярного добутку та суми нечітких змінних відповідно, матриця коефіцієнтів $\tilde{A}=[\tilde{a}_{ij}]_{m\times n}\big| i,j=\overline{1,n}$ – $(m\times n)$-нечітка матриця.

Етап 4. Подання нечіткої змінної у вигляді тензор-змінної.

Тензорне подання невизначеності у вигляді тензор-змінної можна сформувати на інтервалі $I_x=[x_1\ x_2\ x_3]$, де $x_1\le x_2\le x_3$ – мінімальне, середнє та максимальне значення інтервалу [128]:

$$I_x=[x_1,x_2,x_3]\to T_x=\begin{pmatrix}0 & 0 & 0\\ x_1 & x_2 & x_3\\ 0 & 0 & 0\end{pmatrix}.$$

Етап 5. Заміна нечіткої змінної у СЛАР її тензорними аналогами. Якщо у НСЛАР (2.1) підставити тензорні аналоги нечіткої змінної, отримаємо нову систему рівнянь, яка матиме блокову матрицю коефіцієнтів, вільні члени якої також будуть матрицями:

$$\begin{cases}[A_{11}]\otimes[X_1]\oplus\cdots\oplus[A_{n1}]\otimes[X_n]=[B_1]\\ \vdots\\ [A_{1n}]\otimes[X_1]\oplus\cdots\oplus[A_{nn}]\otimes[X_n]=[B_n]\end{cases},$$

тоді структура блокової СЛАР набуде вигляду:

$$\begin{cases}\underbrace{\begin{bmatrix}\vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\end{bmatrix}}_{A_{11}}\otimes\underbrace{\begin{bmatrix}\vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\end{bmatrix}}_{x_{11}}\otimes\cdots\otimes\underbrace{\begin{bmatrix}\vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\end{bmatrix}}_{A_{1n}}\otimes\underbrace{\begin{bmatrix}\vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\end{bmatrix}}_{x_{1n}}=\underbrace{\begin{bmatrix}\vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\end{bmatrix}}_{B_1}\\ \cdots\\ \underbrace{\begin{bmatrix}\vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\end{bmatrix}}_{A_{1n}}\otimes\underbrace{\begin{bmatrix}\vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\end{bmatrix}}_{x_{1n}}\otimes\cdots\otimes\underbrace{\begin{bmatrix}\vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\end{bmatrix}}_{A_{nn}}\otimes\underbrace{\begin{bmatrix}\vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\end{bmatrix}}_{x_{nn}}=\underbrace{\begin{bmatrix}\vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\\ \vdots & \vdots & \vdots\end{bmatrix}}_{B_n}\end{cases}. \qquad (2.2)$$

Етап 6. Розв’язання блокової СЛАР стандартними методами. Розв’язання блокових СЛАР у результаті зводиться до стандартних матричних операцій скалярного добутку, множення та суми матриць [111, 112, 129].

Розв'язання блокової системи $[A]\otimes[X]=[B]$ (2.2) має формальний запис

$$[X]=[A]^{-1}\otimes[B],$$

що зумовлює необхідність визначення величини $\det[A]$ матриці $[A]$:

$$[X_i]=\frac{1}{\det[A]}\sum_{j=1}^{n}[B]_j\otimes[A]_{ji}\,\Big|\,i=\overline{1,n}.$$

Як видно з наведеного виразу, для визначення $[X_i]$ потрібно реалізувати операції скалярного добутку і визначити суму матриць.

Нехай матриця A має вигляд

$$A=\begin{bmatrix} A_{11} & \cdots & A_{1q} \\ \vdots & & \vdots \\ A_{p1} & \cdots & A_{1q} \end{bmatrix}\begin{matrix} m_1; \\ \\ m_p, \end{matrix} \qquad (2.3)$$

де $m_1+...+m_p=m$, $n_1+...+n_q=n$, кожний елемент A_{ij} відповідає (i, j)-му блоку, який має розмірність $m_i\times n_{nj}$. Отже, $A=(A_{ij})$ – $(p\times q)$-блокова матриця. Відомо, що для блокових матриць операцію додавання можна реалізувати як додавання звичайних матриць за умови відповідності розмірності блоків [126, 130]. Наприклад, якщо матриця B має вигляд

$$B=\begin{bmatrix} B_{11} & \cdots & B_{1t} \\ \vdots & & \vdots \\ A_{q1} & \cdots & A_{qt} \end{bmatrix}\begin{matrix} r_1; \\ \\ r_q, \end{matrix}$$

то вона поділена на блоки відповідно до матриці A (2.3), тоді сума цих матриць $C=A+B$ є $(p\times q)$-блоковою матрицею $C=(C_{ij})$, де $C_{ij}=A_{ij}+B_{ij}$.

Множення блокових матриць виконується таким чином. Якщо матриці A і B та поділений на блоки добуток $C = AB$ мають такий вигляд:

$$A=\begin{bmatrix} A_{11} & \cdots & A_{1q} \\ \vdots & & \vdots \\ A_{p1} & \cdots & A_{1q} \end{bmatrix}\begin{matrix} m_1; \\ \\ m_p, \end{matrix} \qquad B=\begin{bmatrix} B_{11} & \cdots & B_{1t} \\ \vdots & & \vdots \\ A_{q1} & \cdots & A_{qt} \end{bmatrix}\begin{matrix} r_1; \\ \\ r_q, \end{matrix}$$

$$\begin{matrix} r_1 & & r_q \end{matrix} \qquad \begin{matrix} n_1 & & n_t \end{matrix}$$

$$C=\begin{matrix}\left[\begin{matrix} C_{11} & \cdots & C_{1t} \\ \vdots & & \vdots \\ C_{p1} & \cdots & C_{pt} \end{matrix}\right] \begin{matrix} m_1; \\ \\ m_p, \end{matrix} \\ \begin{matrix} n_1 & & n_t \end{matrix}\end{matrix}$$

тоді

$$C_{ij}=\sum_{k=1}^{q} A_{ik}B_{kj}\Big| i=\overline{1,p},\ j=\overline{1,t}\,.$$

Приклад 1. Застосування методології розв'язання НСЛАР [128].

Розв'яжемо НСЛАР

$$\begin{cases} \tilde{a}_{11}\tilde{x}_1+\tilde{a}_{12}\tilde{x}_2=\tilde{b}_1 \\ \tilde{a}_{21}\tilde{x}_1+\tilde{a}_{22}\tilde{x}_2=\tilde{b}_2 \end{cases}$$

з такими параметрами нечітких змінних:

$a_{11}=[2.5\ \ 3\ \ 3.5];\ a_{12}=-[3.5\ \ 4\ \ 4.5];$

$a_{21}=-[1.5\ \ 2\ \ 2.5];\ a_{22}=[0.7\ \ 1\ \ 1.3];$

$b_1=-[8.5\ \ 10\ \ 11.5];\ b_2=-[3.5\ \ 4\ \ 4.5].$

Розв'язання:

Сформуємо блокову СЛАР:

$$\begin{cases} [a_{11}][x_1]+[a_{12}][x_2]=[b_1] \\ [a_{21}][x_1]+[a_{22}][x_2]=[b_2] \end{cases}. \qquad (2.4)$$

Зміст параметрів блокової СЛАР наведений далі:

$$[a_{11}]=\begin{pmatrix} 0 & 0 & 0 \\ 2.5 & 3.0 & 3.5 \\ 0 & 0 & 0 \end{pmatrix},\ [a_{12}]=\begin{pmatrix} 0 & 0 & 0 \\ -3.5 & -4.0 & -4.5 \\ 0 & 0 & 0 \end{pmatrix};$$

$$[a_{21}]=\begin{pmatrix} 0 & 0 & 0 \\ -1.5 & -2.0 & -2.5 \\ 0 & 0 & 0 \end{pmatrix},\ [a_{22}]=\begin{pmatrix} 0 & 0 & 0 \\ 0.7 & 1.0 & 1.3 \\ 0 & 0 & 0 \end{pmatrix};$$

$$[b_1]=\begin{pmatrix} 0 & 0 & 0 \\ -8.5 & -10.0 & -11.5 \\ 0 & 0 & 0 \end{pmatrix},\ [b_2]=\begin{pmatrix} 0 & 0 & 0 \\ -3.5 & -4.0 & -4.5 \\ 0 & 0 & 0 \end{pmatrix}.$$

Розв'язки блокової СЛАР (2.4), отримані на підставі інтервального методу Гауса, а також модифіковані розв'язки, отримані із застосуванням тензорного базису, наведено в табл. 2.1. Обчислення виконано засобами пакета прикладних програм *Matlab*.

Таблиця 2.1

Розв'язання блокової СЛАР

Розв'язки блокової СЛАР інтервальним методом Гауса [120]	Розв'язки блокової СЛАР у тензорному базисі [128]
$x_1=\begin{pmatrix} 0{,}78 & 0{,}90 & 1{,}02 \\ 1{,}37 & 1{,}58 & 1{,}79 \\ 1{,}96 & 2{,}26 & 2{,}57 \end{pmatrix}$	$y_1=\begin{pmatrix} 0 & 0 & 0 \\ -3{,}51 & 4{,}93 & 5{,}38 \\ 0 & 0 & 0 \end{pmatrix}\to\begin{pmatrix} 0 & 0 & 0 \\ 0 & 4{,}93 & 5{,}38 \\ 0 & 0 & 0 \end{pmatrix}$
$x_2=\begin{pmatrix} 1{,}95 & 2{,}27 & 2{,}59 \\ 1{,}80 & 2{,}09 & 2{,}38 \\ 1{,}64 & 1{,}91 & 2{,}18 \end{pmatrix}$	$y_2=\begin{pmatrix} 0 & 0 & 0 \\ -6{,}57 & 6{,}22 & 7{,}15 \\ 0 & 0 & 0 \end{pmatrix}\to\begin{pmatrix} 0 & 0 & 0 \\ 0 & 6{,}22 & 7{,}15 \\ 0 & 0 & 0 \end{pmatrix}$

У праці [128] узагальнено переваги розглянутої методології розв'язання СЛАР з нечіткими змінними порівняно з відомим інтервальним методом Гауса [120], які обґрунтовують вибір методів дослідження для розв'язання задач дисертаційної роботи:

– метод розв'язання НСЛАР у тензорному базисі дає змогу незалежно від кількості значень в інтервалі подати початкову задачу в невизначеному базисі у вигляді однієї задачі з блоковими параметрами, що істотно спрощує процедуру обчислень;

– застосування відомого інтервального методу Гауса для розв'язання тестових НСЛАР надає на 40% ширший інтервал розв'язання порівняно з

методом розв’язання СЛАР з нечіткими змінними у тензорному базисі, що доводить його високу ефективність через збільшення точності розрахунків;

– метод розв’язання НСЛАР у тензорному базисі має високу здатність до багаторівневого розпаралелення, що створює передумови для ефективного розв’язання цих задач засобами динамічно РКС.

У праці [131] запропоновано модифікацію методу розв’язання НСЛАР для реалізації на ПЛІС, у межах якої блокові СЛАР розв’язуються методом Гауса для чітких СЛАР. У працях [119, 131, 132] розроблено апаратні засоби для розв’язання блокових СЛАР та виконання матричних операцій в тензорному базисі для використання їх як бібліотечних функцій реконфігуровних комп’ютерних систем. Приклади розв’язання задач керування, що зводяться до розв’язання чітких СЛАР, наведено у праці [117]. Приклади використання теорії нечітких множин, що зводяться до виконання операцій з багатовимірними матрицями під час розв’язання сучасних прикладних задач, опубліковані у праці [119]. Як один з прикладів розглянуто особливості виконання операцій скалярної суми ($\oplus$) та добутку ($\otimes$) над нечіткими змінними, поданими тензор-змінними.

Приклад 2. Застосування операцій скалярної суми ($\oplus$) та добутку ($\otimes$) над нечіткими змінними.

Нехай задано дві нечіткі змінні:

<приблизно *5*> $\to \tilde{5} \triangleq \{3/0, 5/1, 7/0\}$, <приблизно *7*> $\to \tilde{7} \triangleq \{3/0, 5/1, 7/1, 9/0\}$

,

де $\triangleq$ – знак «дорівнює за визначенням».

Обчислити в тензорному базисі:

<приблизно 5> $\oplus$ <приблизно 7>,

<приблизно 5> $\otimes$ <приблизно 7>.

Розв’язання:

Визначимо тензор-змінні <*приблизно 5*> та <*приблизно 7*>:

$$\tilde{5} = \{3/0, 5/1, 7/0\} \rightarrow \mathrm{T}^{(5)} = [3\,5\,7] \otimes [0\,1\,0]^{\mathrm{T}} = \begin{pmatrix} 0 & 0 & 0 \\ 3 & 5 & 7 \\ 0 & 0 & 0 \end{pmatrix};$$

$$\tilde{7} = \{3/0, 5/1, 7/1, 9/0\} \rightarrow \mathrm{T}^{(5)} = [3\,5\,7\,9] \otimes [0\,1\,1\,0]^{\mathrm{T}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 3 & 5 & 7 & 9 \\ 3 & 5 & 7 & 9 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Тоді

$$T^{(5)} \oplus T^{(7)} = \begin{pmatrix} 0 & 0 & 0 \\ 3 & 5 & 7 \\ 0 & 0 & 0 \end{pmatrix} \oplus \begin{pmatrix} 0 & 0 & 0 & 0 \\ 3 & 5 & 7 & 9 \\ 3 & 5 & 7 & 9 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} \left(\mathrm{T}^{(5)}\right)_3^3 & (0)_4^3 \\ (0)_3^4 & \left(\mathrm{T}^{(7)}\right)_4^4 \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 3 & 5 & 7 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 & 0 \\ 3 & 5 & 7 & 9 \\ 3 & 5 & 7 & 9 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{pmatrix};$$

$$T^{(5)} \oplus T^{(7)} = \begin{pmatrix} 0 & 0 & 0 \\ 3 & 5 & 7 \\ 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 & 0 \\ 3 & 5 & 7 & 9 \\ 3 & 5 & 7 & 9 \\ 0 & 0 & 0 & 0 \end{pmatrix} =$$

$$= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 9 & 15 & 21 & 27 & 15 & 25 & 35 & 45 & 21 & 35 & 49 & 63 \\ 9 & 15 & 21 & 27 & 15 & 25 & 35 & 45 & 21 & 35 & 49 & 63 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

У працях [111, 112, 119, 128] показано, що застосування теорії нечітких множин дозволяє виконувати всі операції з нечіткими змінними на рівні формальних моделей матричного обчислення без необхідності застосування різноманітних евристичних прийомів та принципів, що створює передумови для ефективного розв’язання цих задач засобами динамічно РКС.

2.1.2. Особливості розв’язання задач керування в режимі реального часу. Під час процесів керування в режимі реального часу розв’язуються траєкторні задачі, що зводяться до реалізації різних методів інтерполяції:

– поліноміальна апроксимація,
– розкладання в ланцюгові дроби,
– таблично-алгоритмічні,
– ітераційні, тощо.

Моделювання технічних і технологічних процесів керування зводяться до розв’язання

– систем лінійних алгебричних рівнянь,
– диференційних рівнянь, тощо.

У праці [51] визначено, що в системах керування реального часу більшість задач мають дрібнозернисту структуру, за якої найвищий ступінь розпаралелення може бути досягнуто, коли кожній вершині графу відповідає окрема операція, а саме: обчислення елементарних функцій або перетворення координат. При цьому максимально збільшується кількість паралельних гілок графу, що приводить до використання великої кількості паралельно працюючих обчислювальних елементів. Проте на швидкість обробки інформації негативно впливають витрати часу на обмін інформацією між паралельними гілками. У праці [6] зазначено, що розв’язання дрібнозернистих задач засобами сучасних високопродуктивних комп'ютерів забезпечує лише 1% від їх пікової продуктивності.

Зважаючи на те, що в системах керування реального часу важливішою характеристикою є не продуктивність виконання ряду обчислювальних

операцій, а час обчислення певної функції, у працях [51, 110] визначено поняття макрофункції як набір програмних або апаратних засобів для обчислення певної функції.

Актуальною парадигмою підвищення ефективності систем керування в реальному часі є пришвидшення виконання таких макрофункцій на всіх рівнях обчислювальної системи. У праці [51] як підвищення ефективності систем керування реального часу розглядається реалізація спеціалізованих обчислювальних структур для виконання цих макрофункцій, а у працях [6, 22, 107, 108, 134 – 138] – реалізація парадигми апаратного пришвидшення обчислень у системах керування реального часу на базі реконфігуровних обчислювальних пристроїв та систем на ПЛІС.

Засоби для реалізації дрібнозернистих обчислювальних задач у режимі реального часу запропоновано в таких працях: у [137, 139] – спеціалізовані матричні обчислювачі на ПЛІС, які забезпечують значне пришвидчення обчислень, але характеризуються високими апаратурними витратами; у [140] –спеціалізований конвеєрний обчислювач на ПЛІС, на підставі дослідження якого визначено, що час виконання обчислень значно залежить від часу заповнення конвеєра. Теорію побудови мультипроцесорних обчислювальних систем для розв’язання задач керування в реальному часі подано у працях [51, 134, 141, 142], розвиток якої із застосуванням сучасної перепрограмовної елементної бази створює передумови для підвищення ефективності систем керування в реальному часі. У цьому контексті основною проблемою, що залишається актуальною для реалізації методів та засобів підвищення ефективності дрібнозернистих обчислень, є значні накладні витрати в процесі обробки даних, зокрема витрати логічних ресурсів і ресурсів внутрішньої пам’яті ПЛІС.

Апаратурні витрати для виконання матричних операцій, методи визначення яких наведено у працях [143 – 145], пропорційні значенню n^2 функціональних вузлів, а табличні обчислювачі потребують використання великої ємності внутрішньої пам’яті ПЛІС, а саме $(2^2 \times n)$ біт пам’яті, де n –

розрядність операндів. Для зменшення апаратурних витрат матричних обчислювачів використовують комбіновані матрично-табличні обчислювачі [144, 145], які базуються на використанні елементів пам’яті. Для зменшення використання ємності постійної пам'яті в табличних обчислювачах у працях [133, 141] запропоновано використання таблично-алгоритмічних способів обчислень, які поєднують табличні методи з іншими, наприклад, ітераційними або поліноміальної апроксимації.

Таблично-алгоритмічні способи зводяться до задачі обчислення багаточленів. У працях [133, 141] наведено оцінку апаратурних витрат на реалізацію відомих методів обчислення багаточленів.

Таблично-алгоритмічні обчислювачі для реалізації прямого способу обчислення багаточленів вигляду

$$P(X) = a_0 + a_1 X + a_2 X + ... + a_m X^m \qquad (2.5)$$

ґрунтуються на побудові таблиць значень $X^2, X^3, ..., X^m$ [144, 145]. Для побудови обчислювачів потрібно $(m-1)$ блоків зведення в степінь, m помножувачів і суматорів. Обчислення багаточленів за схемою Горнера [134 – 145] потребує m помножувачів і m суматорів. Обчислення за методом Горнера «вищих порядків» створює передумови для підвищення швидкодії обчислень за рахунок реалізації незалежних гілок алгоритму обчислення. Наприклад діленням багаточлена (2.5) на багаточлен $f(x) = x - x_0^2$ отримаємо схему Горнера «другого порядку»:

$$P(x) = \left(...\left(a_m \cdot x^2 + a_{m-2}\right)\cdot x^2 + ...\right)\cdot x^2 + a_0 + \left(\left(...\left(a_{m-1}x^2 + a_{m-3}\right)x^2 + ...\right)x^2 + a_1\right)x .$$

Утім апаратурні витрати не змінюються – використовується m помножувачів і m суматорів, але реалізуються дві незалежні гілки обчислення. Також відомий метод Естріна [134, 144], який дозволяє ефективне розпаралелення обчислювального процесу і потребує m помножувачів, m суматорів, $]m/2[$ блоків зведення в степінь.

У працях [133, 141] запропоновано способи обчислення багаточленів, що дає змогу зменшити апаратні витрати порівнянно з класичними способами.

Показано, що запропоновані способи обчислення зменшують апаратурні витрати порівняно з методами Горнера і Естріна у 3 рази, а методом прямого обчислення багаточленів – у 1,5 разу. Запропоновані способи ґрунтуються на попередньому обробленні коефіцієнтів багаточлена і поданні багаточлена у вигляді суми функцій однієї змінної. Приклад застосування одного з таких способів наведено далі.

Приклад 3. Зведемо багаточлен (2.5) до вигляду

$$P(x)=\frac{\left(\beta+(a_1+x)^2+\left(a_2+x^2\right)^2+...+\left(a_m+x^m\right)^2-F(x)\right)}{2}, \qquad (2.6)$$

де $\beta=2a_0-a_1^2-a_2^2-...-a_m^2$, $F(x)=x^2+x^4+...+x^{2m}$. Рівність виразів доводиться підстановленням значень β і $F(x)$ у вираз (2.6). Коефіцієнт β попередньо обчислюється для заданого багаточлена. Пристрій для обчислення багаточлена відповідно до виразу (2.6) містить такі блоки: П1 – блок для збереження таблиці функції $F(x)$; К і К1 – блоки зведення в квадрат (квадратори); СМ – суматори, БП – блок підсумовування, П – блоки зведення у степінь. Структуру відомого пристрою показано на рис. 2.1.

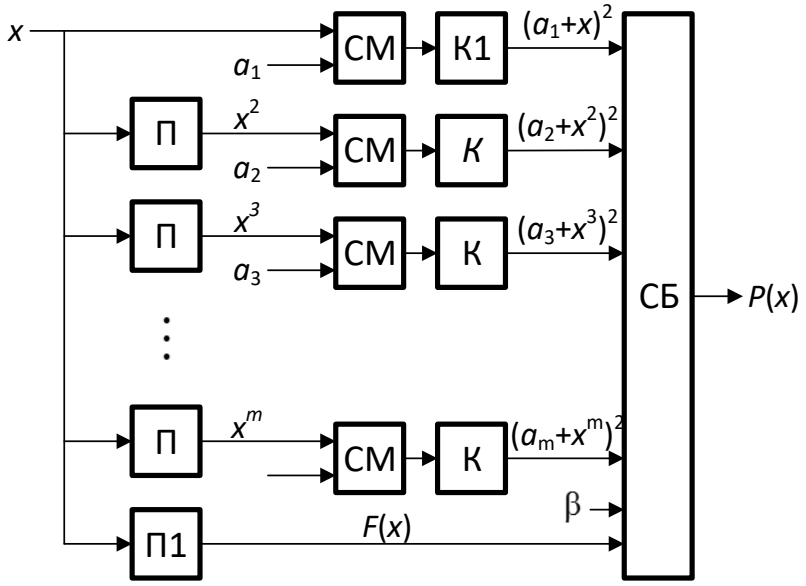


Рис. 2.1. Обчислювач для обчислення багаточлена [133, 141]

Розглянутий спосіб значно зменшує кількість обчислювальних операцій, але потребує використання пам’яті для зберігання коефіцієнтів, що є іншим критичним фактором для реалізації на ПЛІС.

Для реалізації обчислювача, функціональну структуру якого зображено на рис 2.1, потрібна така ємність постійної пам’яті, розрахунки якої обґрунтовано у працях [133, 141]:

$$V_1 = 2^{n+2+\lceil \log_2(m+2) \rceil} \cdot (n+1+\lceil \log_2(m+2) \rceil);$$

$$V_2 = 2^{n+1} \cdot (n+1+\lceil \log_2(m+2) \rceil);$$

$$V_3 = 2^{n} \cdot (n+1+\lceil \log_2(m+2) \rceil);$$

$$V_4 = 2^{n} \cdot (n+1+\lceil \log_2(m+2) \rceil) + \lceil \log_2 m \rceil,$$

де V_1, V_2, V_3, V_4 – відповідно ємність блоків К, К1, П, П1 (у бітах). Вважаємо, що коефіцієнти a_i і аргумент x змінюються в межах $0 \le a_i < 1$, $0 \le x < 1$ і подані n розрядами двійкових чисел.

Загальна ємність постійної пам’яті становить

$$V = (m-1)V_1 + V_2 + (m-1)V_3 + V_4. \qquad (2.7)$$

Для подолання проблем надмірного використання пам’яті у працях [133, 141] запропоновано послідовно-паралельний спосіб обчислення багаточленів, який не потребує попередньої підготовки коефіцієнтів, а розраховує їх у процесі обчислень. Такий спосіб є універсальним для різних багаточленів і дозволяє зменшити апаратні витрати порівняно з паралельним пристроєм. Ємність постійної пам’яті в цьому випадку становить

$$V = V_1 + V_3(m-1) + V_4,$$

що менше ніж ємність паралельного пристрою (2.7). Окрім того, зменшується обсяг використання обчислювального ресурсу – кількість суматорів зменшується до одного суматора замість m, один суматор використовується для реалізації блока підсумовування (рис. 2.1).

Наведена оцінка апаратурних витрат дозволила виявити найбільш ефективні способи щодо реалізації багаточленів в обчислювальних системах

на сучасній перепрограмовній елементній базі для розв’язання сучасних задач керування в режимі реального часу, які, зокрема, характеризуються великою розмірністю.

У структурі пристрою (рис. 2.1) помножувачі реалізовані за допомогою постійної пам’яті, що критично для застосування в реконфігуровних комп’ютерах на ПЛІС. У працях [133, 138, 141] запропоновано реалізацію помножувачів та квадраторів для пристрою (2.2) за матричними схемами, що дозволяє в 1,5 разу зменшити використання пам’яті та відкриває перспективи ефективного використання РКС для реалізації послідовно-паралельних способів обчислення багаточленів з використанням убудованих на ПЛІС функціональних елементів і матричних обчислювачів замість комірок постійної пам’яті. У працях [34, 117, 131] розглянуто реалізацію прикладних задач на ПЛІС у межах розв’язання задач керування. У праці [146] запропоновано спосіб зменшення апаратурних витрат під час зберігання поточної інформації у процесі обчислення через реалізацію частини кеш-пам’яті процесорного ядра в зовнішній пам’яті.

У працях [51, 147, 148] показано, що ефективним підходом до пришвидшення реалізації дрібнозернистих алгоритмів є використання динамічних засобів розподілу операцій між обчислювальними вузлами, заснованих на моделі обчислень, що керуються потоком даних. Основною функціональною особливістю є те, що команди виконуються не в описаній програмою послідовності, а в міру надходження повної інформації про команду і операнди. Таким чином, визначальним для виконання команди є доступність даних, а не заданий програмою порядок виконання команд. Ця особливість дозволяє розподіляти операції динамічно у процесі обчислення на апаратному або мікропрограмному рівні. Під час підготовки завдання немає потреби описувати процедури синхронізації процесів та обміну даними між гілками алгоритмів у прямій формі. Це створює передумови для пришвидшення процесу обробки інформації, зокрема за рахунок його

реалізації на програмно-апаратному рівні обчислювального модуля реконфігуровної комп’ютерної системи.

Відому обчислювальну систему, що реалізує модуль обчислень, керованих потоком даних, описано у працях [51, 148]. Дані для обчислень готуються на основі потокового графу, вершини якого означають операції, а дуги – потоки даних, таким чином, що зв'язок між командами здійснюється лише за даними. Операція описується актором (*actor*), який передається кортежем:

$$A = \langle n, I, N, P \rangle, \tag{2.8}$$

де n – ідентифікатор актора; I – функція перетворення даних; N – ім'я актора, для якого результат передається як операнд; P – сукупність ознак операнда. Дані передаються у вигляді дескрипторів (*descriptor*), які мають вигляд

$$D = \langle n, A, N, P \rangle, \tag{2.9}$$

де A – адресна інформація.

Потокова обчислювальна система [51] складається із середовища формування команд (СФК), декількох однотипних обчислювальних модулів і блока керування. Зв'язок між структурними компонентами та зовнішнім оточенням здійснюється через комутаційне середовище. У систему надходять актори і дані. Із використанням певних засобів і алгоритмів формуються команди, які виконуються у вільному обчислювальному модулі.

Оскільки немає потреби планувати обчислювальний процес і обчислювальні ресурси в обчислювальних системах, керованих потоком даних, створюються передумови для ефективного вирішення проблеми підвищення ефективності паралельної обробки інформації в РКС за рахунок апаратного пришвидшення обчислень та апаратної реалізації процесу розпаралелення та розподілу задач на реконфігуровне обчислювальне середовище і його автоматизації.

Спосіб автоматичного розпаралелення задач, розроблений для традиційних обчислювальних систем з фіксованою структурою

обчислювального середовища описано у праці [51]. Запропоноване безпріоритетне обслуговування заявок на виконання завдань спростило структуру пристрою та механізми їх формування і розподілення. Це створило передумови для реалізації автоматичного керування обробкою інформації на апаратному рівні. У праці [51] функція керування розподіленням задач покладена на автомат з жорсткою логікою, що значно зменшує час формування заявок.

На підставі аналізу основних положень класичної та сучасної теорій обчислювальних систем, що керуються потоком даних [51, 147 – 153] підсумуємо таке.

Реалізація засобів автоматичного розподілу задач на базі моделі обчислень, керованих потоком даних, дозволяє:

– зменшити складність засобів керування обробкою інформації;
– підвищити ефективність багатозадачного режиму функціонування;
– підвищити ефективність урахування параметрів фізичного рівня ПЛІС;
– спростити механізм формування заявок;
– реалізувати апаратне пришвидшення функціональних процесів;
– зменшити непродуктивне використання ресурсів пам’яті;
– зменшити обсяг передаваних непродуктивних даних;
– ефективно реалізувати прихований паралелізм;
– автоматизувати обчислювальний процес на апаратному рівні.

Таким чином, у цілому створюються передумови для підвищення ефективності реконфігуровних комп’ютерних систем у процесі розв’язання задач керування великої розмірності, у тому числі в режимі реального часу.

2.2. Визначення формальних параметрів основних функціональних елементів процесу обробки даних

Обчислювані задачі, що є вихідними для досліджень у дисертаційній роботі, належать до типу паралельних задач зі змішаним типом паралелізму,

для опису яких зазвичай застосовують модель програмування *M*-задач (макрозадач) [86, 107, 110, 154]. *M*-задача задається макрографом потоку даних *MDG* (*Macro Dataflow Graphs*) [109, 155], у вершинах якого розміщуються макрофункції або макрозавдання, а ребра вказують на залежності між вершинами.

Для визначення апаратної реалізації на ПЛІС певного завдання використовується абстрактне поняття *апаратної задачі*, що є деякою функцією (функціональним ядром) обчислювального алгоритму, яка синтезована в цифрові схеми. В алгоритмах зі змішаним типом паралелізму кожній апаратній задачі ставиться у відповідність завдання, що є певним обсягом роботи, який відповідає вершині макрографу алгоритму обчислювальної задачі.

Традиційна модель апаратної задачі, описана у працях [44, 46] характеризується розміром, формою відображення і часом виконання. Розмір апаратної задачі визначає просторові вимоги до динамічної ділянки ПЛІС [27]. Форма відображення залежить від моделі розміщення задач у реконфігуровній ділянці [27], зазвичай для всіх моделей розміщення апаратна задача має форму прямокутника.

Конфігураційні дані – це дані для налаштування цифрової схеми на кристалі ПЛІС, що у вигляді файлів з бітовим даними БКД.

Конфігурація кристала ПЛІС – це налаштування (програмування або прошивання) кристала ПЛІС або деякої ділянки кристала ПЛІС для апаратного виконання певної обчислювальної функції, що відповідає виконанню апаратної задачі [13, 15, 24].

Реконфігурація обчислювального середовища ПЛІС – це зміна конфігурації кристала ПЛІС для виконання нової апаратної задачі [116, 113].

Процес виділення реконфігуровних обчислювальних ресурсів для виконання завдання являє собою конфігурування (програмування) обчислювального середовища – налаштування відповідної цифрової схеми (апаратної задачі) у певній ділянці реконфігуровного обчислювального

середовища. Новостворена обчислювальна структура на реконфігуровному обчислювальному середовищі називається функціональним блоком (ФБ).

Конфігураційні дані апаратної задачі – це дані для налаштування ФБ на кристалі ПЛІС.

Для формалізації процесу обробки інформації в реконфігуровних комп'ютерних системах використовуються позначення та визначення основних функціональних елементів.

Алгоритм вихідної обчислювальної задачі поданий ярусно-паралельною формою (ЯПФ) макрографу (далі графу) [154 – 156] $G_M = \{N_M, D_M\}$, де N_M – множина вершин, що відповідають завданням, а D_M – множина ребер, що визначають залежності між завданнями. Приклад ЯПФ графу алгоритму (ГА) обчислювальної задачі ілюструє рис. 2.2, а взаємозв'язок між основними функціональними елементами обчислювального процесу – рис. 2.3.

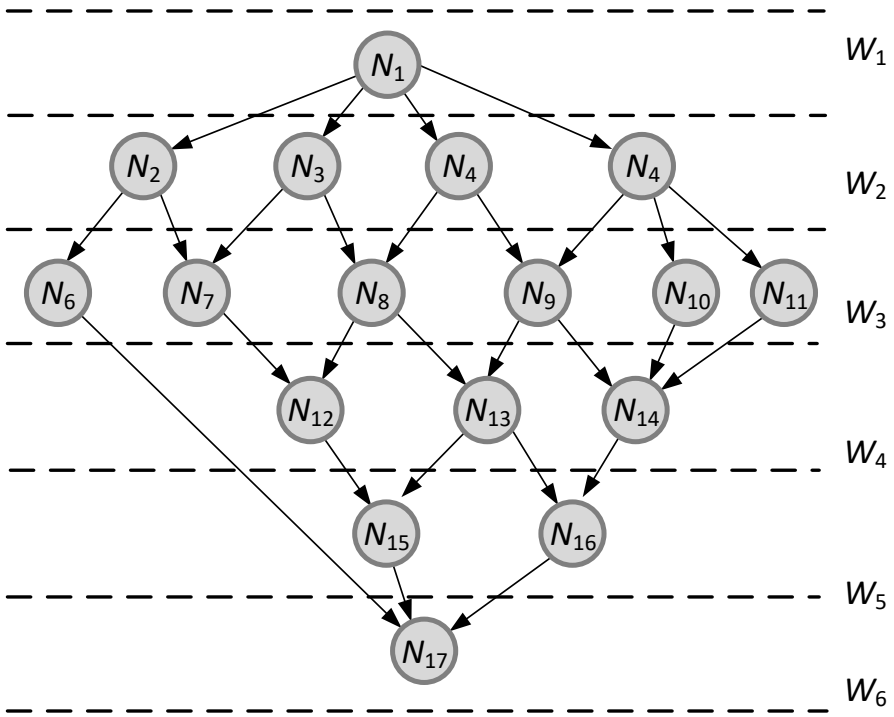


Рис. 2.2. Приклад ЯПФ графу алгоритму обчислювальної задачі

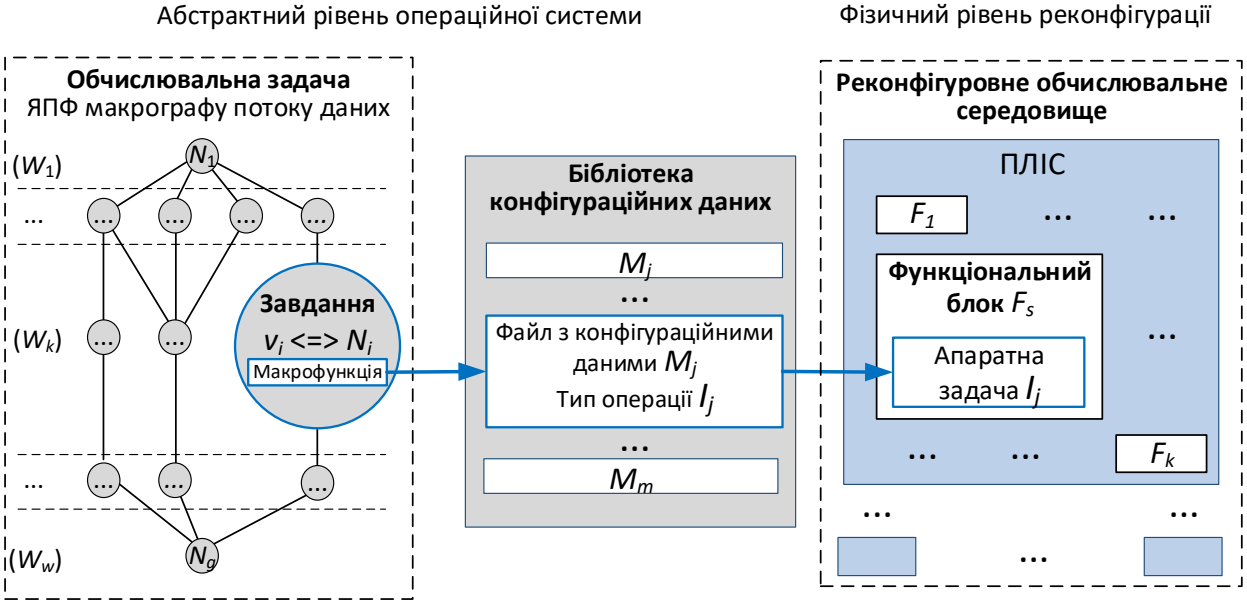


Рис. 2.3. Взаємозв'язок основних функціональних елементів реконфігуровної комп'ютерної системи

Параметри обчислювальних задач:

$N_M = \{N_1, N_2, N_3, ..., N_i, ..., N_g\}$ – множина завдань у вершинах ГА G_M;

N_i – унікальний ідентифікатор завдання у вершині ГА G_M;

$i = \overline{1, g}$ – індекс завдання у вершині ГА G_M;

g – кількість вершин ГА обчислювальної задачі;

W_k – унікальний ідентифікатор ярусу ЯПФ ГА G_M обчислювальної задачі;

$k = \overline{1, w}$ – індекс ярусу;

w – кількість ярусів ЯПФ ГА обчислювальної задачі.

Параметри обчислювального середовища на ПЛІС:

F_s – унікальний ідентифікатор ФБ на поверхні реконфігуровної ділянки ПЛІС;

$s = \overline{1, k}$ – індекс ФБ;

k – кількість ФБ на поверхні реконфігуровної області ПЛІС.

Параметри апаратної задачі.

Кожна апаратна задача $Task_j$ визначається як вектор [44, 46]:

$$Task_j = \{S_j, T_{SUM\,j} | (T_j + R_j), I_j, N_j\}, \quad (2.10)$$

де S_j – площа прямокутника, що містить задачу $Task_j$; R_j – час, витрачений на пересилання конфігураційних даних і реконфігурацію обчислювальної структури на ПЛІС для виконання апаратної задачі; $T_j = T_{HW\,j}$ – час виконання апаратної задачі на поверхні реконфігуровної області з урахуванням часу введення-виведення даних для обчислення; $T_{SUM\,j} = T_j + R_j$ – загальний час виконання завдання; I_j – тип операції/функції, яку реалізує апаратна задача; $j \mid j = \overline{1,m}$ – індекс апаратної задачі; m – кількість апаратних задач в БКД.

2.3. Визначення критеріїв ефективності динамічно реконфігуровних комп'ютерних систем

У працях [86, 107, 108] наведено спосіб визначення основних критеріїв ефективності реконфігуровних комп'ютерних систем на підставі наступного показника пришвидшення обчислень

$$\rho = \frac{T_{SW}}{R + T_{HW}}, \quad (2.11)$$

де T_{SW} – час обчислення завдання на процесорному ядрі; T_{HW} – час обчислення завдання апаратними засобами ПЛІС; R – час реконфігурації обчислювального середовища на ПЛІС. Показник пришвидшення (2.11) дорівнює відношенню часу обчислення задачі програмними засобами до суми часу апаратного обчислення завдання і часу реконфігурації обчислювальної структури.

Час реконфігурації – це час передавання конфігураційних даних із зовнішніх сховищ до інтерфейсів ПЛІС і час налаштування обчислювального середовища на поверхні кристала ПЛІС. Час реконфігурації R є вагомою

складовою накладних видатків часу в процесі реконфігурації обчислювального середовища на ПЛІС. Показник пришвидшення обчислень вимірює позитивний ефект від апаратного пришвидшення обчислювального процесу, що лежить в основі принципу функціонування реконфігуровних комп'ютерних систем.

Показник пришвидшення (2.11) визначено для статично реконфігуровних обчислювальних систем [86, 107]. У цьому випадку планування і налаштування обчислювальних ресурсів, а також розподілення завдань на обчислювальне середовище на ПЛІС розглядаються як неподільний процес, що здійснюється на рівні операційної системи і визначається параметром R у виразі (2.11). Для динамічно РКС застосування такого показника пришвидшення неефективне, оскільки він не дозволяє оцінити й оптимізувати накладні витрати часу, зумовлені великою часовою складністю процесів планування обчислень і розподілу реконфігуровних обчислювальних ресурсів. У працях [27, 90] обґрунтовано, що велика складність процесів керування в динамічно РКС потребує врахування фізичних параметрів кристалів ПЛІС на всіх етапах організації обчислювального процесу та змінюваних умов відображення задач.

Для визначення критеріїв ефективності динамічно реконфігуровних комп'ютерних систем пропонується модифікація відомого показника пришвидшення з урахуванням часової складності процесів планування обчислень і розподілу реконфігуровних обчислювальних ресурсів ($T_{CONTROL}$):

$$\rho = \frac{T_{SW}}{[T_{CONTROL} + R] + T_{HW}}. \qquad (2.12)$$

Згідно з модифікованим показником пришвидшення одним з критеріїв ефективності динамічно реконфігуровних комп'ютерних систем є апаратне пришвидшення обчислень порівняно з програмною реалізацією за рахунок зменшення часу відображення завдань на реконфігуровне обчислювальне середовища на ПЛІС $T_{MAPP} = T_{CONTROL} + R$, або зменшення часу виконання завдання на апаратурі ПЛІС T_{HW}.

Залежність функції мінімізації часу відображення від затримань процесу організації обчислень і реконфігурації обчислювального середовища на ПЛІС виражатиметься таким чином:

$$\min(T_{MAPP}) = \min(T_{CONTROL}) + \min(T_{COMM}) + T_{CONFIG}, \qquad (2.13)$$

де T_{MAPP} – час відображення завдань на реконфігуровне обчислювальне середовище; T_{COMM} – час передавання конфігураційних даних; T_{CONFIG} – час конфігурації (прошивання) кристала ПЛІС.

Впливати на час конфігурування кристала ПЛІС складно, бо цей час залежить від технічних показників мікросхеми. Для визначення часу часткової реконфігурації обчислювального середовища на ПЛІС у працях [15, 68] описано спосіб визначення часу конфігурації певної ділянки кристала ПЛІС. За відомим способом T_{CONFIG} визначається як сума часу ініціалізації інтерфейсу конфігурації ПЛІС і часу передавання масиву конфігураційних слів через інтерфейс конфігурації ПЛІС:

$$T_{CONFIG} = T_{INI} + T_{BIT}. \qquad (2.14)$$

Час ініціалізації T_{INI} – це час читання і опрацювання першого конфігураційного слова бітового потоку. Час передавання T_{BIT} – це час, витрачений на пересилання всіх інших конфігураційних слів бітової послідовності через конфігураційний інтерфейс пристрою:

$$T_{BIT} = N_{BIT} \times T_W,$$

де N_{BIT} – кількість конфігураційних слів у бітовій послідовності; T_W – час пересилання одного слова даних через конфігураційний інтерфейс ПЛІС.

У результаті на підставі виразу (2.14) час конфігурації ділянки кристала ПЛІС визначиться як

$$T_{CONFIG} = T_{INI} + N_{BIT} + T_W.$$

Параметри T_{INI} і T_W є сталими величинами і залежать від технічних показників використовуваних сімейств мікросхем ПЛІС. Таким чином, під час

часткової реконфігурації ПЛІС час конфігурування кристала залежить від розміру бітового потоку конфігураційних даних, що завантажуються [15, 68].

Вираз (2.14) не враховує час передавання слів бітової послідовності конфігураційних даних через системне комунікаційне середовище РКС, його архітектуру і властивості. Цей час визначається виразом (2.13) як комунікаційні затримання під час передавання конфігураційних даних T_{COMM} . Комунікаційні затримання визначають час, витрачений на передавання конфігураційних даних з віддалених бібліотек в інтерфейси конфігурації кристалів ПЛІС, що є непродуктивним часом в обчислювальному процесі і критичним параметром для оцінювання ефективності динамічно реконфігуровних комп'ютерних систем.

Таким чином, проблема підвищення ефективності обчислювального процесу в реконфігуровних комп'ютерних системах зводиться до мінімізації часу відображення завдань на реконфігуровне обчислювальне середовище відповідно до виразу (2.13) через зменшення комунікаційних витрат T_{COMM} під час реконфігурації і зменшення часової складності процесу керування обчисленнями та реконфігуровними ресурсами $T_{CONTROL}$, а також відповідно до виразу (2.12) зменшення безпосередньо часу обчислення апаратної задачі на ПЛІС T_{HW} .

Функціональні особливості реалізації реконфігуровних комп'ютерних систем надають широкі можливості для скорочення комунікаційних затримань під час реконфігурації. На виникнення комунікаційних затримань впливають такі фактори:

- структура обчислювальної системи;
- організація адресного простору;
- структура комунікаційного середовища;
- місце розташування конфігураційних даних;
- кількість конфігураційних даних.

2.4. Формалізація адаптивного відображення задач на реконфігуровне обчислювальне середовище на ПЛІС

2.4.1. Спосіб скорочення критичного часу виконання обчислювальних задач у динамічно реконфігуровних комп'ютерних системах. Критичними параметрами обчислювальних задач, поданих ЯПФ ГА, які впливають на ефективність обчислень у реконфігуровних комп'ютерних системах є критичний шлях і ширина графу алгоритму. Критичний шлях визначає максимальний (критичний) час виконання алгоритму обчислювальної задачі, який згідно з виразом (2.11) обмежується таким співвідношенням:

$$(R + T_{HW}) < T_{SW}\,. \tag{2.15}$$

Ширина графу алгоритму обмежується наявним обсягом обчислювальних ресурсів ПЛІС і визначається таким

$$\max[H_k] \,|\, k = \overline{1,w} \le n, \tag{2.16}$$

де H_k, – кількість вузлів на ярусі ЯПФ ГА W_k; $k = \overline{1,w}$ – індекс ярусу; w – кількість ярусів; n – кількість функціональних блоків на ПЛІС (див. рис. 2.2).

У працях [154, 156] описано методи модифікації графів і засоби підвищення ефективності відображення задач для фіксованих обчислювальних структур з необмеженим обсягом обчислювальних ресурсів. Застосування таких методів неефективне за наявності функціональних та апаратних обмежень, якими характеризуються динамічно реконфігуровні комп'ютерні системи. Отже, зменшити критичний шлях за рахунок розширення ЯПФ графу неможливо через апаратурні обмеження реконфігуровної ділянки ПЛІС, а зменшити ширину графу за рахунок збільшення критичного шляху неможливо через часові обмеження функціонального процесу в РКС. У праці [155] запропоновано і досліджено метод модифікації графів для статично реконфігуровних комп'ютерних систем. Метод базується на трансформації ГА обчислювальної задачі в межах оптимальної відповідності ширини ГА до довжини його критичного шляху

відповідно до виразів (2.15) і (2.16). Застосування методу передбачає наявність необхідного часу для реалізації і апаратних ресурсів ПЛІС, що характерно для статично РКС. Запропонований метод характеризується значними часовими витратами під час застосування в динамічно реконфігуровних комп'ютерних системах, зважаючи, окрім того, на змінні умови відображення завдань на реконфігуровне обчислювальне середовище.

У праці [157] запропоновано новий спосіб трансформації графу алгоритму обчислювальних задач у динамічно РКС, що дозволяє підвищити ефективність їх відображення на реконфігуровне обчислювальне середовище через скорочення критичного часу виконання. Для скорочення критичного часу виконання обчислювальних задач запропоновано комплексне застосування технологій повторного використання ресурсів ФБ, що запобігає повторному передаванню конфігураційних даних, і попередньої реконфігурації обчислювального середовища. Наведемо основні особливості реалізації запропонованого способу скорочення критичного часу виконання задач.

Згідно з виразами (2.11) і (2.12) визначено, що час виконання кожного завдання у вершині графу складається з двох складових: часу реконфігурації обчислювального середовища і часу виконання відповідної апаратної задачі на апаратурі ПЛІС. Це проілюстровано часовою діаграмою (рис. 2.4) на прикладі абстрактного ГА G_M обчислювальної задачі.

Кожному завданню N_i у вершині ЯПФ ГА G_M відповідає апаратна задача $Task_j$, яка характеризується часом виконання $T_{COUNT\,j} = T_j + R_j$ відповідно до виразу (2.10), де $T_j = T_{HW\,j}$. На кожному ярусі ЯПФ ГА (рис. 2.4) позначено час виконання завдань апаратними засобами реконфігуровного обчислювального середовища T і час налаштування обчислювального середовища на виконання кожного завдання R.

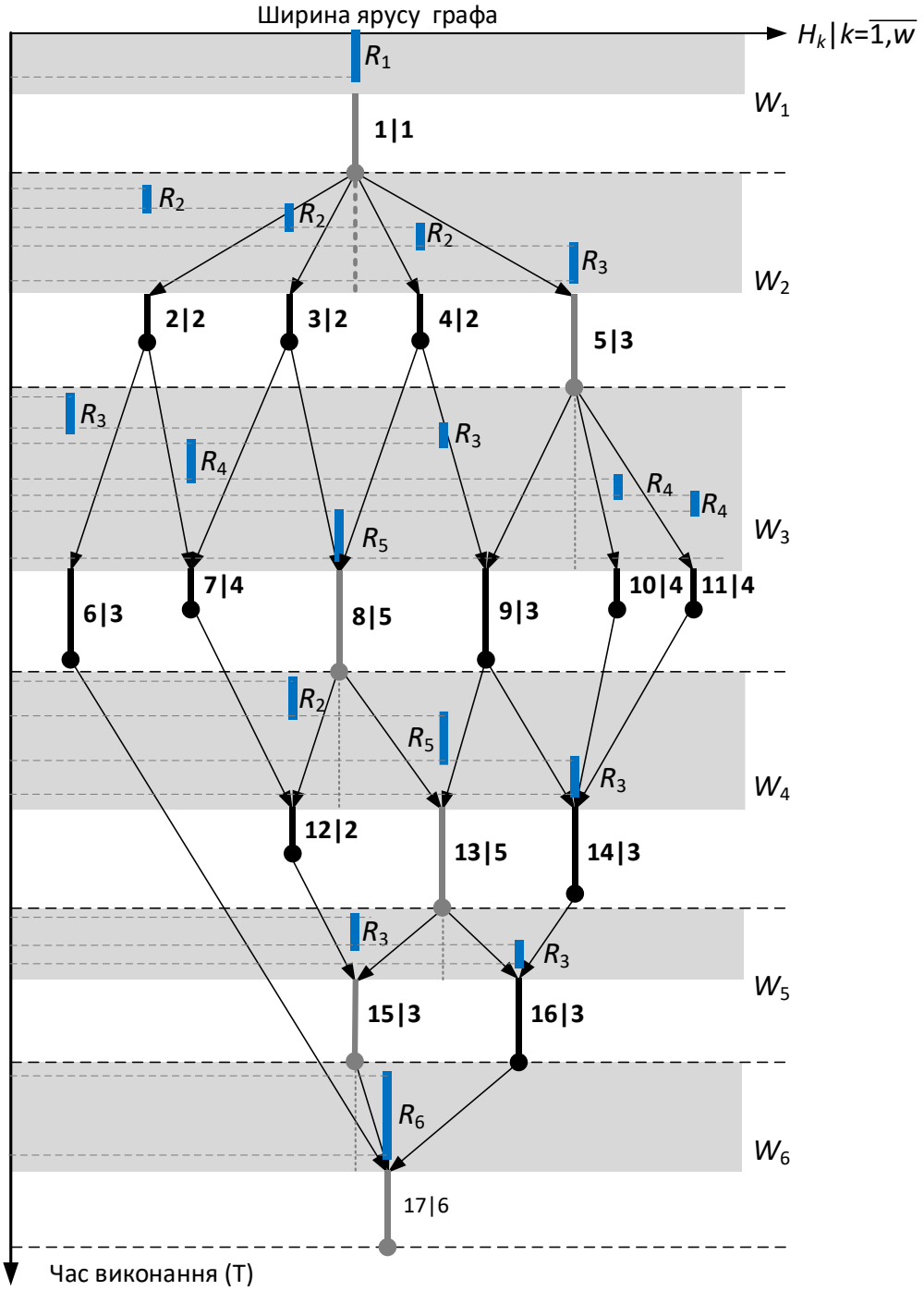


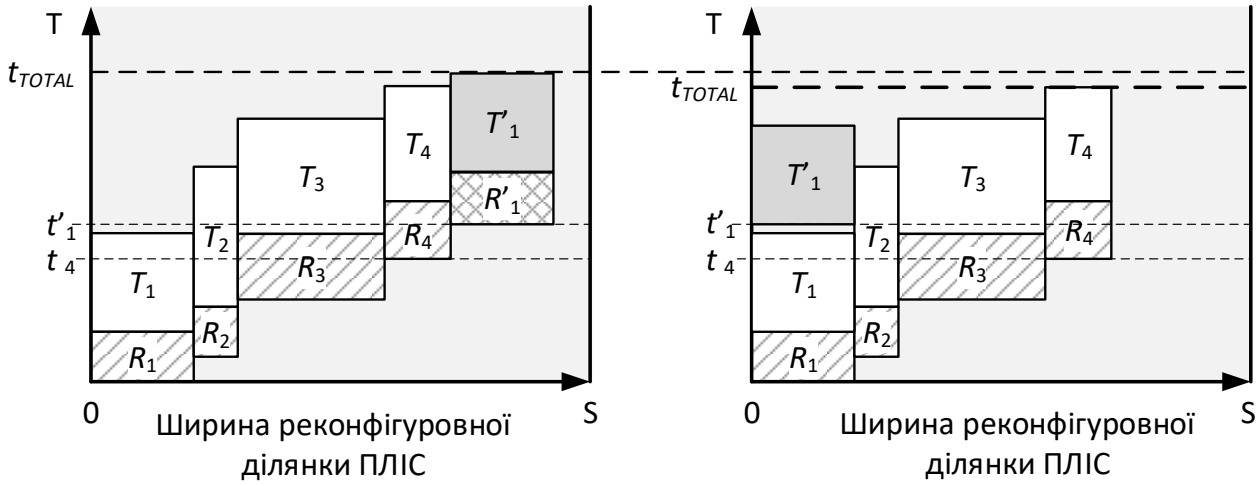
Рис. 2.4. Приклад виконання алгоритму обчислювальної задачі під час стандартного процесу реконфігурації: ***i*|*j*** – номер вершини N_i | тип операції I_j; – час реконфігурації R_j; – час виконання завдання T_j; – час виконання завдання на критичному шляху T_j; – непродуктивний час

Зменшення критичного часу виконання обчислювальної задачі, поданої ЯПФ графу, полягає у видаленні з критичного шляху графу непродуктивної

складової часу відповідно до виразу (2.12). Часова складність алгоритмів керування паралельними обчисленнями і реконфігуровними ресурсами та час прошивання мікросхеми ПЛІС розглядаються, як сталі величини, тому умовно не враховуються на етапі розроблення способу. Таким чином, непродуктивна складова часу реконфігурації обчислювального середовища визначається часом, витраченим на передавання конфігураційних даних, для зменшення якого пропонується комплексне використання технологій попередньої реконфігурації функціональних блоків на поверхні ПЛІС і запобігання повторному завантаженню конфігураційних даних.

Технологія запобігання повторному завантаженню ґрунтується на відомих технологіях повторного використання обчислювальних ресурсів ФБ. Принцип зменшення загального часу обчислення за рахунок повторного використання ресурсів ФБ апаратних задач описано в роботі [44].

Нехай обчислювальна задача $Task_1'$ з параметрами часу $T_1' = T_1' + R_1'$, повторюється в момент часу t_1' (рис. 2.5, *а*). Якщо після виконання апаратної задачі її конфігурація видаляється з поверхні ПЛІС, реконфігуровна ділянка кристала ПЛІС використовується засобами планування і розподілу задач для розміщення нової апаратної задачі. Тоді апаратна задача T_1' має бути повторно розміщена на вільне місце реконфігуровної ділянки кристала ПЛІС. Заштрихована частина R_1' відповідає часу, витраченому на реконфігурацію певної області ПЛІС для виконання апаратної задачі. За технологією повторного використання обчислювальних ресурсів функціональних блоків апаратна задача $Task_1$ після виконання не видаляється з поверхні реконфігуровної ділянки ПЛІС і у разі потреби використовується повторно для виконання однотипних завдань. Апаратна задача $Task_1'$ запускається в момент часу t_1' в уже сконфігурованому функціональному блоці (рис. 2.5, *б*). При цьому вилучаються витрати на повторне завантаження конфігураційних даних ($R_1' = 0$), що зменшує час виконання обчислювального алгоритму T_{TOTAL} .



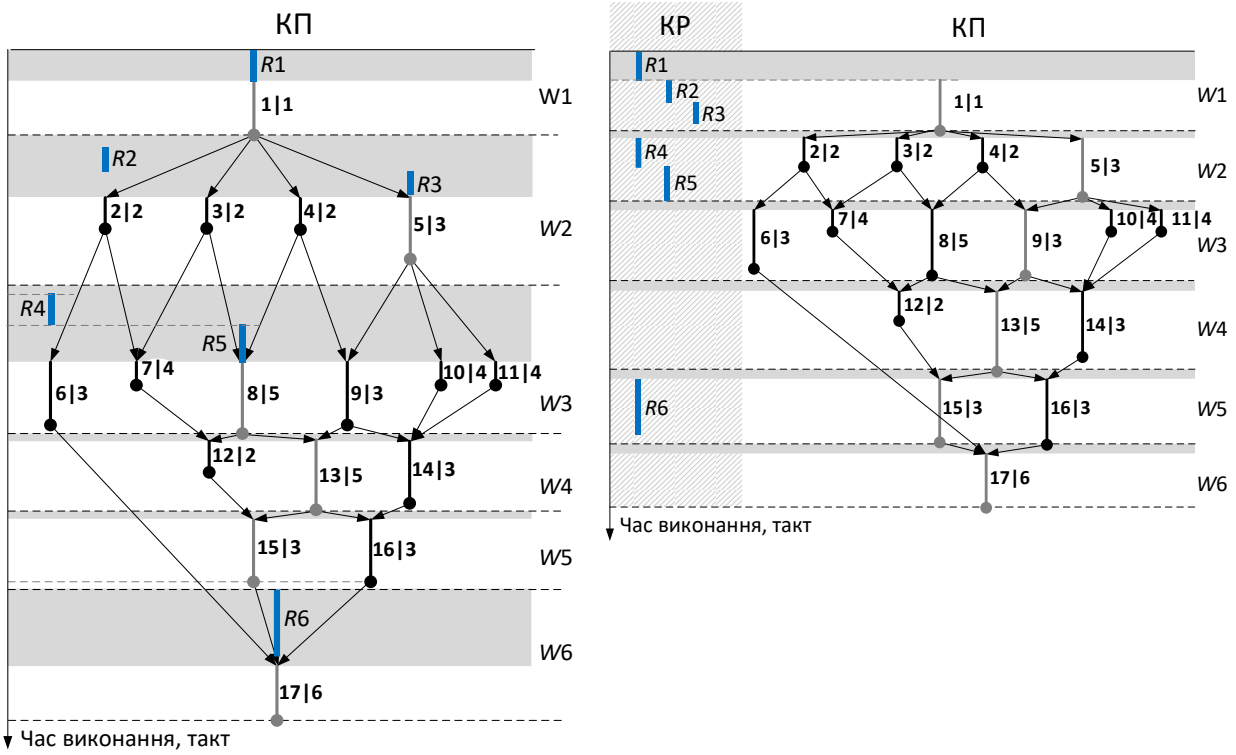
а *б*

Рис. 2.5. Приклад розподілення задач на реконфігуровне обчислювальне середовище: *а* – стандартний процес реконфігурації, *б* – повторне використання обчислювальних ресурсів [44]

У праці [158] з’ясовано, що найбільший ефект від попередньої реконфігурації досягається за рахунок розпаралелення процесів керування обчисленнями і керування реконфігурацією обчислювального середовища, для чого запропоновано застосування контролера реконфігурації і контролера прямого доступу до пам’яті, які забезпечують децентралізацію процесів пошуку і передавання конфігураційних даних та прошивання мікросхеми ПЛІС. На базі такого розпаралелення комплексне застосування двох технологій приводить до того, що частина непродуктивного часу передавання конфігураційних даних буде видалена з критичного шляху через повторне використання ресурсів ФБ, а частина – переміщена на попередні рівні ЯПФ графу за рахунок попередньої реконфігурації. У результаті такої модифікації критичний шлях графу алгоритму визначається тільки продуктивним часом виконання завдань на кожному ярусі ЯПФ ГА.

Приклад застосування запропонованого способу скорочення критичного часу виконання задач показано на рис. 2.6. Часову діаграму стандартного процесу реконфігурації обчислювального середовища зображено на рис. 2.4. Технологія повторного використання обчислювальних ресурсів

функціональних блоків сприяє значному скороченню непродуктивного часу (рис. 2.6, *а*), а завчасна реконфігурація майже повністю видаляє непродуктивний час з критичного шляху ЯПФ ГА (рис. 2.6, *б*).



а *б*

Рис. 2.6. Приклад застосування комплексного способу скорочення критичного часу виконання обчислювального алгоритму: *а* –повторне завантаження ресурсів ФБ; *б* – завчасна реконфігурація; ***i*|*j*** – номер вершини N_i | тип операції I_j; – час реконфігурації R_j; – час виконання завдання T_j; – час виконання завдання на критичному шляху T_j; – непродуктивний час обчислення; КП – керувальний процесор; КР – контролер реконфігурації

Абстрактна оцінка часу виконання обчислювальних задач, поданих ЯПФ графів (рис. 2.4, рис. 2.6) за використання комплексного способу скорочення критичного часу обчислення наведена в табл. 2.2. На підставі абстрактної оцінки часу (табл. 2.2) видно, що шляхом комплексного

скорочення критичного часу виконання обчислювальної задачі непродуктивний час значно зменшується і вимірюються часом синхронізації процесів реконфігурації та обчислення і часом реконфігурації обчислювального середовища для виконання задач першого ярусу ЯПФ ГА.

Таблиця 2.2

Порівняльна оцінка часу виконання алгоритму обчислювальної задачі

Номер ярусу	Стандартна послідовність реконфігурації	Повторне використання ресурсів ФБ	Завчасна реконфігурація з повторним використанням ресурсів ФБ
W_1	$T_{W_1} = R_1 + T_1$	$T_{W_1} = R_1 + T_1$	$T_{W_1} = R_1 + T_1$
W_2	$T_{W_2} = 3R_2 + R_3 + T_5$	$T_{W_2} = R_2 + R_3 + T_5$	$T_{W_2} = R_4 + R_5$
W_3	$T_{W_3} = 2R_3 + 3R_4 + T_8$	$T_{W_3} = R_3 + R_4 + T_8$	$T_{W_3} = T_8$
W_4	$T_{W_4} = R_2 + R_5 + R_3 + T_{13}$	$T_{W_4} = T_{12}$	$T_{W_4} = T_{12}$
W_5	$T_{W_5} = 2R_3 + T_{15}$	$T_{W_5} = T_{14}$	$T_{W_5} = R_6$
W_6	$T_{W_6} = R_6 + T_{17}$	$T_{W_6} = R_6 + T_{17}$	$T_{W_6} = T_{17}$

2.4.2. Спосіб багаторівневого кешування конфігураційних даних в динамічно реконфігуровних комп’ютерних системах. Традиційний підхід до пришвидшення процесу реконфігурації обчислювального середовища, описаний у працях [52, 53, 58, 68], ґрунтується на кешуванні прошивань в зовнішній пам’яті або у внутрішній пам’яті мікросхеми ПЛІС. Перший спосіб кешування не використовує внутрішню пам’ять кристала, але характеризується комунікаційними затриманнями під час завантаження бітових потоків конфігураційних даних через зовнішні інтерфейси ПЛІС. Другий спосіб дозволяє позбавитись комунікаційних затримань, але використовує критично великі ємності внутрішньої пам’яті ПЛІС. Кешування

тут означає тимчасове зберігання конфігураційних даних у безпосередній близькості до цільового реконфігуровного обчислювального середовища на ПЛІС з метою зменшення часу передавання конфігураційних даних [52].

На відміну від відомих механізмів завчасної реконфігурації, що зазвичай гуртуються на статичній реконфігурації ПЛІС, в межах підвищення ефективності виконання завдання пришвидшення реконфігурації з врахуванням просторових обмежень реконфігуровної ділянки ПЛІС пропонується попереднє кешування конфігураційних даних на декількох рівнях обчислювального модуля РКС.

У працях [68, 159] виконано оцінку використання апаратних ресурсів ПЛІС, за якою визначено, що апаратурні витрати кристалів ПЛІС під час зберігання вже сконфігурованих апаратних задач значно менші, ніж під час зберігання їх конфігураційних даних у внутрішній пам’яті ПЛІС. Це стало підґрунтям для способу багаторівневого кешування конфігураційних даних, що описаний в роботах [158, 159]. Цей спосіб, який розроблений для підвищення ефективності відомих технологій кешування конфігураційних даних [33, 44, 46, 47, 91, 95] в динамічно РКС, полягає в тому, що для тимчасового зберігання конфігураційних даних разом з використанням зовнішньої пам’яті використовується поверхня реконфігуровної області ПЛІС.

У праці [160] розроблено формалізацію способу багаторівневого кешування конфігураційних даних. Для оцінки комунікаційних витрат під час реконфігурації обчислювального середовища, що залежать від місця зберігання конфігураційних даних (див. рис. 1.6), визначено три базові послідовності виконання процесу реконфігурації: стандартна послідовність, коли конфігураційні дані завантажуються з віддаленої централізованої бібліотеки конфігурацій, і дві послідовності з кешуванням конфігураційних даних у локальній пам'яті обчислювального модуля і на поверхні ПЛІС (рис. 2.7).

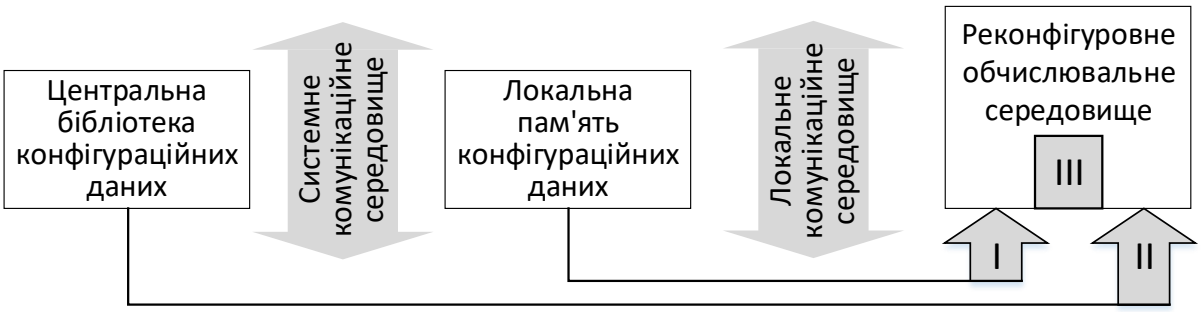


Рис. 2.7. Основні етапи завантаження конфігураційних даних на ПЛІС

Послідовність реконфігурації I. Конфігураційні дані апаратної задачі $j \mid j = \overline{1,m}$, де m – кількість апаратних задач в централізованій бібліотеці конфігурацій, містяться у віддаленій централізованій бібліотеці конфігурацій. Час виконання завдання

$$T_{SUM\,j}^{\text{I}} = R_j^{\text{БК}} + T_{HW\,j} = (T_{COMM\_NET\,j}) + T_j, \qquad (2.17)$$

де $T_{COMM\_NET\,j}$ – час пошуку і передавання конфігураційних даних з бібліотеки конфігурацій на рівень обчислювального модуля мережевими засобами зв’язку та прошивання ПЛІС.

Послідовність реконфігурації II. Конфігураційні дані апаратної задачі j містяться у зовнішній локальній пам’яті конфігурацій (ЛПК) обчислювального модуля. Сумарний час виконання завдання

$$T_{SUM\,j}^{\text{II}} = R_j^{\text{ЛПК}} + T_j = T_{COMM\,j} + T_j, \qquad (2.18)$$

де $T_{COMM\,j}$ – час пошуку і передавання конфігураційних даних з ЛПК та прошивання ПЛІС, який вимірюється пропускною здатністю інтерфейсів введення-виведення ПЛІС та швидкодією ЛПК і локального комунікаційного середовища. При цьому $T_{COMM\_NET\,j} >> T_{COMM\,j}$, з чого випливає, що $R_j^{\text{БК}} >> R_j^{\text{ЛПК}}$.

Послідовність реконфігурації III. Апаратна задача j уже сконфігурована на поверхні реконфігуровної ділянки ПЛІС. Час виконання завдання

$$T_{SUM\ j}^{\text{III}} = T_j, \tag{2.19}$$

де T_j – час виконання апаратної задачі на поверхні реконфігуровної ділянки ПЛІС, включаючи процеси введення вихідних даних та виведення результатів. За цією послідовністю $R_j = 0$.

У праці [160] виконано формальну оцінку критичного часу виконання обчислювальних задач за різних способів кешування конфігураційних даних.

Нехай деяка послідовність B є послідовністю взаємозв'язаних завдань (макрофункцій) $I_j \mid j = \overline{1,K}$, з яких складається критичний шлях графу алгоритму обчислювальної задачі, де K – кількість типів операцій, реалізованих у межах ГА.

Визначимо час виконання послідовності завдань B під час стандартного процесу реконфігурації, ураховуючи вираз (2.17):

$$T_B = \sum_{j=1}^{K} (P_j \cdot T_{SUM\,j}^{I}), \tag{2.20}$$

де P_j – кількість екземплярів кожного типу операції j.

Механізм повторного використання ресурсів функціональних блоків, за якого конфігураційні дані кешуються на поверхні ПЛІС, забезпечує час виконання критичного шляху алгоритму, ураховуючи вираз (2.19):

$$T_B^{FPGA} = \sum_{j=1}^{K} (T_{SUM\ j}^{\text{I}} + (P_j - 1) T_{SUM\ j}^{\text{III}}). \tag{2.21}$$

Прискорення обчислення за кешування на поверхні ПЛІС можна виміряти абсолютним приростом швидкодії ΔT_B і коефіцієнтом прискорення реконфігурації K_R. З урахуванням виразів (2.17), (2.19) перетворимо вирази (2.20) і (2.21) відповідно таким чином:

$$T_B = \sum_{j=1}^{K} (P_j (T_j + R^{\text{I}}{}_j)) = \sum_{j=1}^{K} (P_j T_j + P_j R^{\text{I}}{}_j); \tag{2.22}$$

$$T_B^{FPGA} = \sum_{j=1}^{K} ((T_j + R^{\text{I}}{}_j) + (P_j - 1) T_j) = \sum_{j=1}^{K} (P_j T_j + R^{\text{I}}{}_j). \tag{2.23}$$

Отримаємо значення абсолютного прирощення:

$$\Delta T_B^{\text{БК}} = T_B - T_B^{FPGA} = \sum_{j=1}^{K}(P_j T_j + P_j R^{\text{I}}{}_j) - \sum_{j=1}^{K}(P_j T_j + R^{\text{I}}{}_j) = \sum_{j=1}^{K}(P_j - 1)R^{\text{I}}{}_j .$$

Складова $P_j T_j$ у виразах (2.22) і (2.23) відповідає сумарному продуктивному часу виконання на ПЛІС усіх екземплярів апаратної задачі I_j. Складова $R^{\text{I}}{}_j$ відповідає часу їх одноразового завантаження із централізованої бібліотеки конфігурацій. Отриманий показник прирощення $\Delta T_B^{\text{БК}}$ дозволяє оцінити обсяг зменшення непродуктивного часу обчислення алгоритму з видаленням повторного завантаження конфігурацій, який дорівнює:

$$\Delta T_B^{\text{БК}} = T_{B\_remove}^{\text{БК}}.$$

Застосування швидкодійної локальної пам’яті обчислювального модуля для кешування конфігурацій дозволяє додатково зменшити обсяг непродуктивних витрат. На підставі $R_j^{\text{БК}} >> R_j^{\text{ЛПК}}$ та виразів (2.17) і (2.18) отримаємо:

$$\Delta T_B^{\text{ЛПК}} = \Delta T_{B\_remove}^{\text{ЛПК}} = \sum_{j=1}^{K}(P_j R_j^{\text{I}} - R_j^{\text{II}}),$$

тоді

$$\Delta T_{B\_remove}^{\text{ЛПК}} > \Delta T_{B\_remove}^{\text{БК}}.$$

Математичні моделі функціональних процесів обробки даних, що визначають зменшення критичного часу виконання обчислювальних задач, поданих ЯПФ графів, і обсяг непродуктивного часу, узагальнені в табл. 2.3.

Графічні залежності критичного часу виконання обчислювальних задач від кількості однотипних задач за різних способів кешування конфігураційних даних, отримані на основі формальних визначень пришвидшення реконфігурації (2.24) – (2.27) (табл. 2.3), показано на рис. 2.8.

Із діаграм видно, що час виконання обчислень за кешування апаратних задач на поверхні ПЛІС значно менший ніж за кешування конфігураційних даних у зовнішній локальній пам’яті.

Таблиця 2.3

Математичні моделі базових стратегій організації процесів обробки інформації

Критичний час виконання алгоритму, T^B	Обсяг зменшення непродуктивних витрат, ΔR_{remove}
Стандартна послідовність реконфігурації	
$T^B = \sum_{j=1}^{K} P_j T_{R\,j} + \sum_{j=1}^{K} P_j T_{HW\,j}$ (2.24)	-
Завантаження із центральної бібліотеки конфігурацій	
$T^B_{rapid\_\mathrm{I}} = \sum_{j=1}^{K} T_{R\,j}{}^{\mathrm{I}} + \sum_{j=1}^{K} P_j T_{HW\,j}$ (2.25)	$\Delta R_{remove\_\mathrm{I}} = \sum_{j=1}^{K} R^{\mathrm{I}}{}_j (P_j - 1)$ (2.28)
Завантаження із локальної пам'яті обчислювального модуля	
$T^B_{rapid\_\mathrm{II}} = \sum_{j=1}^{K} R_j{}^{\mathrm{II}} + \sum_{j=1}^{K} P_j T_j$ (2.26)	$\Delta R_{remove\_\mathrm{II}} = \sum_{j=1}^{K} P_j R_j{}^{\mathrm{I}} + \sum_{j=1}^{K} R^{\mathrm{II}}{}_j (P_j - 1)$ (2.29)
Кешування конфігурацій на поверхні ПЛІС	
$T^B_{rapid\_\mathrm{III}} = \sum_{j=1}^{K} P_j T_j$ (2.27)	$\Delta R_{remove\_\mathrm{III}} = \sum_{j=1}^{K} P_j R_j{}^{\mathrm{I}} + \sum_{j=1}^{K} P_j R_j{}^{\mathrm{II}}$ (2.30)

У свою чергу відсутність кешування за стандартної послідовності реконфігурації обумовлює збільшення часу обчислення. На діаграмах видно, що ефективність застосування повторного використання ресурсів апаратних задач збільшується зі збільшенням кількості однотипних задач. Стандартний процес реконфігурації характеризується значними затримками і не залежить від кількості повторень однотипних задач в обчислювальному алгоритмі.

Надалі визначимо коефіцієнт пришвидшення реконфігурації K_R. На підставі виразів (2.22) і (2.23) отримаємо:

$$K_R = \frac{T_B}{T_B^{FPGA}} = \frac{\sum_{j=1}^{K} (P_j T_j + P_j R_j^{\mathrm{I}})}{\sum_{j=1}^{K} (P_j T_j + R_j^{\mathrm{I}})} = \frac{\sum_{j=1}^{K} (T_j + R_j^{\mathrm{I}})}{\sum_{j=1}^{K} (T_j + \frac{R_j^{\mathrm{I}}}{P_j})}.$$

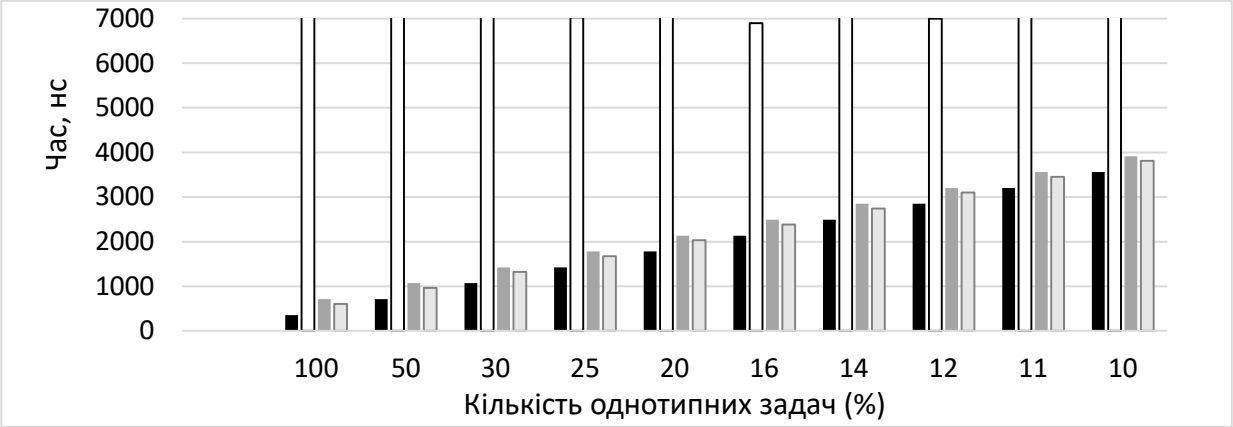


Рис. 2.8. Залежність критичного часу виконання обчислювального алгоритму від кількості однотипних задач: ■ – збереження на поверхні ПЛІС (Послідовність III); □ – завантаження з центральної бібліотеки без повторного використання (Стандартна); ■ – завантаження з центральної бібліотеки з повторним використанням (Послідовність I); ■ – завантаження з локальної пам’яті з повторним використанням (Послідовність II)

Залежності коефіцієнта прискорення реконфігурації від інтенсивності повторення однотипних задач для моделей (2.26) і (2.27) подано на рис. 2.9. Коефіцієнт пришвидшення зростає зі збільшенням кількості екземплярів задач кожного набору; при цьому ефективність використання запропонованого методу залежить від частоти надходження нових типів задач, яка розраховується за виразом

$$P = \frac{K}{w} \cdot 100,$$

де K – кількість нових завдань в ГА; w – довжина критичного шляху ГА.

Час виконання критичного шляху ГА визначається таким виразом:

$$T_B = \sum_{j=1}^{K} R_j + \sum_{j=1}^{K} (P_j T_j). \qquad (2.31)$$

Отриманий час складається із суми часу одноразового прошивання апаратної задачі кожного типу та суми часу виконання всіх задач обчислювального алгоритму. При цьому складова сумарного часу реконфігурації визначає непродуктивний час і може бути зменшена за

допомогою кешування даних на різних рівнях пам’яті згідно із запропонованим у роботі способом (див. табл. 2.3).

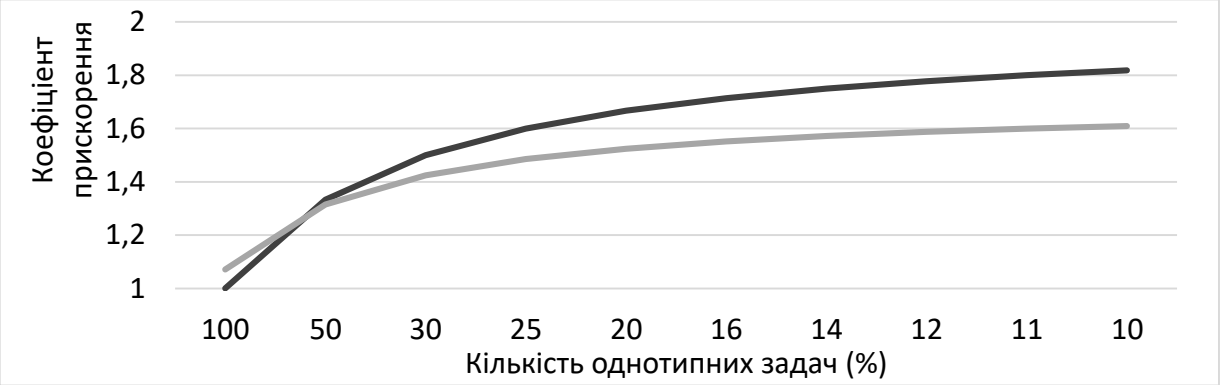


Рис. 2.9. Залежність коефіцієнта прискорення реконфігурації від інтенсивності надходження задач нових типів: ━ – пришвидшення реконфігурації за рахунок зберігання на ПЛІС; ━ – пришвидшення реконфігурації за рахунок завантаження в ЛПК

Для будь-якого паралельного обчислювального алгоритму A критичний час виконання визначається так:

$$T^{\|A} = \sum_{j=1}^{K} R_j + \frac{\sum_{j=1}^{K} (P_j T_j)}{F}, \qquad (2.32)$$

де складова часу обчислення розгалужується на F ФБ, а час реконфігурації є послідовною частиною, розгалуження якої є неможливим, зважаючи на послідовні засоби виконання реконфігурації.

Для задач, поданих ЯПФ графів, час обчислення паралельного алгоритму визначається сумарним часом виконання завдань такої послідовності:

$$B = \{T_{\max\_k} \,|\, k = \overline{1, w}\},$$

де $k = \overline{1, w}$ – номер ярусу; w – кількість ярусів обчислювального алгоритму; $T_{\max\_k}$ – найбільший час виконання задачі k-го ярусу.

Тоді критичний час обчислення задач, поданих ЯПФ графів, дорівнює

$$T_B = \sum_{k=1}^{w} \max \{T_h \big| h = \overline{1, H_k}\}, \quad (2.33)$$

де $h = \overline{1, H_k}$ – номер вузла на ярусі k; H_k – кількість вузлів на ярусі k.

На підставі виразів (2.31) – (2.33) отримаємо критичний час виконання обчислювальної задачі, поданої ЯПФ графу, у реконфігуровній комп'ютерній системі:

$$T_{DAG} = \sum_{j=1}^{K} R_j + \sum_{k=1}^{w} \max \{T_h \big| h = \overline{1, H_k}\}. \quad (2.34)$$

Потенційна можливість зменшення часу виконання паралельного алгоритму (2.34) лежить у зменшенні послідовної компоненти часу, що можна реалізувати шляхом розпаралелення процесів обчислення та реконфігурації. Це дозволяє реалізувати завчасну реконфігурацію, відповідно до якої всі процедури реконфігурації переміщуються на попередні яруси ЯПФ ГА. Таке розпаралелення процесів реконфігурації та обчислення призводить до повного знехтування часом реконфігурації і наближає до нуля непродуктивні витрати часу. Винятком є процедури реконфігурації першого ярусу ЯПФ ГА.

Відповідно до викладеного та виразів (2.31) – (2.34) математична модель пришвидшення реконфігурації для ЯПФ графу G_M, визначається як

$$T_{G\_DAG} = \sum_{j=1}^{K} T_{IO\_j} + \sum_{k=1}^{w} \max\left(\sum_{h=1}^{H_k} R_h, \{T_h \big| h = \overline{1, H_k}\}\right), \quad (2.35)$$

де послідовна компонента T_{IO} визначає послідовність процесів, які остаточно не можуть бути розподілені у часі, наприклад, процеси обміну даними під час реконфігурації, що задіюють загальне комунікаційне середовище.

2.4.3. Метод адаптивного відображення задач на реконфігуровне обчислювальне середовище. Скорочення критичного шляху способом перенесення непродуктивних витрат часу може призвести до того, що інший шлях графу алгоритму стане довший ніж критичний. При цьому в пошуках найбільш ефективного розв'язку виникає необхідність послідовного перебирання всіх шляхів ГА. Ця ітераційна задача може бути розв'язана

швидко за певних умов, а може потребувати більшого часу. Ефективніше така задача пошуку критичного шляху розв’язується в статичному режимі. Однак для реалізації динамічних обчислень за наявності певних функціональних обмежень, коли умови відображення задач непередбачувані, таке розв’язання неефективне. Для підвищення ефективності відображення задач у динамічно РКС у праці [161] розроблено новий спосіб скорочення критичного часу виконання задач, поданих ЯПФ графів, та його формалізація. Запропонований адаптивний спосіб ґрунтується на модифікації відомого алгоритму гілок та границь, який застосовується послідовно в межах кожного ярусу ЯПФ графу. Запропонований спосіб скорочення критичного часу виконання обчислювальних задач, на відміну від статичного способу [155], заснований на послідовному аналізі нащадків вершин кожного ярусу ЯПФ графу та реалізації для кожного з них завчасної реконфігурації. Це забезпечує ефективне врахування змінюваних умов відображення завдань на реконфігуровне обчислювальне середовище, що зумовлені станом обчислювальної структури на ПЛІС у певні моменти обчислювального процесу.

У праці [158] розроблено й обґрунтовано метод адаптивного відображення задач на динамічно реконфігуровне обчислювальне середовище, який базується на запропонованому способі скорочення критичного часу виконання обчислювальних задач [161], пришвидшенні відображення (2.35) [160], інтегрованому підході до скорочення непродуктивного часу [157], багаторівневому кешуванні конфігураційних даних [159]. Математичну модель визначення часу адаптивного відображення обчислювальних задач на динамічно реконфігуровне обчислювальне середовище, що розроблена у праці [158], подано таким виразом:

$$T_{G\_DAG} = \sum_{j=1}^{K} T_{IO\_j} + \sum_{h=1}^{H_1} R_h + \sum_{k=1}^{w} \max\Big(\sum_{h=1}^{H_{(k+1)}} R_h, \{T_h \,\big|\, h = \overline{1, H_k}\}\Big), \qquad (2.36)$$

де $k = \overline{1, w}$ – номер ярусу; w – кількість ярусів обчислювального алгоритму; $v = \overline{1, H_k}$ – номер вузла на ярусі k; H_k – кількість вузлів на ярусі k.

Час виконання обчислювальної задачі, що подана ЯПФ графу, характеризується мінімальним непродуктивним часом, який визначається часом налаштування обчислювального середовища для виконання завдань першого ярусу ГА та витрат на синхронізацію процесів обчислення і реконфігурації. Складова T_{IO} визначає час виконання процесів, які не можуть бути розподілені у часі, наприклад, процеси передавання даних для реконфігурації обчислювального середовища, що виконуються засобами загального комунікаційного середовища.

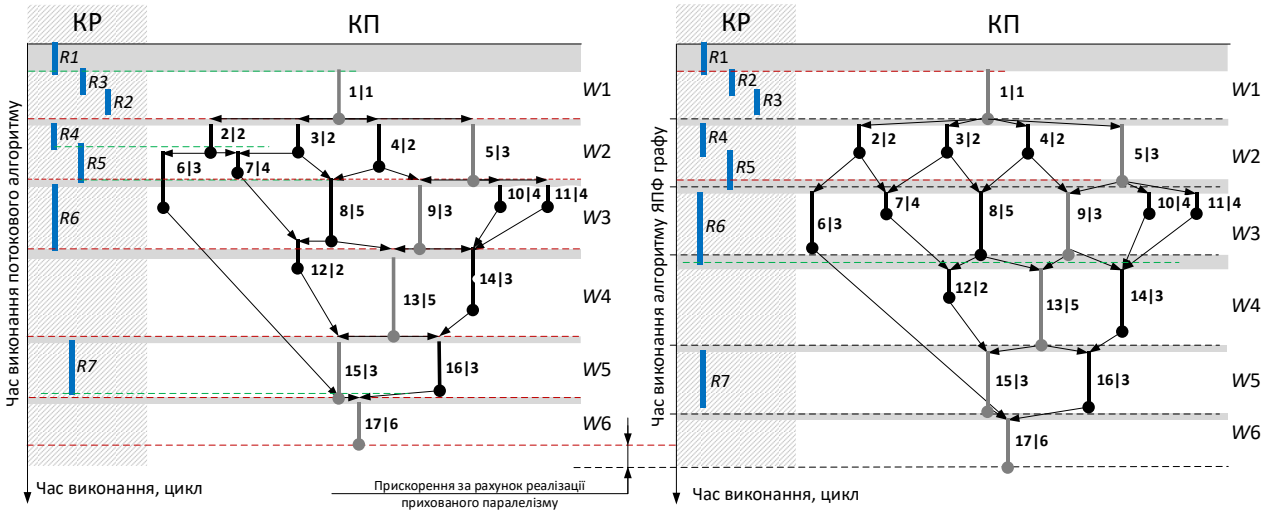
У праці [162] на базі адаптивного способу скорочення критичного часу запропоновано алгоритм адаптивного планування розкладу виконання завдань і розподілу їх на реконфігуровне обчислювальне середовище.

2.5. Математичні моделі адаптивного відображення задач на обчислювальне середовище на ПЛІС у реконфігуровних комп'ютерних системах, керованих потоком даних

У праці [163] запропоновано використання комплексного способу скорочення критичного часу виконання обчислювальних задач у реконфігуровних комп'ютерних системах за реалізації моделі обчислень, керованих потоком даних. Приклад адаптивного скорочення критичного часу виконання ГА G_M представлений на рис. 2.10, *а*. На часовій діаграмі (рис. 2.10, *а*) видно скорочення загального часу обчислення за рахунок реалізації прихованого паралелізму порівняно з традиційними технологіями розпаралелення, коли графи обчислювальних задач подаються в ярусно-паралельній форми (рис. 2.10, *б*).

Для реалізації адаптивного відображення потокових алгоритмів обчислювальних задач на базі комплексного скорочення критичного часу виконання виникає потреба у визначенні критичного шляху графу потокового алгоритму. При статичному аналізі графів можливе використання способів, заснованих на направленому аналізі часу виконання всіх нащадків вершин ГА.

Однак у динамічному режимі використання механізмів направленого перебору може призводити до невизначених часових витрат за непостійних умов відображення. Це пояснюється тим, що скорочення часу виконання завдань на критичному шляху за рахунок завчасної реконфігурації може приводити до того, що час виконання послідовностей задач на інших шляхах графу ставатиме більше ніж час виконання завдань на критичному шляху.



а *б*

Рис. 2.10. Цифрова діаграма адаптивного скорочення критичного часу виконання: *а* – потокового алгоритму; *б* – ЯПФ графу алгоритму; ***i*|*j*** – номер вершини N_i | тип операції I_j; ▮ – час реконфігурації R_j; 🌡 – час виконання завдання T_j; 🌡 – час виконання завдання на критичному шляху T_j; ▒ – непродуктивний час обчислення; КП – керувальний процесор; КР – контролер реконфігурації

У праці [164] запропоновано модифікацію адаптивного методу скорочення критичного часу виконання потокових алгоритмів. Для реалізації модифікованого методу запропоновано автоматичну функцію керування відображенням завдань, що базується на визначенні й аналізі нащадків у кожному поточному вузлі потокового ГА згідно з модифікацією методу гілок та границь. У праці [164] розроблено математичну модель реалізації

автоматичної функції керування відображенням завдань на реконфігуровне обчислювальне середовище, яку показано на рис. 2.11.

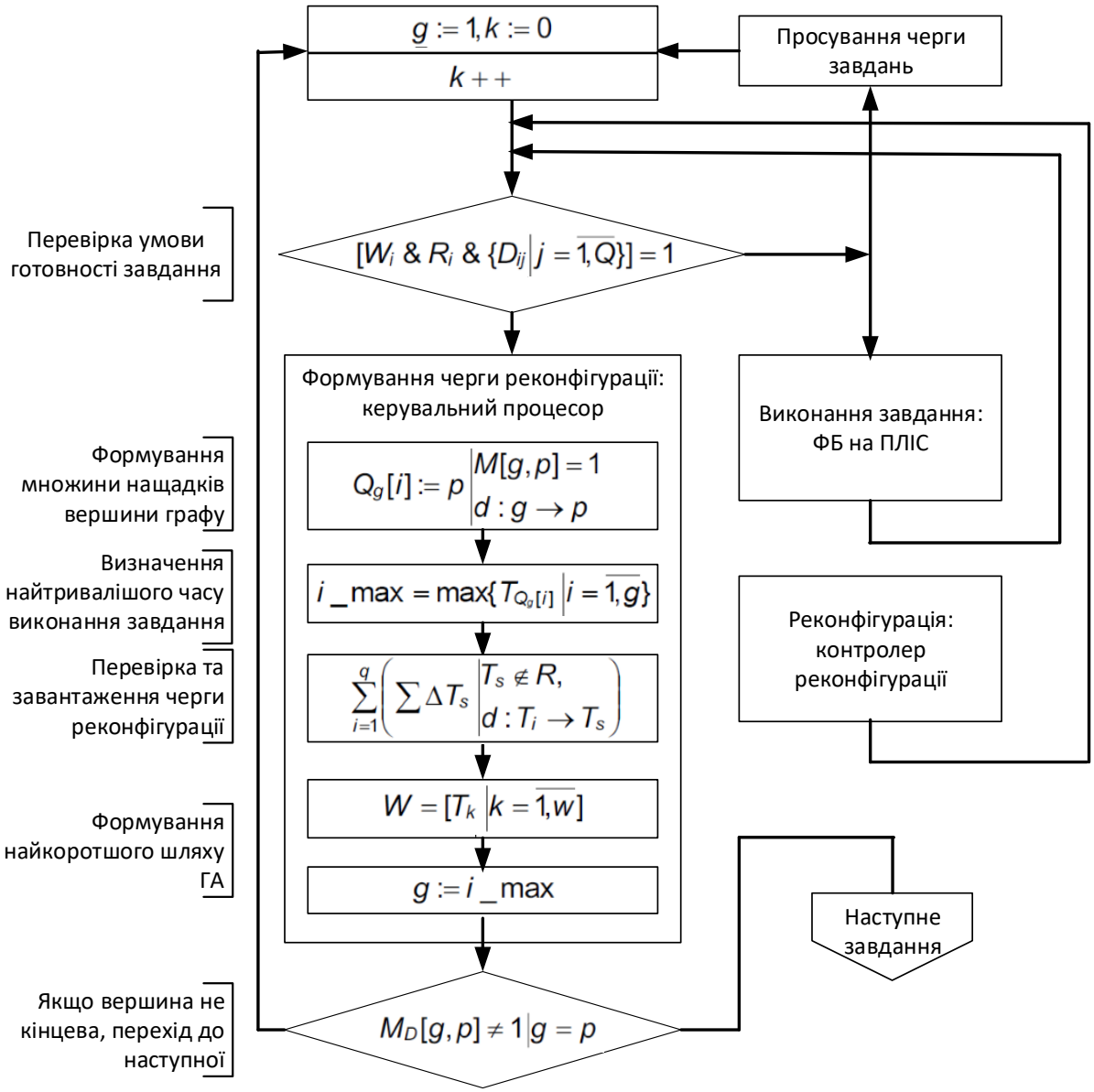


Рис. 2.11. Математична модель автоматичної функції керування адаптивним відображенням завдань на реконфігуровне обчислювальне середовище

Для формалізації автоматичної функції відображення застосовуються такі позначення: T_g і T_i – очікуваний час виконання g-го і i-го завдання графу G_M обчислювальної задачі; $g = \overline{1,v}$, де v – кількість вершин графу G_M; $i = \overline{1,q}$, де q – кількість вершин-нащадків батьківської вершини T_g; $i\_\max$ – індекс

задачі, що має максимальний час виконання серед множини вершин-нащадків $Q_g = \{T_i \mid i = \overline{1,q}\}$; T_D – час виконання обчислювального алгоритму, розрахований за шуканим маршрутом.

Граф алгоритму описується у вигляді матриці зв'язності M_D розмірністю $v \times v$. Рядки $M_D[g] \mid g = \overline{1,v}$ відповідають виконуваним задачам. У стовпцях $M_D[s] \mid s = \overline{1,v}$ кожного рядка $M_D[g]$ розміщується одиниця, якщо задача T_g готує дані для задачі T_s. Отримати множину всіх батьківських задач, які готують дані для задачі T_s, можна проаналізувавши елементи стовпця $M_D[s]$, у якому індекси g вказують на шукані батьківські задачі T_g, якщо $M_D[g,s] = 1$.

Для графу, показаного на рис. 2.10, *а* наведено приклад матриці зв'язності для пошуку нащадків вершин графу алгоритму (рис. 2.12).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Рис. 2.12. Матриця зв'язності

Процес відображення відбувається за такими узагальненими етапами, де N_x готове до виконання завдання в потоковому алгоритмі G_M, що ініціює функцію керування відображенням:

Етап 1. Початкові встановлення: $g := x, k := 0$.

Етап 2. Приріст змінної k $(k++)$. Аналіз чергового рядку матриці $M_D[g]$ виконується за умови готовності задачі T_g до виконання, що відповідає наявності вхідних даних і налаштування обчислювального середовища для виконання задачі. Формується масив задач-нащадків Q_g, для яких задача T_g готує вхідні дані, на підставі якого оновлюється черга реконфігурації та обирається найтриваліша за часом виконання задача $T_{i\_\max}$. Очікуваний час її виконання додається до загального часу виконання алгоритму: $T_D := T_D + T_{i\_\max}$.

Етап 3. Формування послідовності задач адаптивного маршруту: $W[k] := i\_\max$.

Етап 4. Якщо $M_D[g]$ не кінцева вершина графу: $M_D[g,s] \neq 1 | \, g = s$, повернення на етап 1.

Для автоматичної функції керування відображенням (див. рис. 2.11) розроблено узагальнений вираз для визначення часу адаптивного відображення обчислювальних задач на реконфігуровне обчислювальне середовище в реконфігуровних комп'ютерних системах, що керуються потоком даних:

$$T_D := R_{N_1} + T_{BUS} + \left[T_D + \sum_{k=1}^{w} \max \left[\left\{ \begin{matrix} Q_g[i] \middle| i = \overline{1,g}, \\ d : T_g \to T_i \end{matrix} \right\}, \sum_{i=1}^{g} \left(\sum \Delta T_s \middle| \begin{matrix} T_s \notin R, \\ d : T_i \to T_s \end{matrix} \right) \right] \right]. \quad (2.37)$$

Час обчислення потокового алгоритму включає мінімальний непродуктивний час, який визначається часом налаштування обчислювального середовища для виконання завдань першого ярусу R_{N1}. Складова T_{BUS} визначає часові витрати на синхронізацію процесів обчислення і реконфігурації, а також на виконання процесів, які не можуть бути розподілені в часі, наприклад, процесів передавання даних для реконфігурації обчислювального середовища, що виконуються засобами загального комунікаційного середовища.

ВИСНОВКИ ДО РОЗДІЛУ 2

1. У дисертаційній роботі визначено, що найбільш перспективними класами задач для розв’язання в динамічно РКС є задачі керування в реальному часі, зокрема задачі керування в невизначеному базисі, що мають інформаційний, багатовимірний та динамічний характер.

2. З’ясовано, що застосування теорії нечітких множин дозволяє виконувати всі операції з нечіткими змінними на рівні формальних моделей матричного обчислення. На підставі цього визначено основні методи теорії нечітких множин для розв’язання прикладних задач, які зводяться до розв’язання систем лінійних алгебричних рівнянь великої розмірності та виконання операцій з багатовимірними матрицями. Розв’язання таких задач зводиться до реалізації дрібнозернистих алгоритмів.

3. Застосування методів подання певних сукупностей дрібнозернистих операцій як макрофункцій та подальше їх програмне або апаратне пришвидшення створюють передумови для підвищення ефективності розв’язання прикладних задач керування великої та несталої розмірності в реальному часі в динамічно реконфігуровних комп’ютерних системах.

4. Застосування визначених у цьому розділі методів вирішення завдань керування приводить до реалізації алгоритмів зі змішаним типом паралелізму, що дозволяє трансформувати графи алгоритмів обчислювальних задач для налаштування на реконфігуровному обчислювальному середовищі найбільш ефективних обчислювальних структур з високою швидкістю обробки даних.

5. Визначено основні критерії ефективності функціонування динамічно РКС і критерії ефективності реалізації алгоритмів зі змішаним типом паралелізму в динамічно РКС. Удосконалено показник пришвидшення обчислень, який дозволяє розширити визначені критерії ефективності динамічно РКС урахуванням часової складності процесів керування плануванням обчислень і виділенням реконфігуровних обчислювальних ресурсів.

6. Запропоновано новий спосіб багаторівневого кешування конфігураційних даних для динамічно РКС, який дозволяє зменшити непродуктивний час процесу відображення задач на реконфігуровне обчислювальне середовище. Багаторівневе кешування є технологічною основою для реалізації різних стратегій скорочення критичного часу виконання обчислювальних задач.

7. Кешування конфігурацій апаратних задач на поверхні динамічної ділянки ПЛІС дозволяє запобігти повторному передаванню конфігураційних даних для налаштування обчислювального середовища і сприяє вирішенню проблеми обмеження ресурсів внутрішньої пам’яті ПЛІС.

8. Розроблено спосіб скорочення критичного часу виконання обчислювальних задач, який базується на модифікації методу гілок та границь. Визначення і аналіз непродуктивного часу виконання вузлів нащадків вершин ГА, а також трансформація ГА здійснюються за ярусами в динамічному режимі відображення в певному порядку і в різні способи, що визначаються станом реконфігуровного обчислювального середовища, станом функціональних процесів і параметрами виконуваних завдань. Таким чином, цей метод може бути використаний в РКС з динамічно змінюваними умовами відображення задач на реконфігуровне обчислювальне середовище.

9. Запропонований новий метод адаптивного відображення задач на реконфігуровне обчислювальне середовище ґрунтується на інтеграції статичного і динамічного підходів до скорочення непродуктивного часу та багаторівневому кешуванні конфігураційних даних. Застосування методу дозволяє реалізовувати різні стратегії обслуговування завдань із забезпеченням удосконаленого критерію ефективності динамічно РКС на базі доступних апаратних ресурсів ПЛІС.

10. Розроблено математичні моделі функціональних процесів обробки інформації, які дозволили теоретично оцінити ефективність запропонованих методів та засобів адаптивного відображення задач на реконфігуровне обчислювальне середовище. Розроблені математичні моделі на відміну від

відомих формалізують процеси зменшення непродуктивних витрат часу в процесі динамічного відображення задач на реконфігуровне обчислювальне середовище РКС.

11. Реалізація моделі обчислень, керованих потоком даних, створює передумови для підвищення ефективності процесу відображення задач шляхом автоматизації процесу паралельної обробки інформації на локальному рівні обчислювального модуля динамічно реконфігуровних комп'ютерних систем, а також для підвищення їх надійності та розширення функціональних можливостей.

12. Для автоматизації адаптивного відображення завдань у динамічно РКС запропоновано модифікацію методу адаптивного відображення задач на реконфігуровне обчислювальне середовище для моделі обчислень, керованих потоком даних, через реалізацію автоматичної функції керування в кожному вузлі графу алгоритму, яка гуртується на застосуванні методу гілок та границь.

РОЗДІЛ 3
ОПТИМІЗАЦІЯ ПРОЦЕСУ ОБРОБКИ ІНФОРМАЦІЇ У РЕКОНФІГУРОВНИХ КОМП'ЮТЕРНИХ СИСТЕМАХ НА БАЗІ ПЛІС

3.1. Метод оптимізації процесу відображення задач на реконфігуровне обчислювальне середовище

3.1.1. Визначення критеріїв оптимізації. Час відображення обчислювальних задач на реконфігуровне обчислювальне середовище РКС згідно з критеріями ефективності (2.12), (2.13) залежить від накладних витрат процесу реконфігурації обчислювального середовища. У працях [157, 159, 160, 162] запропоновано засоби адаптивного відображення завдань на реконфігуровному обчислювальному середовищі для динамічно РКС, застосування яких зумовлює інтенсивне зменшення непродуктивних комунікаційних витрат $\min(T_{COMM})$ процесу реконфігурації згідно з виразами (2.37) і (2.38). Запропоновані засоби дозволяють відобразити алгоритм обчислювальної задачі якомога ефективніше *Best Effort* на доступні апаратурні ресурси за допомогою динамічної адаптації процесу відображення до стану реконфігуровного обчислювального середовища. Критерієм ефективності в даному випадку є час відображення задач. Такі способи потребують певної оптимізації для підвищення ефективності їх застосування під час розв'язання задач великої розмірності в обмеженому режимі часу. У працях [157, 162, 165] показано, що час обчислення задач інтенсивно зменшується зі збільшенням кількості однотипних функцій в обчислювальному алгоритмі. При цьому засоби адаптивного відображення дозволяють видалити майже всі накладні витрати процесу реконфігурації обчислювального середовища, якщо його розміри відповідають ступеню розпаралелювання алгоритму обчислювальної задачі. Це означає, що ширина графу алгоритму (кількість завдань на ярусі) менша або дорівнює розмірності обчислювального поля ПЛІС (кількості розміщуваних ФБ). Інакше в процесі відображення завдань на реконфігуровне

обчислювальне середовище виникають додаткові непродуктивні витрати часу, що супроводжують процес подолання просторових обмежень обчислювального середовища на ПЛІС.

У математичних моделях (2.36) і (2.37) адаптивного відображення завдань на реконфігуровне обчислювальне середовище додаткові непродуктивні витрати часу враховані так:

$$T_{G\_DAG} = \sum_{j=1}^{K} T_{IO\_j} + \sum_{h=1}^{H_1} R_h + \sum_{k=1}^{w} \max\Big(\sum_{h=1}^{H_{(k+1)}} R_h, \{T_h \,|\, h = \overline{1, H_k}\}\Big) + \Delta R; \qquad (3.1)$$

$$T_{DAG} := R_{N_1} + T_{BUS} + \left[T_D + \sum_{k=1}^{w} \max\left[\left\{ Q_g[i] \,|\, i = \overline{1, g}, \right\}, \sum_{i=1}^{g} \left(\sum \Delta T_s \,\middle|\, \begin{matrix} T_s \notin R, \\ d : T_i \to T_s \end{matrix} \right) \right] \right] + \Delta R, \qquad (3.2)$$

$$\left\{ d : T_g \to T_i \right\}$$

де ΔR – додаткові непродуктивні витрати часу.

На підставі теоретичного аналізу математичних моделей функціональних процесів у РКС (2.24), (2.25), (2.26), (2.27) і (2.36), а також результатів імітаційного моделювання, що виконано у працях [157, 160, 162, 165], визначено основну тенденцію залежності часу відображення від впливу апаратурних обмежень реконфігуровного обчислювального середовища, яку зображено на рис. 3.1.

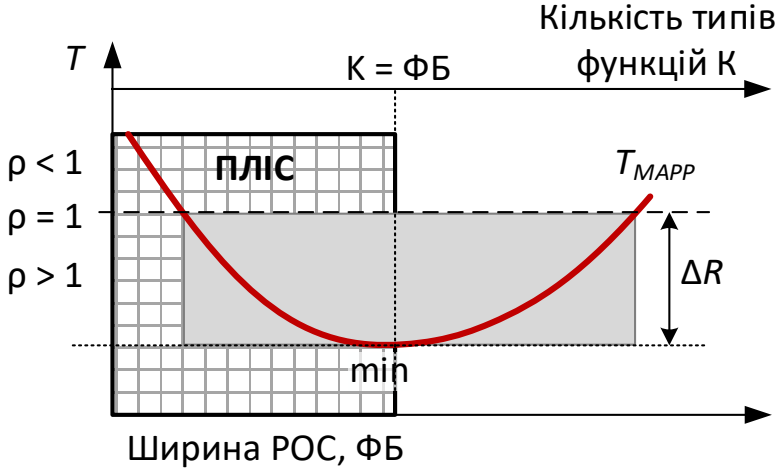


Рис. 3.1. Визначення критеріїв оптимізації процесу відображення задач на реконфігуровне обчислювальне середовище (РОС)

Із діаграми на рис. 3.1 видно, що подолання просторових обмежень реконфігуровної ділянки ПЛІС вносить додаткові затримки, які збільшують час відображення T_{MAPP} .

Таким чином, цільову функцію зменшення непродуктивного часу процесу відображення завдань з урахуванням виразу (2.13) можна подати таким виразом:

$$O(\Delta R) = \min(T_{MAPP}).$$

Для забезпечення мінімальних значень часу відображення визначимо критерії оптимізації процесу адаптивного відображення задач у динамічно РКС, які базуються на таких співвідношеннях:

– параметри ГА обчислювальної задачі відповідають параметрам реконфігуровної ділянки ПЛІС, тобто на поверхні реконфігуровного обчислювального середовища є достатньо вільного місця для розміщення ФБ всіх завдань, шо потребують виконання;

– непродуктивні витрати часу, що супроводжують процес пришвидшення реконфігурації обчислювального середовища, у тому числі з врахуванням першого співвідношення, не перевищують часу ΔR;

– область оптимальних параметрів часу відображення T_{MAPP} обмежена додатними значеннями показника пришвидшення продуктивності $\rho > 1$, відповідно до виразу (2.12). У випадках, коли на час виконання обчислювального процесу накладаються часові обмеження T_{QoS} з боку розв’язуваних додатків ($T_{COUNT} \le T_{QoS}$), область оптимальних параметрів часу відображення обмежується часом T_{QoS} .

У праці [166] узагальнено основні тенденції залежності показників ефективності обробки інформації в РКС від параметрів обчислювальних алгоритмів і параметрів реконфігуровного обчислювального середовища, що зображено діаграмою на рис. 3.2. Критеріями ефективності обробки інформації в РКС є час відображення, пропорційний непродуктивним витратам часу, і час обчислення задачі на апаратурі, що відповідає виразам

(2.11) і (2.12). Часова складність процесу відображення і час програмування ПЛІС (2.12) як константи умовно не враховуються в процесі оптимізації.

Тоді цільова функція зменшення часу виконання обчислень на локальному рівні ОМ РКС набуває такого вигляду:

$$\min(T_{COUNT}) = \min(T_{MAPP} + T_{HW}) \Rightarrow \mathrm{O}(T_{MAPP}, T_{HW}) = \min(T_{COUNT}).$$

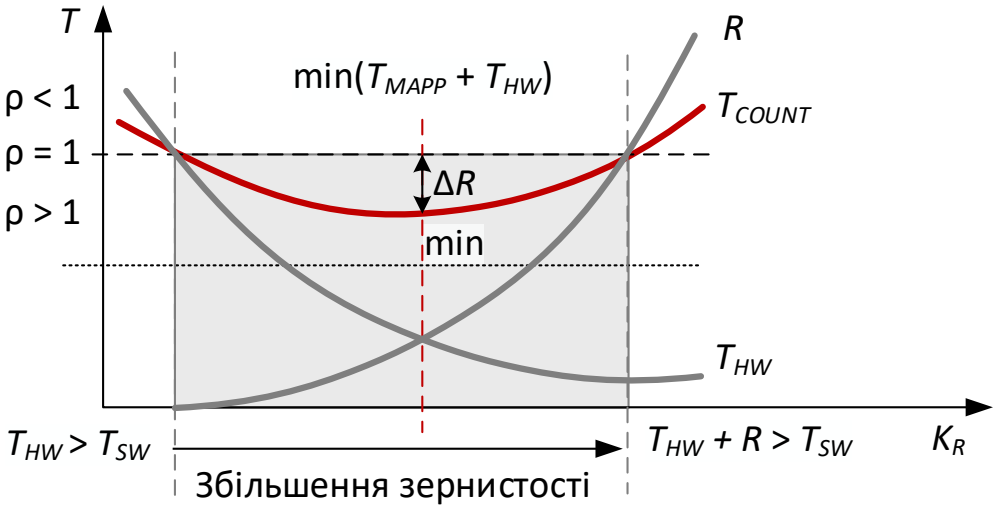


Рис. 3.2. Визначення оптимальних параметрів процесу обробки інформації в РКС

Для забезпечення мінімальних значень часу обробки інформації розширимо критерії оптимізації у динамічно РКС, врахуванням такого співвідношення:

– мінімальні значення часу обробки інформації досягаються в області відповідності ступеня реконфігурації (коефіцієнта реконфігурації) обчислювального середовища і непродуктивних часових витрат, що супроводжують процес реконфігурації обчислювального середовища.

Для визначення кількісного показника співвідношення параметрів алгоритмів обчислювальних задач і параметрів реконфігуровного обчислювального середовища в праці [167] запропоновано використовувати коефіцієнт реконфігурації K_R, який відкладено на горизонтальній осі діаграми на рис. 3.2. Для визначення коефіцієнта реконфігурації запропоновано використання параметра зернистості паралельних обчислень. Коефіцієнт

реконфігурації, значення якого менше або дорівнює нулю, вказує на відсутність реконфігурації. Максимальний коефіцієнт реконфігурації фактично означає, що вся робота виконуватиметься засобами одного функціонального блока, спеціалізованого безпосередньо для виконання певної макрофункції.

Коефіцієнт реконфігурації визначає ступінь перетворення структури обчислювального середовища для виконання певного завдання. З огляду на функціональні й апаратурні обмеження РКС реалізація максимальної зернистості обчислювального середовища супроводжується рядом проблем: значно збільшується непродуктивний час реконфігурації за рахунок збільшення обсягу передавання конфігураційних даних (це зображено на рис. 3.2); для надто великих ФБ може не вистачити місця на ПЛІС; підтримання бібліотеки великої кількості специфічних функцій неефективна у РКС.

На підставі викладеного та основних положень теорії керування зернистістю обчислень [55, 105, 106] для кількісної оцінки коефіцієнта реконфігурації використаємо відомий вираз для визначення зернистості паралельних обчислень [107] у традиційних обчислювальних системах:

$$P_R = Z = \frac{V_{Count}}{V_{IO}}, \tag{3.3}$$

де Z – коефіцієнт зернистості обчислень, V_{Count} – обсяг виконаних операцій для обчислення певної макрофункції у вершині графу обчислювального алгоритму; V_{IO} – кількість операцій пересилання даних (введення-виведення). Тоді для n обчислювальних операцій коефіцієнт реконфігурації буде змінюватись у межах від одиниці до n:

$$P_{R\_\min} = \frac{n}{n},\ P_{R\_\max} = \frac{n}{1},\ P_R = \overline{1,n}. \tag{3.4}$$

Коефіцієнт реконфігурації пов’язаний з поняттям користувацької ефективності E_R обчислювального середовища, що отримано на підставі виразів (3.3) і (3.4):

$$E_R = \frac{P_R}{n} \cdot 100, \tag{3.5}$$

тоді

$$E_{R\_\min} = \frac{1}{n} \cdot 100,\ E_{R\_\max} = \frac{n}{n} \cdot 100 = 100.$$

У праці [167] досліджено характер залежності часу виконання обчислювальних задач від зернистості обчислень на фізичному рівні реконфігуровного обчислювального середовища. Відзначено, що зернистість обчислювального середовища значно впливає на час виконання обчислень у зв'язку з додатковими комунікаційними витратами часу передавання даних. Тому виникає потреба в розширенні розроблених вище критеріїв оптимізації процесу обробки інформації через урахування цільової функції часу виконання завдань на апаратурі ПЛІС:

$$\mathrm{O}(Z_{Task}, Z_{FPGA}) = \min(T_{HW}).$$

Далі визначимо інтегрований критерій оптимізації процесу обробки інформації, який дозволяє розв'язати багатокритеріальну задачу оптимізації процесу обробки інформації на всіх рівнях функціонування РКС шляхом визначення оптимального співвідношення параметрів обчислювальних задач і структури реконфігуровного обчислювального середовища з врахуванням непродуктивного часу і часу апаратного виконання завдань.

Формальне визначення інтегрального критерію оптимізації має такий вигляд:

$$\begin{cases} \mathrm{O}(\Delta R) = \min(T_{MAPP}), \\ \mathrm{O}(T_{MAPP}, T_{HW}) = \min(T_{COUNT}), \\ \mathrm{O}(Z_{Task}, Z_{FPGA}, T_{COMM\_FPGA}) = \min(T_{HW}). \end{cases}$$

Інтегральний критерій оптимізації. Мінімальні значення часу обробки інформації досягаються за таких правил і співвідношень.

1. Область оптимальних параметрів часу відображення T_{MAPP} обмежена додатними значеннями показника пришвидшення продуктивності $\rho > 1$, відповідно до виразу (2.11); у випадках, коли на обчислювальний процес

накладаються часові обмеження з боку розв'язуваних додатків, область оптимальних параметрів часу відображення обмежується затребуваними часовими вимогами T_{QoS}:

$$T_{Mapp} + T_{HW} < T_{SW},\ T_{MAPP} + T_{HW} < T_{QoS}. \quad (3.6)$$

2. Непродуктивні витрати часу, що супроводжують процес пришвидшення реконфігурації обчислювального середовища не перевищують часу ΔR:

$$\Delta R = [\min(T_{Mapp} + T_{HW}); T_{QoS}]. \quad (3.7)$$

3. Оптимальне співвідношення параметрів ГА обчислювальних задач і параметрів реконфігуровної ділянки ПЛІС.

4. Оптимальне співвідношення ступеня реконфігурації (коефіцієнта реконфігурації) обчислювального середовища і непродуктивних часових витрат, що супроводжують процес реконфігурації обчислювального середовища.

5. Оптимальне співвідношення часу виконання апаратних обчислень і непродуктивних часових витрат під час обміну даними.

3.1.2. Формалізація способу визначення непродуктивного часу. Додаткові непродуктивні витрати часу, що супроводжують процес динамічного відображення завдань на реконфігуровному обчислювальному середовищі за невідповідності його структури реконфігуровному обчислювальному середовищу і вимог обчислювальних задач, зазвичай складно оцінити для їх оптимізації.

Подамо вихідну обчислювальну задачу, що є паралельною *M*-задачею зі змішаним типом паралелізму, ЯПФ графу G_M (див. рис. 2.2). Відображення завдань, шо виконуються в вершинах ГА, на реконфігуровну обчислювальну структуру здійснюється за методом адаптивного відображення (2.36), що ґрунтується на аналізі графу за рівнями, коли кожний рівень ЯПФ графу послідовно відображується на структуру обчислювальної системи [157, 159, 162].

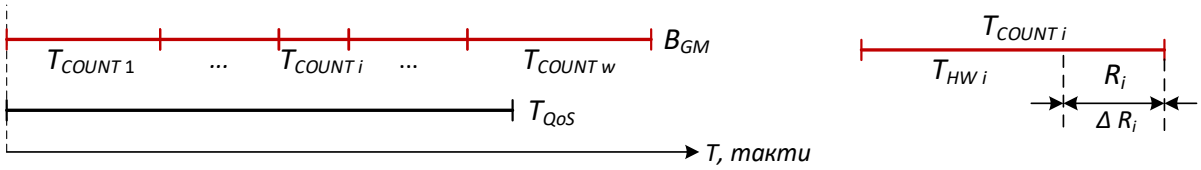
Нехай відповідно до виразу (2.33), що визначає час виконання алгоритму обчислюваної задачі, поданого ЯПФ графу, упорядкована множина завдань:

$$B_{GM} = \{T_{SUM\_\max k} \,|\, k = \overline{1, w}\}, \qquad (3.8)$$

де $T_{SUM\_\max k}$ – час виконання найтривалішого завдання k-го ярусу, $k = \overline{1, w}$ – індекс ярусу; w – кількість ярусів ГА, складає найтривалішу взаємозв'язану послідовність завдань, за якою визначається час виконання вихідної обчислювальної задачі.

На час виконання обчислювальних задач, що розв'язуються в режимі часових обмежень, окрім функціональних часових обмежень реконфігуровних комп'ютерних систем, накладаються додаткові часові обмеження, що встановлюються на підставі певних зовнішніх факторів. Такі обмеження в системах керування реального часу зумовлені тривалістю циклу керування і вимірюються параметром кількості циклів за одиницю часу [51, 115, 168]. Мультимедійні та мережеві додатки, оброблення відео, розпізнавання образів, системи штучного інтелекту часто вимагають від обчислювальних систем певної якості обслуговування *QoS* (*Quality of Service*). Одним з параметрів якості обслуговування в обчислювальних системах є обмеження часу на розв'язання таких задач, що визначається, наприклад, кількістю оброблених кадрів або переданих пакетів даних за одиницю часу. Такі обмеження подаються у циклах виконання обробки даних обчислювальної системи, що приймається як параметр якості обслуговування [54]. Таким чином, будь-яка цільова функція забезпечення часових обмежень зводиться до виконання певного обсягу обчислень за обмежений зовнішніми факторами проміжок часу.

Геометричну інтерпретацію вихідної задачі забезпечення заданого обмеження часу T_{QoS} виконання певної взаємозв'язаної послідовності завдань B_{GM} (3.8), зображено на рис. 3.3.



а *б*

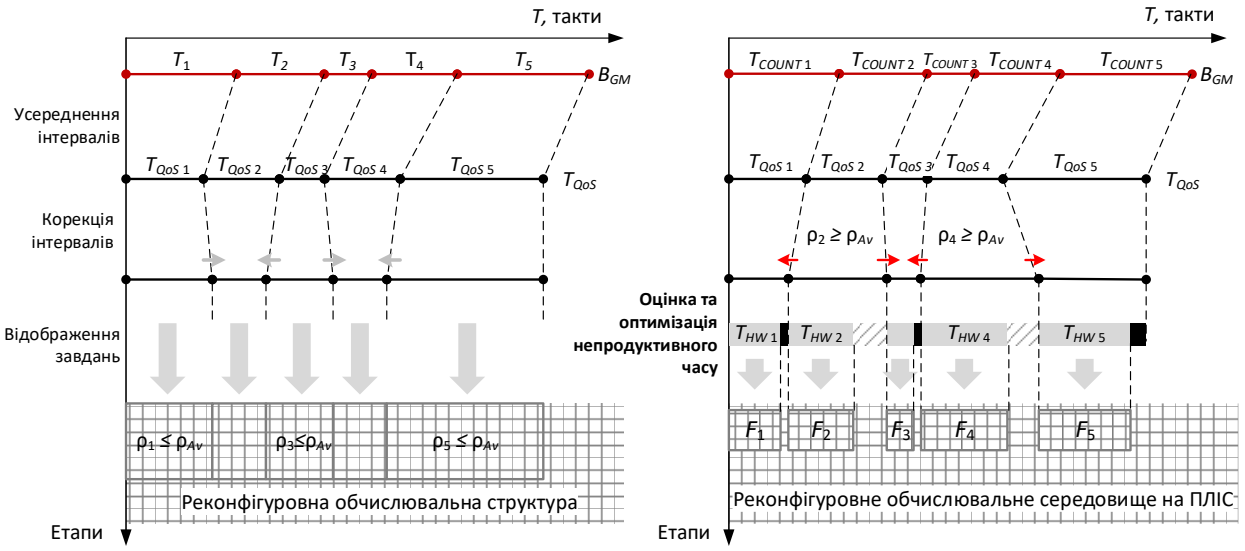
Рис. 3.3. Вихідні дані для розв'язання задачі визначення часових інтервалів виконання завдань: *а* – часові параметри послідовності завдань; *б* – часові параметри завдання

Існує певний інтервал часу ΔR згідно з виразом (3.7), у межах якого час виконання обчислювальної задачі перебуває в межах оптимальних параметрів відображення (див. рис. 3.2, *б*). Параметрами оптимізації відображення є час виконання завдання на апаратурі ПЛІС, час налаштування обчислювального середовища на ПЛІС, частота повторень однотипних завдань, показник пришвидшення обчислення. На всі параметри відображення певним чином впливають просторові обмеження кристалів ПЛІС, подолання яких призводить до додаткових витрат часу відповідно до виразів (3.1) і (3.2). Установлена задача розв'язується шляхом визначення обсягу непродуктивних часових витрат ΔR (рис. 3.3, *б*) і їх оптимізації в межах оптимальних параметрів відображення за критеріями необхідного часу виконання заданої послідовності завдань з урахуванням апаратних обмежень кристалів ПЛІС. Очікуваний час виконання кожного завдання $T_{COUNT\,i}$ (рис. 3.3, *б*) згідно з виразами (2.11) і (2.13) становить:

$$T_{COUNT\,i} = T_{HW\,i} + R_i\,. \qquad (3.9)$$

Для визначення непродуктивного часу запропоновано модифікацію відомого способу забезпечення часових обмежень виконання додатків, який розроблено для мультизадачного режиму обчислень у реконфігуровній обчислювальній системі з комутацією зв'язків [54, 107], де багатоядерні процесори розподіляють між собою регулярне гетерогенне обчислювальне середовище на ПЛІС. При цьому кожне завдання на різних наборах апаратного устаткування реконфігуровної обчислювальної структури виконується з

різною продуктивністю, яка виражається коефіцієнтом пришвидшення обчислень (2.11). Розподіл завдань на динамічно комутоване обчислювальне середовище ґрунтується на визначенні набору устаткування, що забезпечує необхідний інтервал для ефективного виконання кожного завдання в межах виставлених часових вимог, що виражається додатними значеннями показника пришвидшення (2.11), без збиткового використання апаратури. Часову діаграму реалізації способу визначення часових інтервалів ефективного виконання завдань зображено на рис. 3.4, *а*, а часову діаграму виконання модифікованого способу визначення непродуктивного часу – на рис. 3.4, *б*.



а *б*

Рис. 3.4. Приклади часових діаграм основних етапів визначення часових інтервалів: *а* – відомий спосіб; *б* – модифікований спосіб; ▒ – час обчислення на апаратурі ПЛІС; ▨ – виділення додаткового часу для виконання завдання; ■ – непродуктивний час, що потребує оптимізації; ▦ – виділення реконфігуровних обчислювальних ресурсів

Непродуктивний час визначаються у два етапи:

Етап 1. Обмеження часу попередньо розподіляється між завданнями

вихідної послідовності пропорційно очікуваному часу виконання кожного завдання $T_{COUNT\,i}$ (рис. 3.3, *б*). Таким чином, отримані усереднені інтервали часу $T_{QoS\_Av\,i}$ складаються з очікуваного часу виконання апаратної задачі на апаратурі ПЛІС $T_{HW\,i}$ і певної усередненої кількості непродуктивного часу $\Delta R_{Av\,i}$.

Етап 2. Коригуються усереднені інтервали часу. На відміну від відомого способу коригування виконується обернено пропорційно показнику пришвидшення кожного завдання. У результаті корегування час обмеження перерозподіляється між завданнями таким чином, що завдання, пришвидшення яких вище від середнього значення, і так само завдання, пришвидшення яких наближається до середнього значення, отримують додатковий час за рахунок менш ефективних завдань.

Етап 3. Виконується оцінка непродуктивного часу виконання завдань, на підставі якої визначаються завдання, для яких виконується умова

$$T_{COUNT\,i} \le T_{QoS\,i} \tag{3.10}$$

отже

$$R_i \le \Delta R_i, \tag{3.11}$$

тобто такі завдання отримали необхідний час для виконання в межах часових обмежень. Інша частина завдань, для яких

$$T_{COUNT\,i} > T_{QoS\,i}, \tag{3.12}$$

потребує впровадження механізмів інтенсивного зменшення непродуктивних витрат [157, 159, 162], що відповідно до виразу (3.9) виражається функцією мінімізації часу реконфігурації обчислювального середовища

$$F = \min(R_i). \tag{3.13}$$

Це приводить до виконання умови забезпечення часовим обмеженням

$$T_{HW\,i} + \min(R_i) \le T_{QoS\,i}. \tag{3.14}$$

Доведемо справедливість використання способу забезпечення часових обмежень [54] для розв’язання задачі визначення непродуктивного часу у реконфігуровних комп’ютерних системах на ПЛІС.

На підставі викладеного подамо відоме співвідношення [54] у відповідності до постановки завдання дослідження:

$$T_{QoS\,i} = \left(T_{Count\,i} \cdot \frac{T_{QoS}}{\sum_{i=1}^{n} T_{Count\,i}} \right) \cdot \left(\frac{\rho_i}{\sum_{i=1}^{n} \rho_i / n} \right), \tag{3.15}$$

де T_{QoS} – загальні обмеження часу виконання програми; $T_{QoS\,i}$ – обмеження часу, розраховані для кожного i-го завдання, що надходить на виконання, на підставі часових вимог T_{QoS}; $T_{COUNT\,i}$ – очікуваний час виконання i-го завдання; ρ_i – пришвидшення продуктивності, розраховане за формулою (2.11).

Параметр часу програмного виконання функціональних ядер T_{SW} у виразі (2.11), для яких заздалегідь синтезовано бібліотеку апаратних задач, отримують шляхом профілювання програми [54, 170].

Відповідно до виразу (2.11) показник пришвидшення продуктивності i-го завдання визначимо таким чином:

$$\rho = \frac{T_{SW\,i}}{T_{COUNT\,i}}. \tag{3.16}$$

Для доведення справедливості виразу (3.15) та отримання співвідношень між основними параметрами подамо геометричну інтерпретацію визначення часових інтервалів виконання завдань у межах загального обмеження часу (рис. 3.5).

На діаграмі (рис. 3.5), використовуються такі позначення: $\sum_{i=1}^{n} T_{SW\,i}$ – сумарний час обчислення програмними засобами; $T_{TOTAL} = \sum_{i=1}^{n} T_{COUNT\,i}$ – сумарний час обчислення з апаратним пришвидшенням; K_{Av} – коефіцієнт усереднення; $T_{Av\,i}$ і $\sum_{i=1}^{n} T_{SW\,Av\,i}$ – відповідно час обчислення i-ї завдання і сумарний час обчислення, розраховані за усередненими показниками. Лінії I показують розподіл часових інтервалів за очікуваними параметрами часу обчислення завдань; лінії II – за розрахованими усередненими параметрами;

лінії III зображують процес коригування часових інтервалів з урахуванням часових вимог виконання алгоритму обчислювальної задачі.

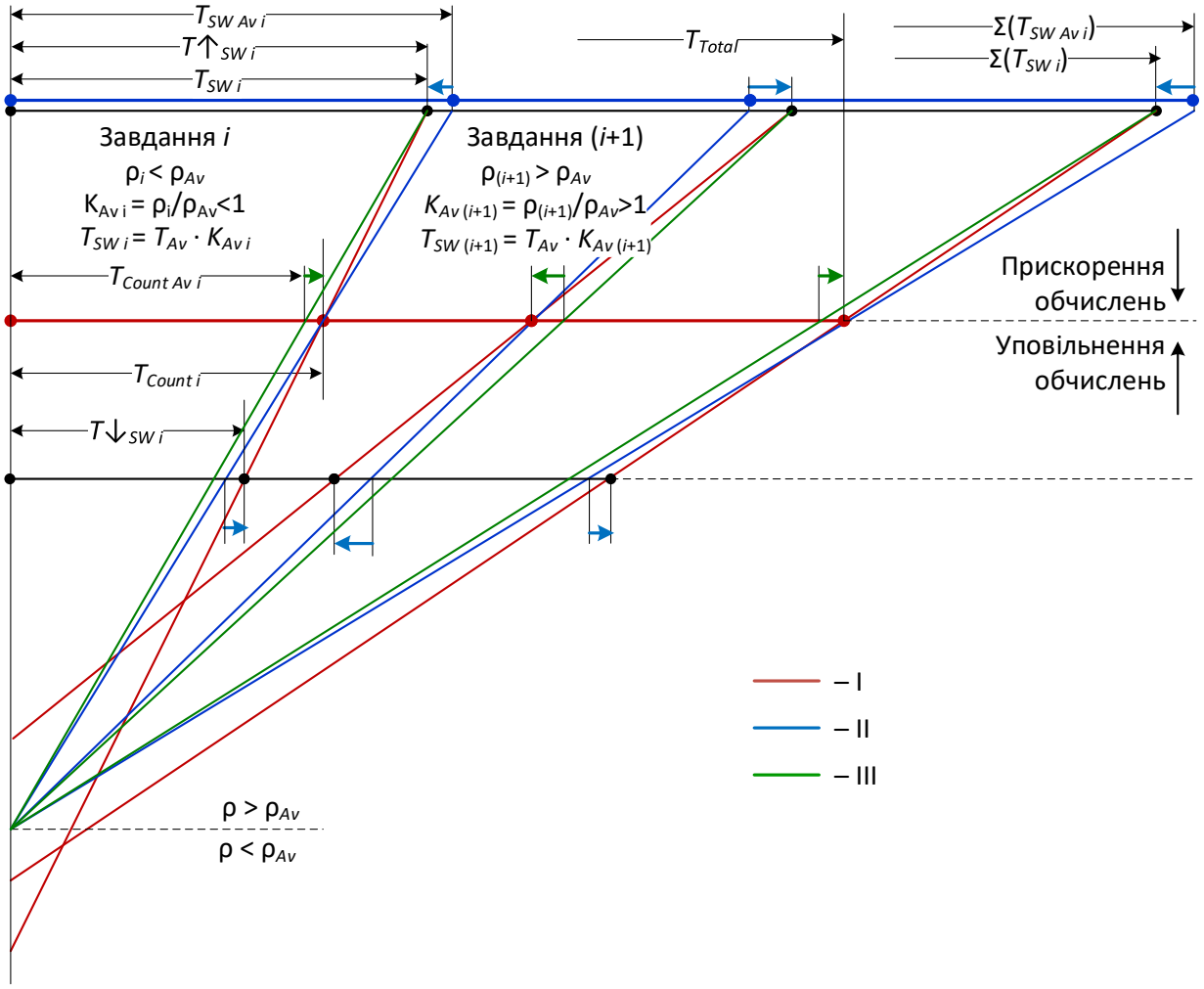


Рис. 3.5. Залежність часу виконання завдань від коефіцієнта апаратного пришвидшення обчислення

Із діаграми на рис. 3.5 видно, що сумарний час обчислення програмними засобами і сумарний час обчислення з апаратним пришвидшення як і всі відповідні проміжки часу, що визначають кожне завдання в послідовності обчислень, пропорційно залежать від середнього коефіцієнта прискорення ρ_{Av}:

$$\frac{\sum_{i=1}^{n} T_{SW\,i}}{\sum_{i=1}^{n} T_{COUNT\,i}} = \frac{T_{SW\,i}}{T_{COUNT\,i}} \Bigg| i = (\overline{1,n}) = \rho_{Av},$$

тоді

$$T_{SW\,i} = T_{COUNT\,i} \cdot \frac{\sum_{i=1}^{n} T_{SW\,i}}{\sum_{i=1}^{n} T_{COUNT\,i}} = T_{COUNT\,i} \cdot \rho_{Av} .$$

Часові проміжки, отримані з використанням середнього коефіцієнта пришвидшення потребують коригування на величину відхилення від середнього значення, яке визначимо як коефіцієнт усереднення. Коефіцієнт усереднення розраховується за формулою

$$K_{Av\,i} = \frac{\rho_i}{\left(\sum_{i=1}^{n} \rho_i\right)/n},$$

де $\left(\sum_{i=1}^{n} \rho_i\right)/n = \rho_{Av}$ – середній коефіцієнт пришвидшення. На діаграмі (рис. 3.5) видно, якщо значення коефіцієнта пришвидшення *i*-го завдання послідовності ρ_i більше від середнього коефіцієнта пришвидшення ρ_{Av} ($K_{Av\,i} > 1$), такий час потребує коригування в бік збільшення, і навпаки, якщо значення коефіцієнта пришвидшення *i*-го завдання послідовності ρ_i менше від середнього коефіцієнта пришвидшення ρ_{Av} ($K_{Av\,i} < 1$), то такий час потребує коригування в бік зменшення.

Таким чином, після коригування для кожного завдання $T_{SW\,i} = T_{Av\,i} \cdot K_{Av\,i}$ отримуємо усереднені значення сумарного часу обчислення без пришвидшення, що дорівнює $T_{Av\,Total} = \sum_{i=1}^{n} T_{SW\,Av\,i}$.

Визначимо тепер залежність часу виконання завдання від його коефіцієнта пришвидшення:

$$T_{SW\,i} = T_{Count\,i} \cdot \frac{\sum_{i=1}^{n} T_{SW\,i}}{\sum_{i=1}^{n} T_{Count\,i}} \cdot \frac{\rho_i}{\left(\sum_{i=1}^{n} \rho_i\right)/n} = T_{Count\,i} \cdot \rho_{Av} \cdot \frac{\rho_i}{\rho_{Av}} = T_{Count\,i} \cdot \rho_i , \qquad (3.17)$$

що з урахуванням виразу (3.16) доводить справедливість виразу (3.15) для зроблених вище вихідних настанов.

Виходячи з діаграми, зображеної на рис. 3.5, запишемо залежності між основними параметрами обчислення:

$$T_{Count\,i} = \frac{T^{\uparrow}_{SW\,i}}{\rho^{\uparrow}_{i}},\ T_{Count\,i} = T^{\downarrow}_{SW\,i} \cdot \rho^{\uparrow}_{i} = \frac{T^{\downarrow}_{SW\,i}}{\rho^{\downarrow}_{i}},\ \text{якщо}\ \rho^{\uparrow}_{i} = \frac{1}{\rho^{\downarrow}_{i}}.$$

Відповідно

$$\rho^{\uparrow}_{i} = \frac{T^{\uparrow}_{SW\,i}}{T_{Count\,i}},\ \rho^{\downarrow}_{i} = \frac{T^{\downarrow}_{SW\,i}}{T_{Count\,i}}.$$

Якщо $T^{\uparrow}_{SW\,i}$ і $T^{\downarrow}_{SW\,i}$ – вихідні параметри часу, що відповідають вхідному значенню часу $T_{SW\,i}$, де індекс ($\uparrow$) зображує пришвидшення обчислення, а ($\downarrow$) – уповільнення, то їх відношення до шуканого значення часу виконання завдання буде визначати значення коефіцієнта пришвидшення згідно з виразом (3.16), а значення коефіцієнта пришвидшення – пришвидшення (якщо $\rho_i > 1$) або уповільнення (якщо $\rho_i < 1$) обчислювального процесу. Тоді вираз (3.17) справедливий для випадків як пришвидшення, так і уповільнення обчислень.

На підставі виразу (3.17) виведемо залежність часу виконання завдань від коефіцієнта пришвидшення продуктивності:

$$T_{Count\,i} = T_{SW\,i} \cdot \frac{\sum_{i=1}^{n} T_{Count\,i}}{\sum_{i=1}^{n} T_{SW\,i}} \cdot \frac{\rho_i}{\left(\sum_{i=1}^{n} \rho_i\right)/n},\ \text{якщо}\ \rho_i = \frac{1}{\rho^{\uparrow}_{i}} = \rho^{\downarrow}_{i} = \frac{T_{Count\,i}}{T_{SW\,i}}. \qquad (3.18)$$

Вихідними параметрами для формалізації модифікованого способу визначення часових інтервалів виконання завдань є загальне обмеження часу T_{QoS} виконання алгоритму обчислення, яке накладається зовнішніми факторами. Часові параметри виконання завдань $T_{SW\,i}$, $T_{COUNT\,i}$ і значення коефіцієнтів пришвидшення продуктивності розраховані відповідно за виразом (3.16).

Накладання часових обмежень потребує від виконуваних завдань певного пришвидшення $\rho_{QoS\,i}$, тобто на час виконання кожного завдання з послідовності завдань накладаються такі обмеження:

$$T_{Count\,i} \le T_{QoS\,i} = \frac{T_{SW\,i}}{\rho_{QoS\,i}}$$

за умови загального пришвидшення обчислення, тобто

$$\rho_i = \frac{T_{SW\,i}}{T_{Count\,i}} > 1.$$

Для розв'язання задачі відповідності обмеженням згідно з виразом (3.18) розрахуємо значення $T_{QoS\,i}$:

$$T_{\mathrm{QoS}\,i} = T_{SW\,i} \cdot \frac{\sum_{i=1}^{n} T_{QoS\,i}}{\sum_{i=1}^{n} T_{SW\,i}} \cdot \frac{\rho_{\mathrm{QoS}i}}{\left(\sum_{i=1}^{n} \rho_{\mathrm{QoS}i}\right)/n},$$

$$T_{\mathrm{QoS}\,i} = T_{\,i} \cdot \rho_i \cdot \frac{T_{QoS}}{\sum_{i=1}^{n} T_{\,i} \times \rho_{Av}} \cdot \frac{\rho_{\mathrm{QoS}i}}{\left(\sum_{i=1}^{n} \rho_{\mathrm{QoS}i}\right)/n}.$$

Таким чином, шуканий вираз для визначення часових інтервалів виконання завдань набуває вигляду:

$$T_{QoS\,i} = T_{\,i} \cdot \left[\frac{T_{QoS}}{\sum_{i=1}^{n} T_{\,Count\,i}} \cdot \frac{\rho_{QoS\,i}}{\left(\sum_{i=1}^{n} \rho_{QoS\,i}\right)/n} \right]_{Av}^{\text{Корекція}} \cdot \frac{\rho_i}{\rho_{Av}}.$$

Порівняно з відомим способом [54] в отриманому виразі враховано похибку, яка вноситься припущенням, що заданий час обмеження T_{QoS} є середньою величиною часу, а заданий коефіцієнт пришвидшення – середнім коефіцієнтом пришвидшення. У цьому випадку справедливим був би такий розв'язок:

$$\rho_{QoS} = \frac{\sum_{i=1}^{n} T^{\uparrow}{}_{SW\,i}}{T_{QoS}} = \frac{\sum_{i=1}^{n} T_{\,i} \cdot \rho_{Av}}{T_{QoS}},\ T^{\uparrow}{}_{SW\,i} = T_{\,i} \times \rho_i,$$

тоді

$$T_{QoS\,i} = \frac{T^{\uparrow}{}_{SW\,i}}{\rho_{QoS\,i}} = T_i \cdot \left[\frac{T_{QoS}}{\sum_{i=1}^{n} T_{Count\,i}} \right]_{\text{Ср}}^{\text{Похибка}} \cdot \frac{\rho_i}{\rho_{Av}}.$$

З огляду на характерні особливості класів задач керування в режимі реального часу такі похибки є критичними під час реалізації обчислювального процесу. Часова діаграма на рис. 3.5 показує, що коригування сумарного часу обчислення до середньої величини відповідає коригуванню сумарного значення часу, виконаного у зворотному напрямі. Відповідно до коефіцієнтів відхилення від середнього перед виконанням обчислень доцільно відкоригувати вихідний час T_{QoS} на таку величину, що зводить його до середнього значення:

$$K_{QoS\,Correct} = \frac{\sum_{i=1}^{n} T^{\uparrow}{}_{SW\,i}}{\sum_{i=1}^{n} T_{\text{Count}} \cdot \rho_{Av}}.$$

Тоді відкориговане значення середнього пришвидшення відповідно до часових обмежень становитиме

$$\rho_{\text{QoS}} = \frac{T_{QoS}}{\sum_{i=1}^{n} T_{Count\,i}} \cdot \frac{\sum_{i=1}^{n} T^{\uparrow}{}_{SW\,i}}{\sum_{i=1}^{n} T_{Count} \cdot \rho_{Av}}. \qquad (3.19)$$

Остаточний вираз для визначення часових проміжків виконання завдань з урахуванням (3.19) має вигляд:

$$T_{\text{QoS}\,i} = T_i \cdot \left[\frac{T_{QoS}}{\sum_{i=1}^{n} T_{Count\,i}} \cdot \frac{\sum_{i=1}^{n} T^{\uparrow}{}_{SW\,i}}{\sum_{i=1}^{n} T_{Count} \times \rho_{Av}} \right] \cdot \frac{\rho_i}{\rho_{Av}}. \qquad (3.20)$$

Виконання часових обмежень відповідно до виразів (3.6) визначається виразом

$$T_{Count\,i} \le T_{QoS\,i},$$

що відповідає вихідним умовам (3.10) і (3.11).

Геометричну інтерпретацію проведених розрахунків ілюструє рис. 3.6.

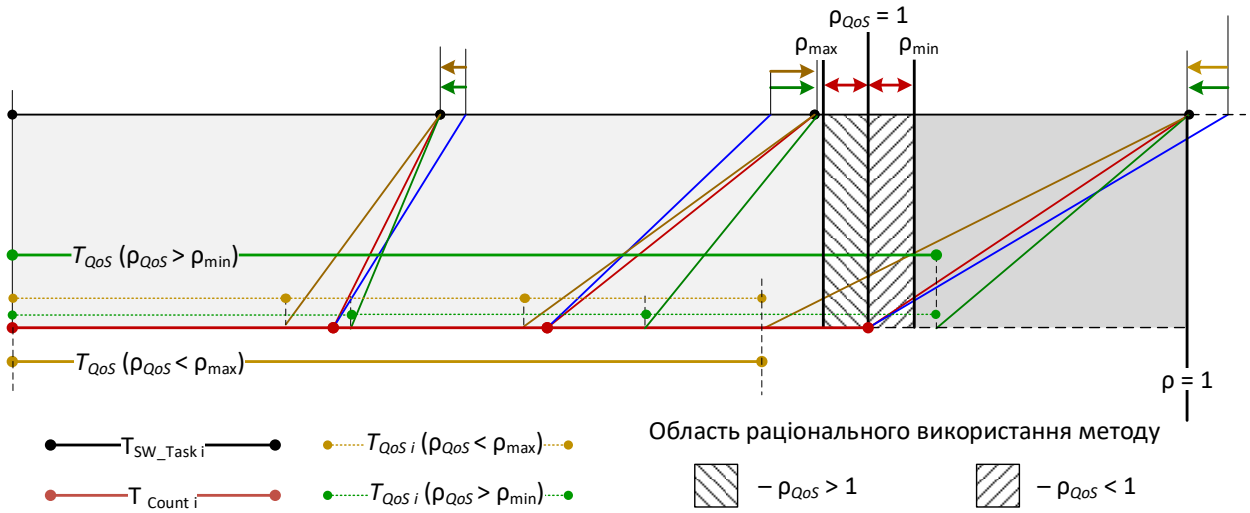


Рис. 3.6. Визначення часу виконання завдань відповідно до заданого показника пришвидшення

3.1.3. Визначення області раціонального використання способу визначення непродуктивного часу. На підставі наведених вище доведень, розрахунків та виразів (3.10), (3.11) і (3.20) визначимо умови раціонального використання способу визначення непродуктивного часу виконання завдань.

1. Випадки, коли пришвидшення обчислювального процесу менше за одиницю

$$\rho_{Av} < 1, \sum_{k=1}^{n} T_{SW\,i} < \sum_{i=1}^{n} T_i ,$$

тобто, коли фактично обчислювальний процес уповільнюється (рис. 3.5, 3.6), не розглядаються. Такі алгоритми можуть бути ефективно виконані засобами універсальних процесорних ядер.

2. Для випадків, коли часові обмеження потребують більшого або меншого пришвидшення від послідовності завдань ніж їх загальне середнє пришвидшення, умови забезпечення часових обмежень наведено в табл. 3.1.

3. Для визначення області раціонального використання способу визначення непродуктивного часу виконання завдань (табл. 3.1) використовується максимальний коефіцієнт відхилення від середнього значення:

– максимальний коефіцієнт відхилення в бік вимоги пришвидшення

обчислення:

$$\vartheta^{\uparrow} = \max(K^{\uparrow}{}_{Av\,i} \,|\, \forall(i = \overline{1,n}, \rho_i > \rho_{Av}));$$

– максимальний коефіцієнт відхилення в бік вимоги уповільнення обчислень:

$$\vartheta^{\downarrow} = \max(K^{\downarrow}{}_{Av\,i} \,|\, \forall(i = \overline{1,n}, \rho_i < \rho_{Av})).$$

Таблиця 3.1

Умови раціонального використання способу визначення непродуктивного часу

Параметри обчислювальної задачі	**Параметри завдання в ГА**	**Коментар**
$\rho_{\max} > \rho_{QoS} > \rho_{Av}$ $\rho_{\max} \to \rho_{Av}$	$K_{AV\,i} > 1,\ \rho_i > \rho_{Av}$	Завдання вкладаються (показник пришвидшення завдань більший від середнього пришвидшення)
	$K_{Av\,i} < 1,\ \rho_i < \rho_{Av}$	Завдання не вкладаються (показник пришвидшення завдань менший за середнє пришвидшення)
$\rho_{\max} < \rho_{QoS} > \rho_{\Sigma Av}$	Всі завдання	Жодне завдання не вкладається в обмеження
$\rho_{\min} < \rho_{QoS} < \rho_{Av}$ $\rho_{QoS} > 1$ $\rho_{\min} \to \rho_{Av}$	$K_{Av\,i} > 1,\ \rho_i > \rho_{Av}$	Завдання вкладаються (показник пришвидшення завдань більший від середнього прискорення)
	$K_{Av\,i} < 1,\ \rho_i < \rho_{Av}$	Завдання вкладаються (показник пришвидшення завдань більший від середнього пришвидшення)
$\rho_{\min} > \rho_{QoS} > 1$	Всі завдання	Усі завдання вкладаються в обмеження

У загальному випадку максимальний коефіцієнт відхилення від середнього значення визначається таким чином:

$$\vartheta = \max\left(\frac{\rho_i}{\rho_{Av}} \middle| \forall (i = \overline{1,n})\right),$$

тоді

$$\rho_{\max} = \rho_{QoS} \times \vartheta \middle| (\vartheta > 1),$$

$$\rho_{\min} = \rho_{QoS} \times \vartheta \middle| (\vartheta < 1),$$

де ρ_{QoS} розраховується відповідно до виразу (3.20).

Уведемо обмеження для максимального коефіцієнта відхилення від середнього значення:

$$\rho_{\max} \to \rho_{Av}\ ,\ \rho_{\min} \to \rho_{Av}\ .$$

Для цього використаємо показник середньоквадратичного відхилення δ, який є мірою відхилення величин від їх середніх значень [168]. Завдання, у яких пришвидшення продуктивності наближаються до значення 3δ,

$$\rho_i \to 3\delta\,, \tag{3.21}$$

значно впливають на рівномірність розподіл часу обмеження. Це з практичної сторони реалізації призводить до надлишкових дій щодо мінімізації часу виконання завдання, і, як наслідок, до збиткового використання апаратних та програмно-апаратних ресурсів системи.

На підставі визначених умов (табл. 3.1) можна сформувати наступні раціональні межі застосування модифікованого способу визначення непродуктивного часу виконання завдань:

$$\begin{cases} \rho_{Av} > 1, \\ \rho_{\max} > \rho_{QoS} > 1, \\ \rho_{\max} \to \rho_{Av}\ ,\ \rho_{\min} \to \rho_{Av}. \end{cases} \tag{3.22}$$

3.1.4. Основні етапи методу оптимізації процесу відображення обчислювальних задач на реконфігуровне обчислювальне середовище. Для реалізації методу оптимізації процесу відображення обчислювальних задач застосовується описаний вище модифікований спосіб визначення непродуктивного часу виконання завдань, який узагальнений виразами (3.10) і (3.20), а також визначено область його раціонального використання (3.22). Реалізація методу складається з етапів, основні з яких відображено на часовій діаграмі на рис. 3.7.

Етап 1. Визначення раціональної межі використання методу на підставі виразу (3.22):

$$\begin{cases} \rho_{Av} > 1, \\ \rho_{\max} > \rho_{QoS} > 1. \end{cases}$$

Етап 2. Видалення з послідовності завдань B (3.8) множини завдань D ($T_i \in D$, якщо $T_{Rconf\,i} = \max$), час виконання яких неможливо зменшити, зокрема виконання завдань першого рівня ЯПФ відповідно до математичних моделей визначення часу (2.36) і (2.37):

$$T_{v_1} \in D, \text{ якщо } T_{v_1} = \max\{T_{v_1} \,|\, v_1 = \overline{1, H_1}\}, \qquad (3.23)$$

тоді

$$T_{QoS} = T_{QoS} - \sum_{i=1}^{n} (T_i \,|\, \forall T_i \in D).$$

Етап 3. Забезпечення виконання умови $\rho_{\max} \to \rho_{QoS}$.

Етап 4. Розрахунок часу обмеження для завдань, що залишилися в послідовності, відповідно до виразу (3.20). Послідовність розрахунків згідно з модифікованим способом визначення часових інтервалів зображено на часових діаграмах на рис. 3.7, I, II:

$$T_{QoS\,i} = T_i \cdot \left[\frac{T_{QoS}}{\sum_{i=1}^{n} T_{Count\,i}} \cdot \frac{\sum_{i=1}^{n} T^{\uparrow}_{SW\,i}}{\sum_{i=1}^{n} T_{Count\,i} \cdot \rho_{Av}} \right] \cdot \frac{\rho_i}{\rho_{Av}} \,\Big|\, \forall (T_i \notin D).$$

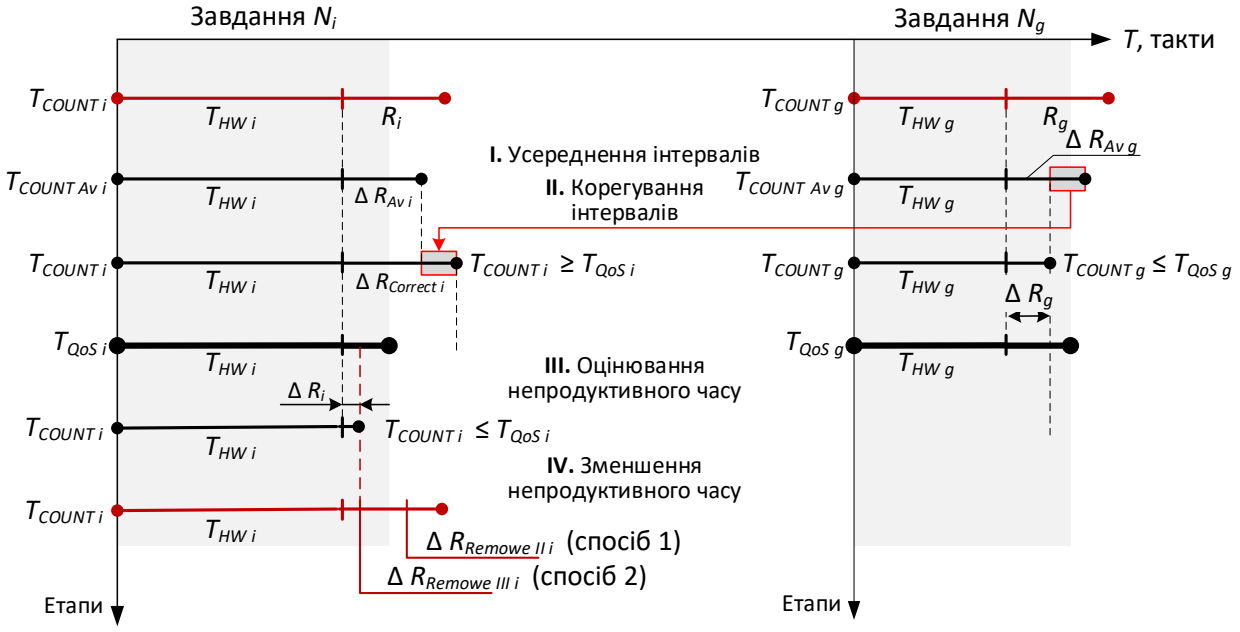


Рис. 3.7. Часова діаграма виконання основних етапів методу оптимізації відображення обчислювальних задач на реконфігуровне обчислювальне середовище на ПЛІС: ▒ – ділянка обмеженого часу виконання завдань

Етап 5. Визначення стратегії обслуговування кожного завдання T_i з послідовності завдань B_{GM} (3.8):

– завдання, що отримали достатньо часу для виконання, згідно з виразами (3.10) і (3.11) виконуються засобами стандартної послідовності реконфігурації відповідно до виразів (2.17) і (2.20). Як приклад наведено завдання N_l на рис. 3.7;

– для завдань, що не вкладаються в часові обмеження, згідно з виразом (3.12), визначається стратегія обслуговування відповідно до виразів (2.18) і (2.19), що різною мірою забезпечують умову (3.13):

$$\rho_i \left| \forall (T_{Count\,i} < T_{QoS\,i}) = \frac{T_{SW\,i}}{T_{Count\,i}} = \frac{T_{SW\,i}}{T_{Rconf\,i} + T_{HW\,i}} \right. \Rightarrow \begin{cases} T_{Rconf\,i} \to \min, \\ T_{Count\,i} \to T_{HW\,i}. \end{cases}$$

Залежно від обраної стратегії обслуговування, що ґрунтується на різних рівнях кешування конфігураційних даних, визначається непродуктивний час виконання, який забезпечує оптимальні параметри відображення завдань за критеріями заданих часових вимог і просторових обмежень реконфігуровної

області.

На підставі визначення критичних параметрів графів алгоритмів обчислювальних задач (2.15) і (2.16) отримаємо вираз для визначення умови забезпечення просторових обмежень реконфігуровної області:

$$\sum_{j=1}^{L} S_{CACHE\,j} + S_i + \sum_{h=1}^{H_k} S_h \middle| S_h \notin S_{CACHE} \le S_{FPGA}, \tag{3.24}$$

де S_{CACHE} – площа реконфігуровної ділянки, що задіяна під кеш-пам'ять апаратних задач $S_{CACHE} = \sum_{j=1}^{L} S_{CACHE\,j}$; S_{FPGA} – площа реконфігуровної області ПЛІС; L – кількість типів апаратних задач, розміщених в кеш-пам'яті; S_i – площа поточного завдання. За відображенням ГА обчислювальної задачі ярусами (2.26), складова $\sum_{h=1}^{H_k} S_h \middle| S_h \notin S_{CACHE}$ визначає площу, задіяну для виконання завдань чергового ярусу за винятком тих, які вже сконфігуровані на поверхні ПЛІС.

Відповідно до умови забезпечення часових обмежень (3.14) та виразів (2.29) і (2.30) функція зменшення непродуктивного часу визначається відповідно за такими способами:

Спосіб 1. Кешування конфігураційних даних у локальній пам'яті обчислювального модуля без використання ресурсів ПЛІС (рис. 3.7, IV):

$$T_{HW\,i} + (\Delta R_i - \Delta R_{Remove\_II\,i}) \le T_{QoS\,i}. \tag{3.25}$$

Такий спосіб не забезпечує інтенсивного пришвидшення процесу реконфігурації, але і не впливає на складову виразу 3.7. Тому, якщо умова (3.25) виконується, такий спосіб є оптимальним щодо забезпечення часових умов і використання ресурсів ПЛІС.

Спосіб 2. Кешування конфігурацій апаратних задач на поверхні реконфігуровної області кристала ПЛІС (рис. 3.7, IV):

$$T_{HW\,i} + (\Delta R_i - \Delta R_{Remove\_III\,i}) \le T_{QoS\,i}. \tag{3.26}$$

Такий спосіб забезпечує інтенсивне пришвидшення процесу реконфігурації згідно з виразами (2.36) і (2.37), але збільшує складову виразу

(3.24). Тому, якщо умова (3.6) не виконується, необхідно забезпечити виконання умови (3.24), що дозволяє використовувати ресурси ПЛІС для кешування конфігурації чергової апаратної задачі.

3.1.5. Дослідження ефективності методу оптимізації відображення задач на реконфігуровне обчислювальне середовище. Експериментальні результати отримано шляхом моделювання методу оптимізації відображення обчислювальних задач на реконфігуровне обчислювальне середовище засобами програмного емулятора реконфігуровної обчислювальної системи та програмної моделі реалізації запропонованого методу [165, 166, 171].

Досліджено тестові послідовності завдань з різним часом апаратного виконання і різними показниками пришвидшення, що відповідають галузі застосування РКС. Як параметри часу тестових завдань використовують реальні характеристики синтезованих на ПЛІС апаратних функціональних ядер [15, 54, 68, 107, 108, 131, 135, 136, 140, 166].

Досліджено серію обчислювальних задач, поданих графами ЯПФ, у вузлах яких розміщуються матричні функції різної розмірності, апаратні реалізації яких синтезовано на ПЛІС *Cyclone* II *Altera* [132, 167].

Із графіка на рис. 3.8 видно, що обернено пропорційне коригування (T_{QoS_Mod}) усереднених проміжків часу (T_{QoS_Av}) надає критичним завданням, що мають коефіцієнт пришвидшення вищий ніж середній, додаткового часового ресурсу. На графіку видно, що завдання 2, 5 – 6 отримали додатковий часу і вийшли із класу критичних завдань, завдання 5, яке має дещо нижчий ніж середній коефіцієнт пришвидшення, також отримало достатньо часу для виконання.

У послідовності завдань можуть бути окремі завдання, що не входять у межі раціонального використання запропонованого способу (3.22), наприклад, задачі першого ярусу алгоритму ЯПФ (3.23) або завдання з показником пришвидшення, що значно відхиляється від середнього значення прискорення, відповідно до виразу (3.21). Такі задачі видаляються із загальної послідовності

і отримують необхідний час для виконання із загального проміжку часового обмеження.

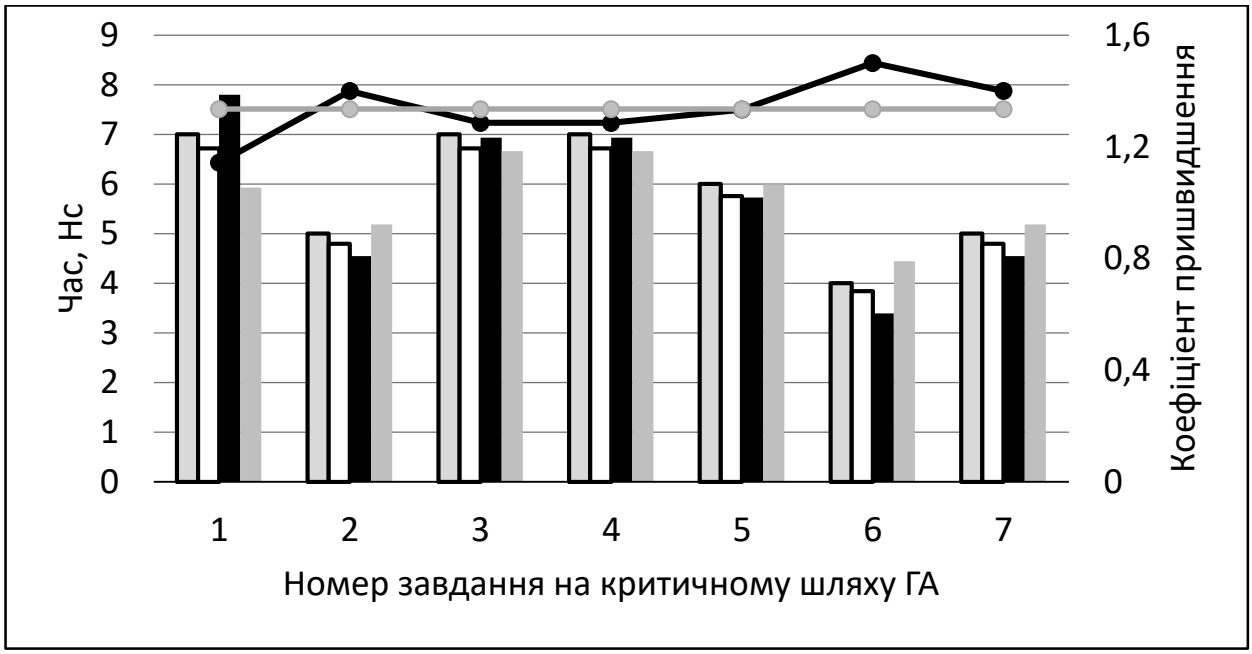


Рис. 3.8. Надання критичним задачам додаткового часу виконання: ▢ – очікуваний час обчислення завдання (T_{Count}); □ – розподіл за усередненим показником ($T_{QoS\ Av}$); ■ – коригування відомим способом ($T_{QoS\ i}$); ▢ – коригування модифікованим способом ($T_{QoS\ i}$); ● – показник пришвидшення завдання ρ; ● – середній показник пришвидшення завдання ρ_{Av}

Порівняльні залежності часу обчислення від застосування різних механізмів пришвидшення обчислювального процесу для алгоритмів з різною кількістю однотипних завдань зображено на рис. 3.9.

Із графіка (рис. 3.9) видно, що обчислення, виконані засобами стандартної послідовності реконфігурації, потребують більше часу за рахунок значної переваги часу реконфігурації над часом виконання апаратної задачі. При цьому час обчислення не залежить від типів виконуваних завдань. Запобігання повторному завантаженню однотипних завдань сприяє інтенсивному пришвидшенню обчислювального процесу [158].

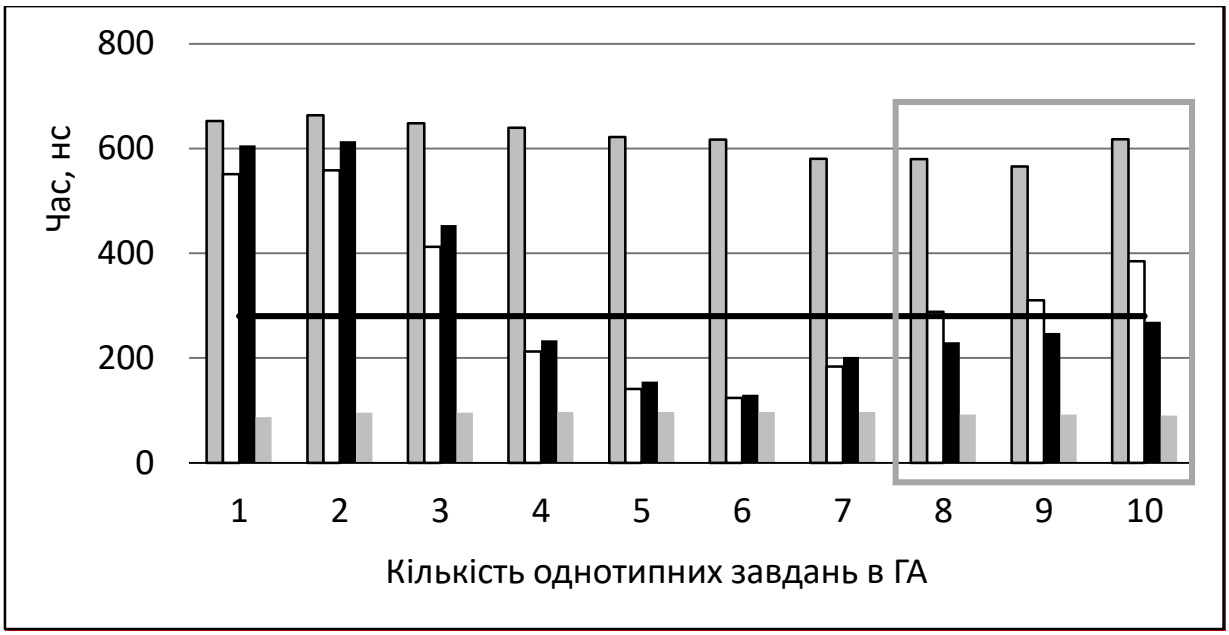


Рис. 3.9. Дослідження часу реконфігурації: ▭ – стандартний процес реконфігурації; ▭ – адаптивне відображення задач на реконфігуровне обчислювальне середовище; ■ –оптимізація відображення задач; ▭ – час обчислення задач на апаратурі ПЛІС; ━ – обмеження часу виконання обчислювальної задачі; ▭ – область раціонального використання методу оптимізації відображення задач

Як видно з графіка (рис. 3.9), мінімальний час обчислення досягається за умови відповідності ширини графу ЯПФ до просторових параметрів реконфігуровного середовища ПЛІС, яку визначено у праці [155]. Зі збільшенням кількості однотипних завдань виникає потреба у запровадженні додаткових засобів для завантаження, або створення копій активних завдань. Це потребує додаткового часу за кількості однотипних завдань понад 50%, що зображено на графіку (рис. 3.9). Запропонований метод дозволяє оптимізувати технологію інтенсивного пришвидшення реконфігурації, описану у праці [158], що видно на діаграмі (рис. 3.9): в області раціонального використання запропонованого методу час виконання завдань укладається у ці часові обмеження.

Із діаграми дослідження показника пришвидшення (рис. 3.10) видно, що механізм повторного використання ресурсів надає інтенсивного

пришвидшення реконфігурації зі збільшенням типів завдань до кількості, сумірної із шириною графу ЯПФ. За сумірної з розміром реконфігуровної області ширини графу ЯПФ та великої кількості однотипних завдань досягається збільшення інтенсивності пришвидшення реконфігурації в середньому на 63%. У процесі подолання просторових обмежень ПЛІС відбувається різке зменшення інтенсивності пришвидшення реконфігурації в середньому на 85%.

Рис. 3.10. Дослідження показників прискорення: ▬ – адаптивне відображення задач на реконфігуровне обчислювальне середовище; ▬ – оптимізація відображення задач

За результатами експериментів у праці [165] установлено, що запропонований метод оптимізації відображення обчислювальних задач на реконфігуровне обчислювальне середовище на ПЛІС зменшує інтенсивність впливу просторових обмежень на швидкість обчислення в цьому випадку приблизно на 10%, що відповідає графічній залежності на рис. 3.10. Але цей вимір залежить від загальної кількості видалених непродуктивних витрат часу в межах оптимальних параметрів реконфігурації згідно з виразом (3.7).

Із практичного погляду метод оптимізації відображення обчислювальних задач на реконфігуровне обчислювальне середовище на ПЛІС означає, що задачі з високою ефективністю апаратної реалізації

порівняно з їх програмною реалізацією ефективно розв’язуються за допомогою стандартного процесу реконфігурації, в противному випадку запроваджуються додаткові механізми пришвидшення обчислень, зокрема, через зменшення комунікаційних затримань під час реконфігурації обчислювального середовища. Усереднені проміжки часу коригуються таким чином, що завданням з високим показником пришвидшення (більше ніж середній) виділяється додатковий час за рахунок завдань, пришвидшення яких невелике (нижче ніж середнє).

3.2. Спосіб визначення оптимального співвідношення зернистості розв’язуваних задач та структури обчислювального середовища на ПЛІС

3.2.1. Обґрунтування області оптимального співвідношення параметрів розв’язуваних задач та структури обчислювального середовища на ПЛІС. Методи підвищення ефективності обчислень, що широко використовуються в традиційних високопродуктивних обчислювальних системах, ґрунтуються на варіюванні зернистістю обчислювального алгоритму відповідно до постійної архітектури обчислювальної системи [104, 106]. Зменшення зернистості приводить до зниження обчислювальної складності виконання алгоритму задачі і збільшення непродуктивних комунікаційних витрат. Збільшення зернистості призводить не тільки до значного зменшення руху даних, але й до збільшення обчислювальної складності паралельно виконуваних функцій.

У праці [167] визначено характер залежності часу виконання обчислювальних задач від зернистості обчислень (рис. 3.11), на підставі якої визначена область оптимальних параметрів зернистості обчислювального алгоритму, що визначається співвідношенням обчислювальної складності виконання «зерна» обчислень і обсягом комунікаційних витрат.

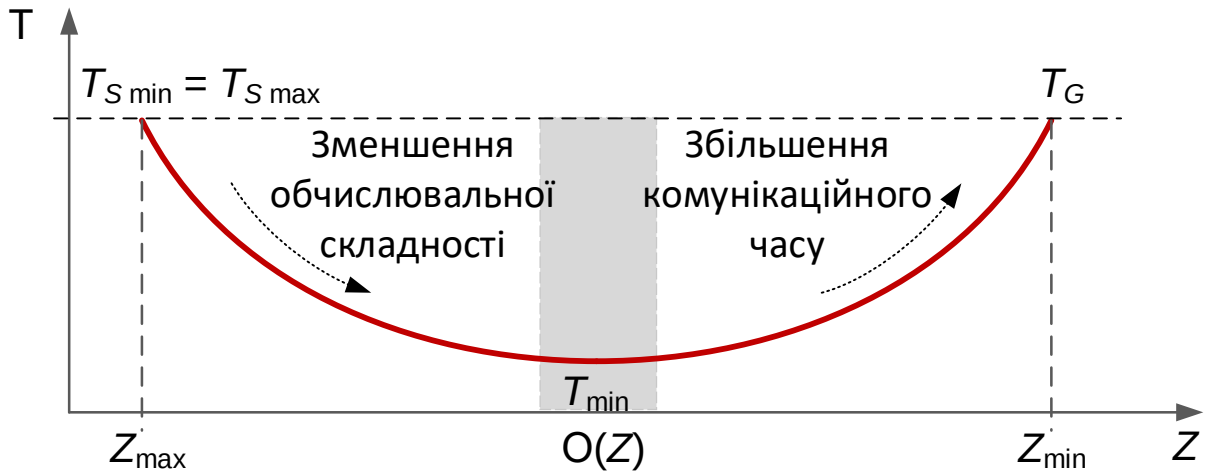


Рис. 3.11. Визначення оптимальних параметрів часу виконання завдань на апаратурі ПЛІС

У динамічно РКС збільшення «зерна» обчислювального середовища відповідно до вимог алгоритмів розв'язуваних задач призводить не тільки до значного збільшення накладних витрат під час налаштування реконфігуровного обчислювального середовища, але і до значного зменшення часової складності виконання функції порівняно з програмною реалізацією. Це зумовлено високою швидкодією апаратного обчислювального середовища і внутрішніх каналів передавання даних ПЛІС. При цьому час передавання вхідних та вихідних даних зовнішніми комунікаційними середовищами з урахуванням затримань інтерфейсів ПЛІС значно переважає час обчислення функцій на апаратурі ПЛІС. Таким чином, зменшення «зерна» обчислювального середовища для реалізації дрібнозернистих алгоритмів приводить до переважної кількісті операцій обміну даними у процесі обчислень, що спричиняє великі непродуктивні витрати передавання даних. Ці витрати значно переважають виграш у зменшенні накладних витрат реконфігурації. З аналізу діаграм (рис. 3.2, *а* і 3.11) випливає, що вибір зернистості обчислювального середовища потребує оптимізації згідно з обсягом непродуктивних витрат часу реконфігурації обчислювального середовища, обсягу непродуктивних витрат часу передавання даних і часової складності виконуваних функцій. При цьому зернистість обчислювального

середовища є важливим фактором, що впливає на час виконання обчислень, тому цільовою функцією є час обробки інформації в РКС.

Оптимізацію зернистості обчислювального середовища враховано інтегрованим критерієм оптимізації, який запропоновано у підрозділі 3.1.1.

Для кількісної оцінки співвідношень 3 – 5 інтегрального критерію оптимізації і відповідно забезпечення екстремумів цільових функцій зменшення накладних витрат та зменшення часу виконання завдань на всіх рівнях процесу обробки інформації у праці [165] запропоновано нову стратегію взаємної адаптації розв'язуваних задач і обчислювального середовища. Процес взаємної адаптації визначається такою формальною моделлю:

$$\mathrm{A}(FPGA, G_M) = \mathrm{A}(G_M, FPGA) \Rightarrow \begin{cases} \min(\Delta R), \\ \max(T_{HW}) = \min(T_{Task}) + \max(T_{IO}), \end{cases}$$

де A – функція адаптації; T_{Task} – час обчислення апаратної задачі на ПЛІС; T_{IO} – час передавання вхідних та вихідних даних; T_{HW} – час обчислення апаратної задачі з урахуванням часу пересилання вхідних та вихідних даних.

Для реалізації взаємної адаптації в дисертаційній роботі запропоновано визначення оптимального співвідношення зернистості розв'язуваних задач і обчислювального середовища, що відповідає максимальному показнику продуктивності обчислювального середовища на ПЛІС. Це формально подано таким виразом:

$$\mathrm{O}(Z_{GM}, Z_{FPGA}) \Rightarrow \begin{cases} Z_{GM} = Z_{FPGA}, \\ \min(T_{COUNT}). \end{cases} \qquad (3.27)$$

3.2.2. Спосіб визначення оптимального співвідношення зернистості розв'язуваних задач і зернистості реконфігуровного обчислювального середовища на ПЛІС. Відповідно до класичного визначення [106] зерном обчислень є множина операцій, що виконується атомарно, при цьому обмін даними виконується масивами або блоками. Таким чином, вся множина

операцій обчислювальної задачі розбивається на певні макрофункції (макрооперації), що розміщуються у вузлах графу [55, 109, 155] і асоціюються з поняттям «зерна» обчислень. Макрооперація є певною функцією *F*, що апаратно реалізується ФБ, налаштованим на ПЛІС. Операцією пересилання даних F_{IO} є введення вхідних та виведення вихідних даних для виконання одного «зерна» обчислень у кожному ФБ. Дрібнозернисті обчислення характеризуються виконанням однієї операції у кожному ФБ, на вхід якого надходять одне або два слова вхідних даних і виводиться одне слово результату – одномісні та двомісні обчислення [51]. Варіюванням зернистістю обчислень є збільшення або зменшення кількості операцій, що виконуються в одному ФБ, і відповідно до цього, збільшення або зменшення порції даних під час однієї операції пересилання.

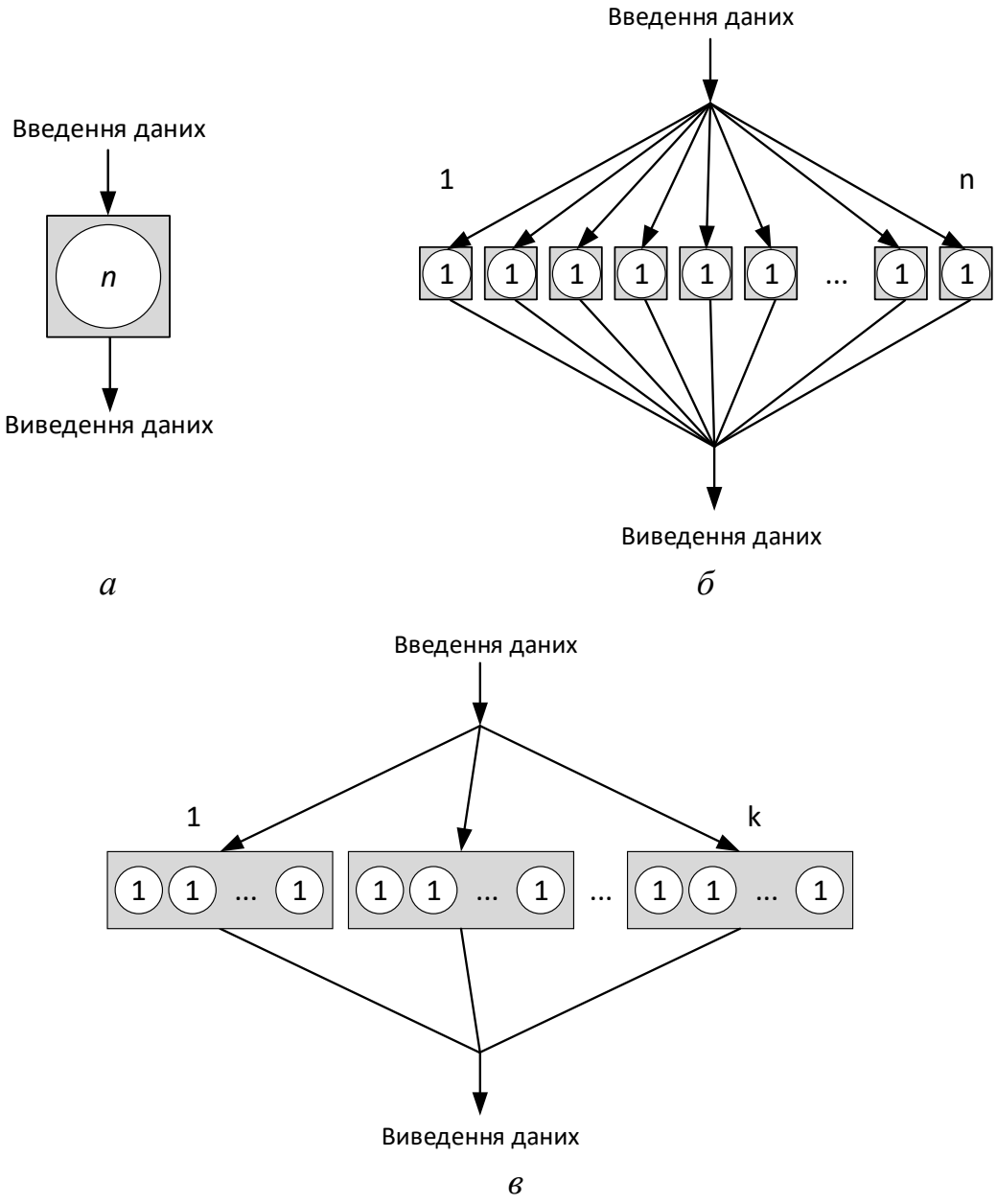
Коефіцієнт зернистості алгоритму обчислювальної задачі виражає відношення обчислювальної складності алгоритму до обсягу обміну даними [165]:

$$Z = \frac{V_{Count}}{V_{IO}},$$

де V_{Count} – часова складність обчислювального алгоритму, що дорівнює кількості виконаних операцій; (V_{IO}) – кількість виконаних операцій пересилань даних.

Тоді для деякого ярусу графу G_M алгоритму ЯПФ визначимо максимальний коефіцієнт зернистості, коли один функціональний блок виконує весь обсяг обчислень (рис. 3.12, *а*) і мінімальний коефіцієнт зернистості, коли алгоритм максимально розпаралелюється, тобто в кожному ФБ виконується одна операція (рис. 3.12, *б*):

$$S_{\max} = \frac{n}{1},\ S_{\min} = \frac{1}{n}.$$



а *б* *в*

Рис. 3.12. Схема визначення зернистості обчислювального середовища: □ – ФБ – «зерно» обчислень; (n) – кількість операцій, що виконуються в ФБ

Результат варіювання зернистістю множини операцій до деякого ступеня k та відображення цього алгоритму на відповідну кількість ФБ зображено на рис. 3.12, *в*, де k – зернистість (кількість «зерен») алгоритму обчислювальної задачі, що має розмірність n, при цьому $k \le n$, (рис. 3.12, *в*). Тоді коефіцієнт зернистості алгоритму обчислювальної задачі

$$Z_G = \frac{n/k}{k} = \frac{n}{k^2}. \qquad (3.28)$$

Оптимізація співвідношення обчислювального алгоритму і структури обчислювального середовища має за мету збільшення ефективності паралельних обчислень через підвищення швидкодії обчислень на апаратурі ПЛІС. Для цього визначимо показник продуктивності реконфігуровного обчислювального середовища на ПЛІС, що виражає відношення швидкості оброблення (v_{Count}) до швидкості передавання даних (v_{IO}):

$$P_{FPGA} = \frac{v_{Count}}{v_{IO}}. \tag{3.29}$$

Для оцінювання швидкодії ФБ застосуємо традиційний параметр частоти роботи F_{func}. Зазвичай частота роботи обчислювального ядра визначається як кількість операцій виконуваних за одиницю часу. Однією з операцій є елементарна дія відповідно до табл. 3.2.

Таблиця 3.2

Визначення параметра частоти роботи ФБ

Кількість тактів, τ	Час виконання операції, с	Частота роботи ФБ, МГц
1	t_{op}	$\Rightarrow F = \frac{1}{t_{op}}$
x	1	

ФБ є обчислювальною структурою, що виконує певну послідовність операцій для реалізації завдання у вершині графу алгоритму обчислювальної задачі на апаратурі ПЛІС. Тоді припустімо виконувану функцію за одиницю обчислення для цього ФБ і подамо частоту його роботи як кількість операцій, виконуваних за одиницю часу:

$$F_{func} = \frac{1}{t_{func}},$$

де t_{func} – час виконання обчислювальної задачі. Для оцінювання швидкості обробки даних усієї реконфігуровної структури F_{FPGA} отримаємо такий вираз:

$$F_{FPGA} = F_{func} \cdot k`, \tag{3.30}$$

де $k`$ – кількість ФБ.

Швидкість передавання даних обмежується технічними показниками інтерфейсів введення-виведення ПЛІС та каналів передавання даних між пам’яттю та реконфігуровною структурою і не перевищує значення F_{IO} – частоти роботи зовнішньої пам’яті. Тоді частота завантаження ФБ апаратних задач F_{IO_func}, реалізованих на поверхні ПЛІС, у межах реалізації визначеного алгоритму обчислення визначається як

$$F_{IO\_func} = \frac{F_{IO}}{m}. \tag{3.31}$$

На підставі виразів (3.29) – (3.31) отримаємо критерії продуктивності реконфігуровного обчислювального середовища на ПЛІС:

$$P_{FPGA} = \frac{F_{FPGA}}{F_{IO\_func}} = \frac{F_{func} \cdot k`}{F_{IO}/m},\ K_{FPGA} \geq 1. \tag{3.32}$$

Виходячи з виразів (3.27), (3.28) і (3.32), отримуємо вираз для визначення зернистості, що виражає оптимальне співвідношення обчислювального алгоритму та структури реконфігуровного обчислювального середовища щодо мінімізації часу паралельних обчислень на апаратурі ПЛІС з урахуванням просторових обмежень кристалів ПЛІС:

$$k = \frac{n}{F_{func}} \cdot \sqrt{\frac{F_{IO}}{F_{func}}}, \tag{3.33}$$

де n – розмірність задачі, а $m = n/k$ справедливе для регулярних обчислювальних структур. Це окремий випадок, який охоплює регулярні обчислювальні структури, зокрема для обчислення функцій лінійної алгебри, матричні операції тощо.

3.2.3. Дослідження фізичної природи затримок поширення сигналів в ПЛІС, що впливають швидкодію реконфігуровного обчислювального середовища. Зробимо припущення, що фізичні властивості кристалів ПЛІС зумовлюють вплив розміру ФБ на часові затримання під час передавання

сигналів внутрішніми каналами зв'язку. Визначимо такі затримання як внутрішні затримання реконфігуровного середовища, які опишемо виразом

$$d_{FPGA} = d_{comm} + d_{io},$$

де d_{comm} – внутрішні комунікаційні затримки; d_{io} – час затримання пересилання масивів даних між пам'яттю та ПЛІС.

Апаратна реалізація обчислень дозволяє ефективно реалізовувати блокове передавання даних. Тоді теоретично час виконання деякого алгоритму за реалізації мінімального ($S_{\min}$) і максимального ($S_{\max}$) ступенів зернистості можна зменшити, зменшивши затримання звернення до пам'яті:

$$T_{S_{\min}} = [n \cdot [m_{\min} \cdot (t_{io} + t_0) + \tau]],\ T_{S_{\max}} = [n \cdot \tau + [t_0 + m_{\max} \cdot t_{io}]], \qquad (3.34)$$

де τ тактів – час виконання однієї операції; m – кількість слів переданих даних під час виконання одного «зерна» алгоритму; t_{io} – час передавання одного слова даних; t_0 – час звернення до пам'яті. Квадратними дужками позначено атомарність операцій, що відповідає передаванню одного блока даних. Із виразів (3.34) випливає, що час виконання зменшується зі збільшенням ступеня зернистості алгоритму ($T_{S_{\max}} < T_{S_{\min}}$). Зважаючи на те, що кількість тактів, необхідні для запису або зчитування даних з пам'яті пропорційні кількості елементів обчислювальних матриць, необхідно визначити реальну залежність затримань від конфігурації реконфігуровного обчислювального середовища.

Реконфігуровні обчислення на фізичному рівні кристала ПЛІС координує машина станів, синтезована мовою *Verilog*. Таким чином, природа виникнення внутрішніх комунікаційних затримань обґрунтовується виконанням алгоритму керування, що супроводжується передаванням відповідних сигналів керування внутрішніми каналами зв'язку.

У праці [165] було проведено експерименти з обчисленням матричних функцій різної розмірності, які характеризуються відповідно різними об'ємами блоків даних, що звантажуються для обчислення. Обсяг даних у цьому випадку залежить від розмірності одного слова даних. У результаті

отримано залежність затримання передавання даних від розміру ФБ і обсягу даних, що передаються для обчислення певної функції, яку зображено на рис. 3.13.

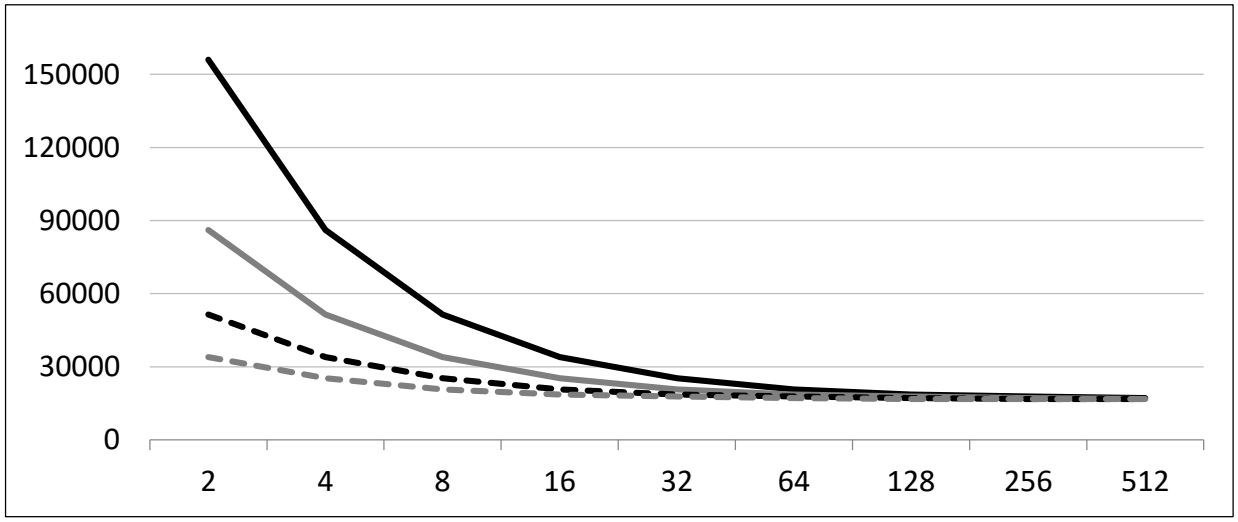


Рис. 3.13. Дослідження затримок блокового пересилання даних: затримки звернення до пам'яті під час передавання блоку даних розміром: ━ – 16384 біт; ━ – 8192 біт; ╍ – 4096 біт; ╍ – 2048 біт

За результатами експериментів отримано, що розмірності блоків від 2 до 32 характеризуються значними затриманнями, які зменшуються зі збільшенням розмірів блоків. Це пояснюється виразом, що випливає з виразу (3.34): $d_{io} = k \cdot t_0 + m \cdot t_{io}$, де k – кількість блоків даних; t_0 – час установлення початкової адреси та отримання сигналу готовності даних, час читання-запису одного слова відповідає одному такту ($t_{io} = 1\tau$). Подальше збільшення розміру блока зменшує затримання таким чином, що вона перестає залежати від розміру блока даних. Причиною цього є те, що значення m стає настільки великим, що добуток $m \times t_{io}$ відіграє більшу роль, ніж $k \times t_0$.

Досліджено вплив просторових параметрів ФБ на часові затримання під час передавання сигналів внутрішніми каналами зв'язку. На рис. 3.14 зображено результати дослідження частоти роботи ФБ для реалізації обчислювальних алгоритмів лінійної алгебри. На діаграмах видно, що

швидкість обчислення зменшується до 50 % зі збільшенням розмірності обчислюваної функції, що відповідає збільшенню просторових параметрів функціонального блока. При цьому інтенсивність уповільнення залежить і від технічних характеристик мікросхеми.

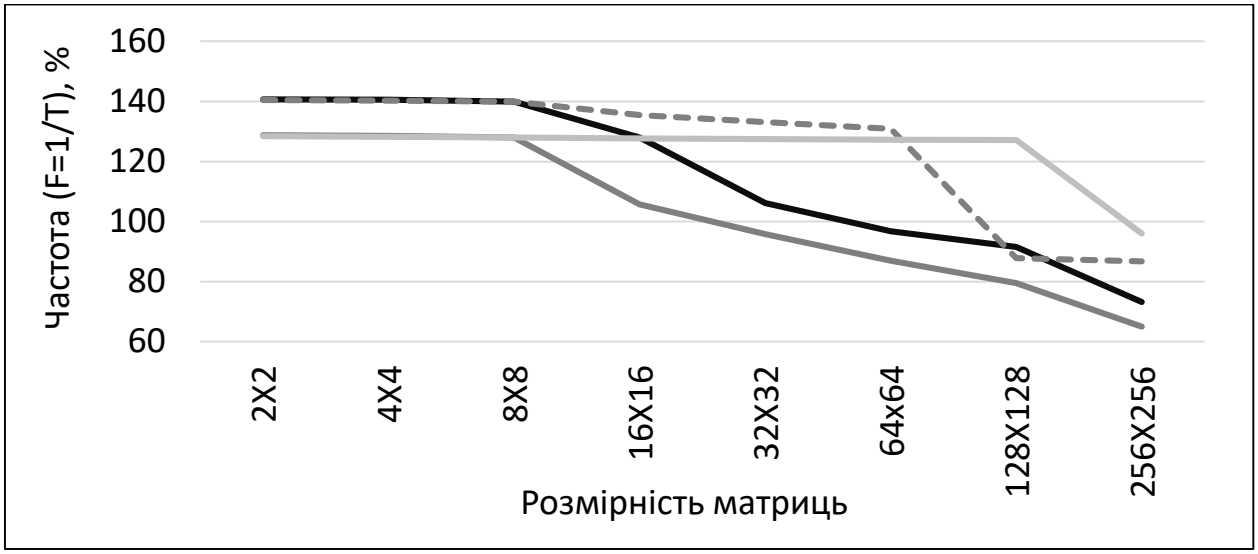


Рис. 3.14. Експериментальна оцінка продуктивності функціональних блоків, побудованих на базі сімейств мікросхем: – *EP*4*CGX*150*DF*31C8; – *EP*4*CGX*110*DF*31*C*7; – *EP*4*CGX*110*DF*27*C*7; – *EP*4*CGX*30*CF*23*C*6

Результати дослідження ефективності зміни зернистості обчислення згідно запропонованого виразу (3.33) представлені на рис. 3.15.

Дослідження виконано для двох сімейств кристалів ПЛІС з високими характеристиками продуктивності, що є доступними за ціною та актуальними для використання в лабораторних умовах.

Усереднені показники часових характеристик функціональних блоків різної розмірності для виконання операції підсумовування матриць наведено в додатку А.

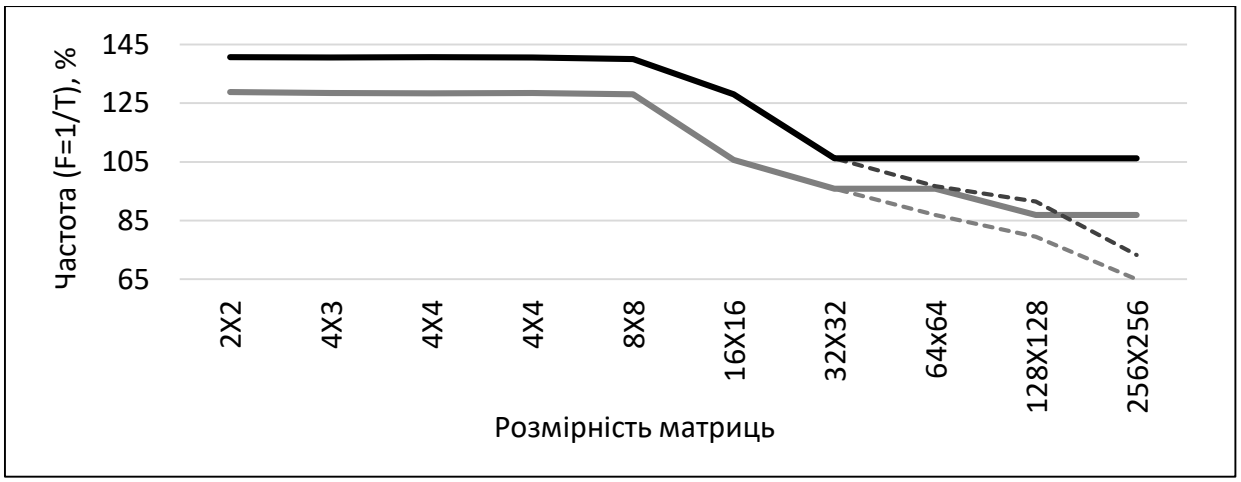


Рис. 3.15. Дослідження ефективності варіювання зернистістю обчислень на сімействі мікросхем *EP*4*CGX*110*DF*31*C*7: ▬ – великозернисті обчислення; ▬ – оптимізація зернистості; на сімействі мікросхем *EP*4*CGX*150*DF*31C8: ▬ – великозернисті обчислення; ▬ – оптимізація зернистості

3.2.4. Моделювання способу визначення оптимального співвідношення зернистості розв’язуваних задач і обчислювального середовища на ПЛІС. В роботі проведено дослідження для серії обчислювальних алгоритмів, поданих макрографами потоку даних (див. рис. 2.2), структуру яких згенеровано випадковим чином [165, 166]. У вершинах досліджуваних графів алгоритмів розміщуються макрозадачі, що являють собою випадковий набір матричних операцій різної розмірності. На базі розробленої бібліотеки ФБ виконано моделювання способу визначення оптимального співвідношення вимог обчислювальної задачі та структури реконфігуровного обчислювального середовища на ПЛІС відповідно до виразу (3.34). На підставі проведених експериментів отримано залежності часу обчислень від обсягу однотипних задач в обчислювальному алгоритмі. На графіках (рис. 3.16) зображено ефект від застосування запропонованого способу визначення оптимального співвідношення зернистості розв’язуваних задач і зернистості реконфігуровного обчислювального середовища. Ефективність застосування запропонованого способу порівняно з ефективністю застосування методу

адаптивного відображення задач на реконфігуровне обчислювальне середовище [157] без будь-якої оптимізації.

На критичних ділянках, яким характерне виникнення просторових обмежень кристала ПЛІС, різко зменшується інтенсивність пришвидшення реконфігурації в середньому на 85 % [165, 166]. Запропонований спосіб визначення оптимального співвідношення вимог обчислювальної задачі та структури реконфігуровного обчислювального середовища на ПЛІС зменшує інтенсивність впливу просторових обмежень на швидкість обчислення в розгляданому випадку в середньому на 10 %.

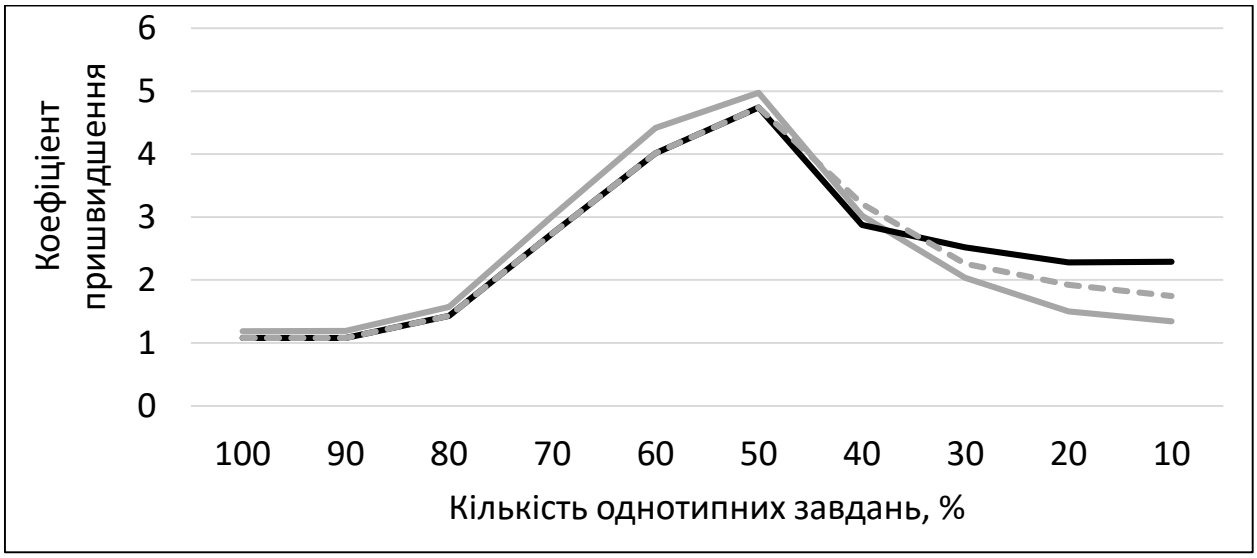


Рис. 3.16. Дослідження показників прискорення: – адаптивне відображення задач на реконфігуровне обчислювальне середовище; – оптимізація відображення задач; – оптимізація зернистості обчислень

Виконані у працях [165, 166] дослідження підтвердили теоретичне положення щодо впливу особливостей фізичних процесів поширення сигналів на рівні кристалів ПЛІС на ефективність реконфігуровних обчислень. Графіки на рис. 3.13 показують, що за реалізації максимального коефіцієнта зернистості зі збільшенням розмірності задачі різко зменшується швидкість обчислення. На графіках на рис. 3.15 подано результати застосування запропонованого способу визначення оптимальної відповідності

обчислювального алгоритму та структури обчислювального середовища. Оптимізація зернистості реконфігуровних обчислень на підставі запропонованого способу пришвидшує обчислення в середньому від 9 % до 15 % і залежить від технічних показників кристалів ПЛІС.

Дослідженням ефективності застосування блокового передавання даних, що теоретично приводить до лінійного зменшення затримань уведення-виведення зі збільшенням розміру блоків даних, доведено, що зі збільшенням розміру блоків даних для матричних операцій розмірністю понад 128 можна умовно не враховувати затримання введення-виведення.

Розроблено та досліджено бібліотека ФБ для обчислення ряду матричних операцій, зокрема, операцій додавання матриць і множення матриці на скаляр. Для отримання найбільш об’єктивної експериментальної оцінки показників, що характеризують затримання обчислювального середовища на ПЛІС, досліджені операції, які реалізуються за один такт за довільної розмірності обчислювальної задачі. Але такі оцінки є достовірними для будь-яких обчислювальних операцій, оскільки досліджувані затримання виникають на етапах уведення-виведення даних та керування обчислювальним процесом.

3.3. Розроблення основ створення бібліотеки функціональних ядер, що забезпечує можливість ефективного варіювання зернистістю обчислень

3.3.1. Спосіб організації бібліотеки функціональних ядер. Для реалізації варіювання зернистістю обчислень у дисертаційній роботі запропоновано новий спосіб організації бібліотеки функціональних ядер, у межах якого для кожної функції запропоновано створення та зберігання в бібліотеці наборів ФБ з різними характеристиками продуктивності. Основні положення запропонованого способу наведено далі.

Як основний критерій для визначення продуктивності обчислювального середовища взято внутрішні затримання апаратного обчислювального середовища. Продуктивність ФБ визначається за виразом (3.32).

Важливим параметром з огляду на обмежені апаратні ресурси ПЛІС є збитковість устаткування ФБ для обчислення певної апаратної задачі; для оцінювання цього параметра в дисертаційній роботі використовується характеристика користувацької ефективності, яка визначена виразом (3.4).

Відповідно до запропонованого способу знаходження оптимального співвідношення обчислювального алгоритму та структури реконфігуровного обчислювального середовища на ПЛІС за виразом (3.33) визначається зернистість обчислювального середовища, що забезпечує оптимальні параметри відображення обчислювальних задач.

Узагальнену структуру бібліотеки функціональних ядер зображено на рис. 3.17. Узагальнену структуру інтерфейсу користувача і місце програмних засобів аналізу графу вхідної обчислювальної задачі з метою визначення ефективніших параметрів обчислювального середовища показано на рис. 3.18.

Для реалізації способу організації бібліотеки функціональних ядер у працях [132, 137] розроблено бібліотеку функціональних ядер на ПЛІС, яка включає набір ФБ на ПЛІС з оптимальними характеристиками для розв’язання задач лінійної алгебри та матричних операцій великої і надвеликої розмірності. Розроблений набір ФБ забезпечує запропонований спосіб варіювання зернистістю обчислень з метою підвищення ефективності паралельного обчислювального процесу. Місце бібліотеки функціональних ядер для розв’язання задач лінійної алгебри зі змінною зернистістю в загальній структурі бібліотеки функціональних ядер показано на рис. 3.17. У працях [172, 173] запропоновано метод та засоби для динамічного введення та аналізу графічної інформації в персональні електронно-обчислювальні машини, які використовуються для введення графів обчислювальних задач на абстрактний рівень обчислювальної системи.

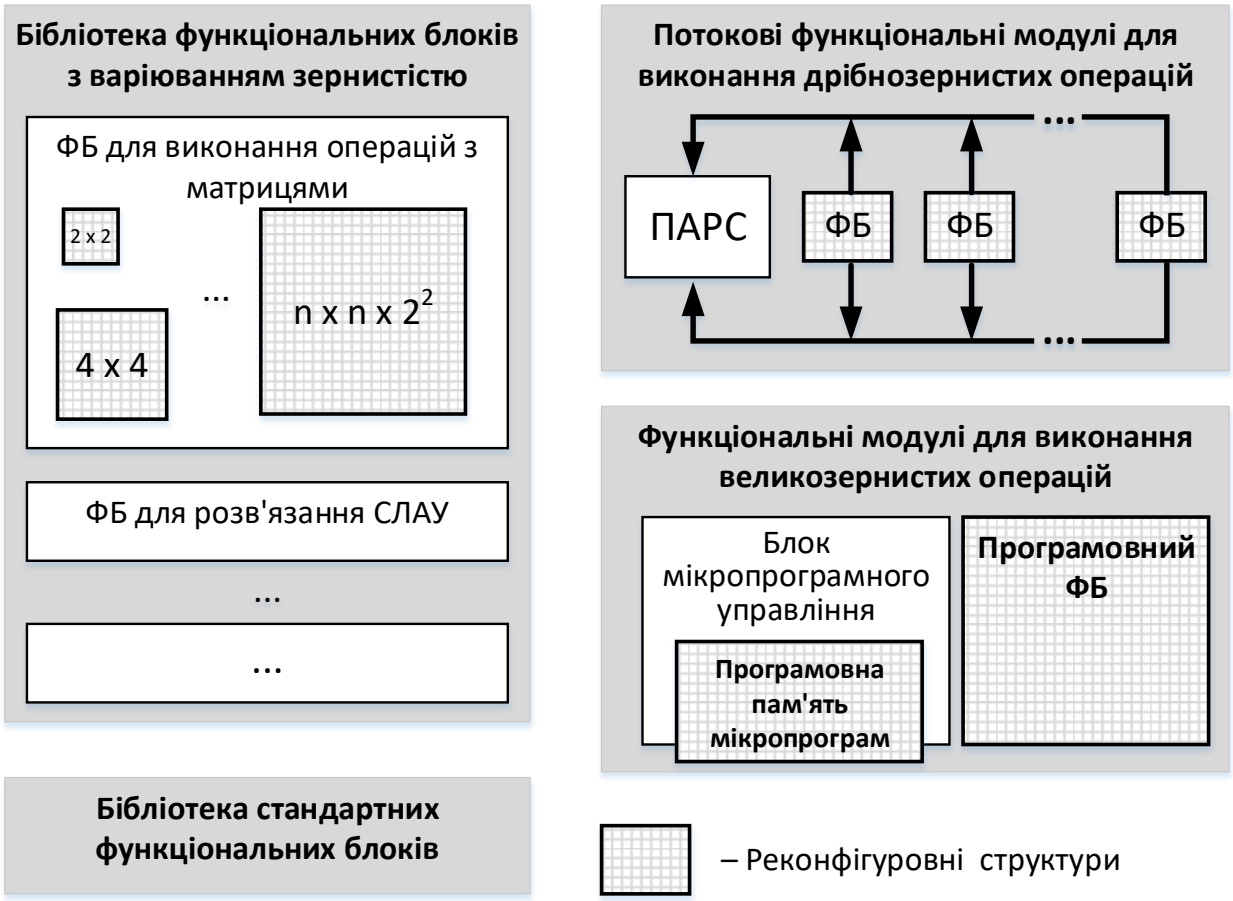


Рис. 3.17. Структура бібліотеки функціональних ядер

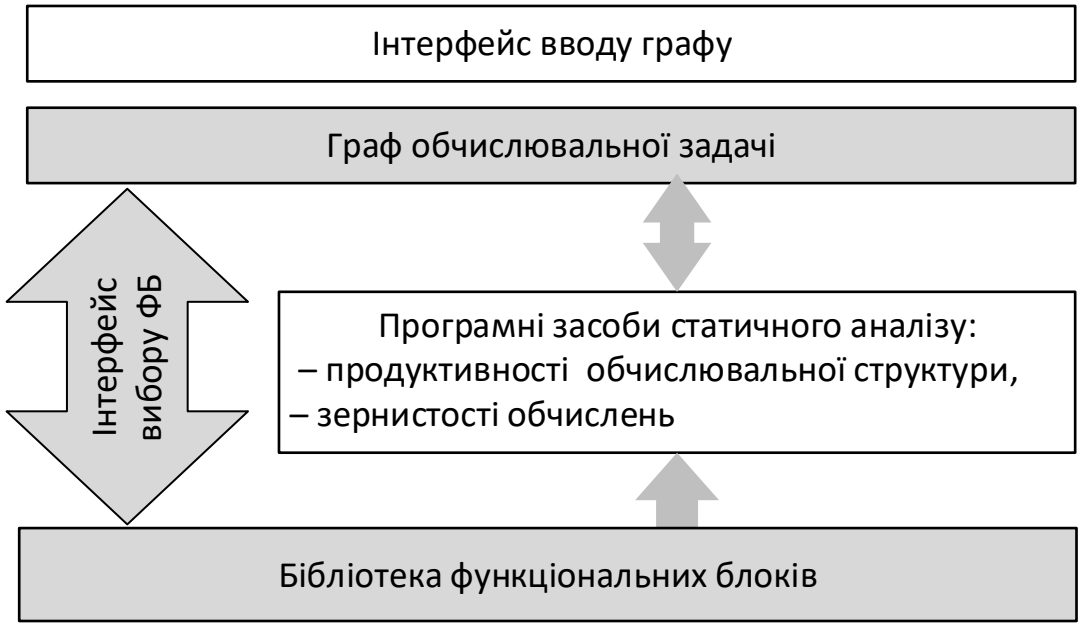


Рис. 3.18. Схема аналізу алгоритму обчислювальної задачі на базі структури бібліотеки функціональних ядер

Наведемо оцінку використаних ресурсів кристалів ПЛІС під час розміщення ФБ для виконання матричних операцій різної розмірності. Досліджено регулярні матричні структури для реалізації операцій додавання матриць і множення матриці на скаляр.

Для універсалізації бібліотеки функціональних ядер з огляду на її кінцевість розмірність блоків розраховується за виразом

$$m \times m = 2^p \times 2^p \mid p = 1,2,3,4,5,6,7,\ldots, \qquad (3.35)$$

де p – параметр, що залежить від цільового призначення комп'ютерної системи, обсягу обчислень, технологічних параметрів використовуваних сімейств мікросхем ПЛІС. На підставі теоретичної оцінки виразів (3.33) і (3.34) і досліджень параметрів ФБ було визначено, що використовувати ФБ апаратних обчислювачів з розмірністю понад 128 (p = 7) неефективно через тривалі внутрішні комунікаційні затримання на певних сімействах мікросхем. Хоча ефективність процесів блокового передавання вхідних та вихідних даних значно вища ніж з використанням ФБ меншої розмірності.

Оцінку обсягу використаних ресурсів кристалів ПЛІС для реалізації бібліотеки матричних функцій на кристалах *Altera* різних сімейств, які відрізняються обсягом логічних ресурсів і частотою виконання обчислень, наведено в табл. 3.3.

Зважаючи на однотипну природу організації обчислювального поля пристроїв, на яких проводились експерименти, використані ресурси для побудови обчислювальних блоків різної розмірності майже однакові, лише під час експериментів відзначається коливання в межах $\pm 10\%$, що спричинено певними налаштуваннями оптимізації розміщення конфігураційних даних, які виконує компілятор системи автоматизації проектування (САПР) під час синтезу цифрової схеми на ПЛІС.

Як видно з таблиці 3.3, розмірність розв'язуваної задачі безпосередньо впливає на розмір ФБ і відповідно на кількість блоків, розміщуваних на одному кристалі ПЛІС.

Таблиця 3.3

Оцінювання використаних ресурсів кристалів ПЛІС для реалізації функціональних блоків

Тип операції	Додавання матриць				
Розмірність матриць	128	64	32	16	8
	Усереднена кількість використаних логічних елементів (± 10%), (*total logic elements*)				
Логічних елементів	7681	3990	2013	1050	572
Сімейство ПЛІС	Частка загальної площі кристала ПЛІС, %				
*EP*4*CGX*15*BF*14*C*8	53,3	27,7	13,9	7,3	3,9
*EP*4*CGX*150*DF*31*C*8	5,1	2,7	1,3	0,7	0,4
*EP*4*CGX*110*DF*31*C*7	7	3,6	1,8	0,9	0,5
*EP*4*CGX*110*DF*27*C*7	7	3,6	1,8	0,9	0,5
*EP*4*CGX*30*CF*23*C*6	26,1	13,5	6,8	3,6	1,9
Тип операції	Множення матриці на скаляр				
Розмірність матриць		64	32	16	8
	Усереднена кількість використаних логічних елементів (± 10%), (*total logic elements*)				
Логічних елементів		11435	5775	2680	1378
Сімейство ПЛІС	Частка загальної площі кристала ПЛІС, %				
*EP*4*CGX*15*BF*14*C*8		79,4	40,1	18,6	9,6
*EP*4*CGX*150*DF*31*C*8		7,6	3,9	1,8	0,9
*EP*4*CGX*110*DF*31*C*7		10,4	5,3	2,4	1,3
*EP*4*CGX*110*DF*27*C*7		10,4	5,3	2,4	1,3
*EP*4*CGX*30*CF*23*C*6		38,8	19,6	9,1	4,7

У праці [90] визначено й обґрунтовано ефективність розв’язання динамічних задач планування та розподілу завдань на реконфігуровне обчислювальне середовище на локальному рівні обчислювального модуля. На

етапі попереднього статичного аналізу доцільно лише оцінювати продуктивність віртуального обчислювального середовища.

Для формальної оцінки ресурсів віртуального обчислювального середовища на ПЛІС запропоновано спосіб, що ґрунтується на характеристиках складових бібліотеки ФБ, які отримані експериментальним шляхом на етапі синтезу ФБ апаратних задач.

Для оцінювання максимальної кількості елементів матриці, яку можна розмістити на одному кристалі ПЛІС або в межах вільного обчислювального простору, що наявний на певному кристалі ПЛІС, запропоновано використання такого виразу

$$X \cdot n + Y = L_u, \tag{3.36}$$

де n – кількість елементів матриці, L_u – кількість використаних логічних елементів як одиниця вимірювання логічного простору обчислювальної поверхні ПЛІС (*total logic elements*); X і Y – коефіцієнти налаштування кристала ПЛІС для певного сімейства, на якому реалізовано бібліотеку функціональних ядер; Y – константа для обраного типу кристала ПЛІС; X – параметр, що змінюється під час експериментів у межах $\pm 10\%$ (табл. 3.3).

Далі наведено приклад визначення максимальної розмірності функціонального блока виконання операції додавання матриць для розміщення на кристалі сімейства *EP*4*CGX*15*BF*14*C*8. Значення X і Y знаходять шляхом розв'язання системи лінійних рівнянь на підставі результатів експериментів, наведених у табл. 3.3, для значень $n = 8$ і $n = 16$:

$$\begin{cases} x \times 8 + y = 572, \\ x \times 16 + y = 1050 \end{cases}$$

за розв’язання якої методом підстановки отримано $X = 59{,}75$; $Y = 94$.

Для повного обчислювального простору кристала сімейства *EP*4*CGX*15*BF*14*C*8 параметр L_u дорівнює загальній кількості логічних елементів – 14400. Визначимо максимальну розмірність оброблюваних матриць, використовуючи вираз (3.36):

$$n = \frac{L_u - Y}{X}, \text{ тоді } n = \frac{144000 - 94}{59{,}75} = 239.$$

Із розрахунків видно, що на даному кристалі ПЛІС можуть бути розміщені ФБ розмірністю 128 і менше.

Основні етапи виконання статичного аналізу продуктивності.

Етап 1. Вводиться ГА обчислювальної задачі. У працях [157, 161] описано програмні засоби для введення графів алгоритмів. У працях [172, 173] запропоновано засоби для автоматизації введення графів у комп'ютерну систему. Також для введення графів алгоритмів можуть бути використані відомі методи графічного введення, запропоновані у працях [51, 153].

Етап 2. Для кожної вершини графу на базі наявного набору функціональних ядер розраховується найбільш ефективна реалізація щодо відповідності розмірності обчислювальної задачі і зернистості обчислювального середовища. На цьому етапі може бути виконана кластеризація алгоритму обчислювальної задачі відповідно до методології, запропонованої у праці [174]. Для розрахунку оптимальної зернистості обчислювального середовища використовується запропонований вираз (3.33).

Етап 3. Розмірність функціональних блоків визначається таким чином:

$$f_{FB} = m / k,$$

де m – розмірність задачі; k – кількість ФБ визначена за виразом (3.33).

Етап 4. Аналізується користувацька ефективність ФБ за виразом (3.5), на підставі якого оцінюються збитковість обчислювальних ресурсів ФБ. Якщо ефективність менша ніж 50%, вибирається ФБ з меншою розмірністю відповідно до виразу (3.35). Схему коригування розмірності ФБ зображено на рис. 3.19.

3.3.2. Розроблення елементів бібліотеки функціональних ядер. На підставі досліджень залежності продуктивності обчислювального середовища та обсягу використаних ресурсів від технічних характеристик сімейств кристалів ПЛІС, результати яких наведено в табл. 3.3, визначено елементну

базу для створення елементів експериментальної бібліотеки функціональних ядер.

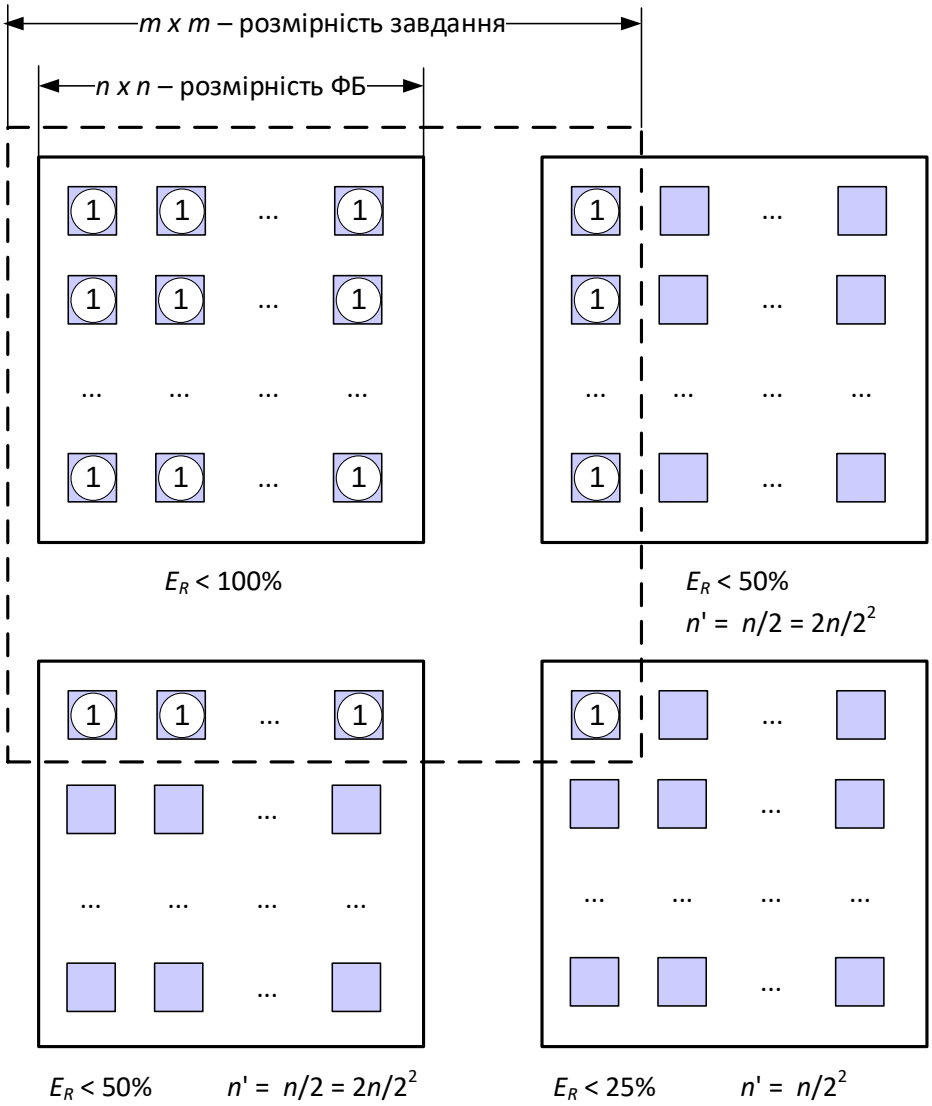


Рис. 3.19. Схема коригування розмірності ФБ апаратних задач: □ – ФБ («зерно» обчислень); ▢ –обчислювальний елемент; ⓝ – кількість операцій в обчислювальному елементі.

Для експериментів обрано сімейство мікросхем *Cyclone* IV *GX EP*4*CGX*15*BF*14*C*8 з такими характеристиками: загальна кількість логічних елементів – 14400; кількість виводів – 72.

Часові параметри моделювання роботи ФБ різної розмірності на кристалі *EP*4*CGX*15*BF*14*C*8 наведено в табл. 3.4.

Таблиця 3.4

Часові параметри моделювання роботи ФБ
на кристалі *EP*4*CGX*15*BF*14*C*8

Виконувана операція	Кількість вхідних даних, біт	Кількість елементів матриці	Час обчислення, такти
Підсумовування матриць	256	8	28
	512	16	52
	1024	32	100
	2048	64	196
	4096	128	388
Множення матриці на константу	160	8	22
	288	16	38
	544	32	70
	1056	64	134
	2080	128	262

Приклади часових діаграм моделювання роботи ФБ у середовищі САПР *Quartus* II *Altera* для виконання операцій додавання матриць (*matrix_plus_matrix*) і множення матриць на константу (*matrix_mult_const*) подано на рис. 3.20 і 3.21 відповідно. Розрядність слова даних – 16 біт.

Одиницею вимірювання часу є 1 такт роботи системи. Тривалість такту під час імітаційного моделювання залежить від таких факторів, як затримання пам'яті, сімейства використаних кристалів ПЛІС, ємності зовнішньої пам'яті даних. Результати імітаційного моделювання роботи ФБ наведено в табл. 3.4. На підставі результатів імітаційного моделювання з урахуванням сімейства кристалів ПЛІС побудовано графіки залежностей (див. рис. 3.13, 3.14, 3.15).

На числових діаграмах (рис. 3.20, 3.21) застосовуються наступні позначення:

i_data – вхідна шина даних;

o_data – вихідна шина даних;

fsm_idle – стан уведення та виведення даних;

fsm_load_size – стан установлення значення кількості елементів вхідної матриці;

*fsm_load_op*1 – стан завантаження елементів першої вхідної матриці;

*fsm_load_op*2 – стан завантаження елементів другої вхідної матриці (рис. 3.20);

fsm_load_c – стан завантаження значення константи (рис. 3.21);

fsm_calc – стан виконання обчислень (обчислення триває 1 такт);

s – кількість елементів матриці (число розрядністю 16 біт);

*m*0, *m*0*x*, *m*0*n* – елементи першої вхідної матриці, де *mx* – це *m*01, *m*02, ..., *m*0(*n* – 1) зображено у вигляді розриву на часових діаграмах (рис. 3.20, рис. 3.21);

*m*1, *m*1*x*, *m*1*n* – елементи другої вхідної матриці, де *mx* – це *m*11, *m*12, ..., *m*1(*n* – 1), зображено у вигляді розриву на часовій діаграмі (рис. 3.20);

c – константа розрядністю 16 біт, на яку множиться елементи матриці (рис. 3.21);

*o*0, *ox*, *on* – елементи результуючої матриці.

Машина станів виконує функцію керування обчисленнями і реалізована як статична частина кожного ФБ. Але машина станів завантажується на ПЛІС разом з ФБ різної розмірності, які визначені як динамічна частина ФБ.

Приклад функціональних схем ФБ для виконання операцій додавання матриць та множення матриці на константу наведено в додатку Б. Результати вимірювань часових характеристик ФБ виконання матричних операцій різної розмірності, отримані під час їх розроблення та експериментальних досліджень – у додатку А. Код програми реалізації функціонального блоку змінюваної розмірності для виконання операції підсумовування матриць наведено в додатку В.

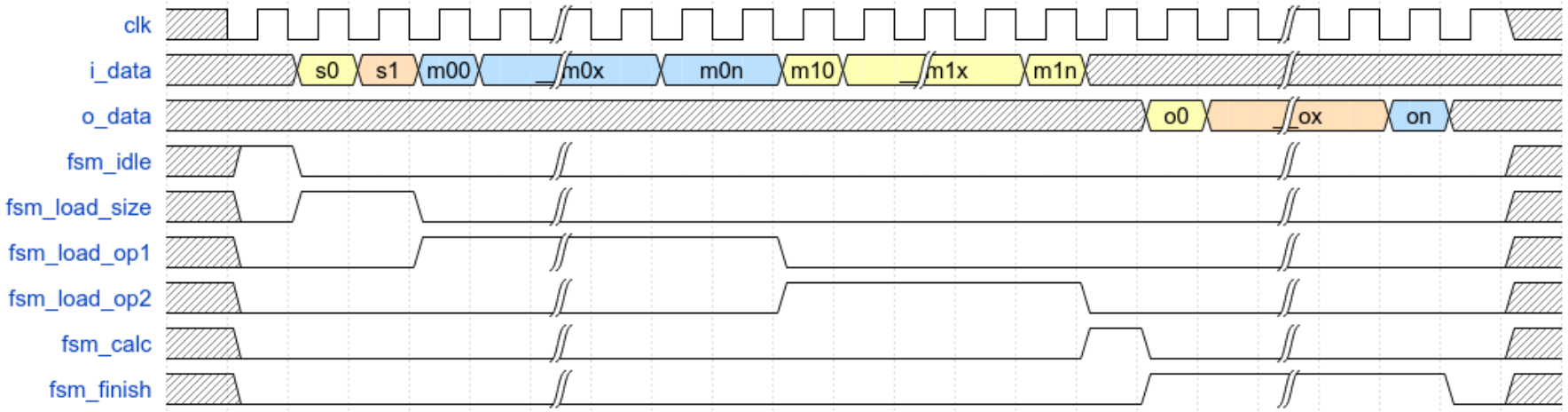


Рис. 3.20. Часова діаграма моделювання роботи блока додавання матриць розмірності 16 × 16

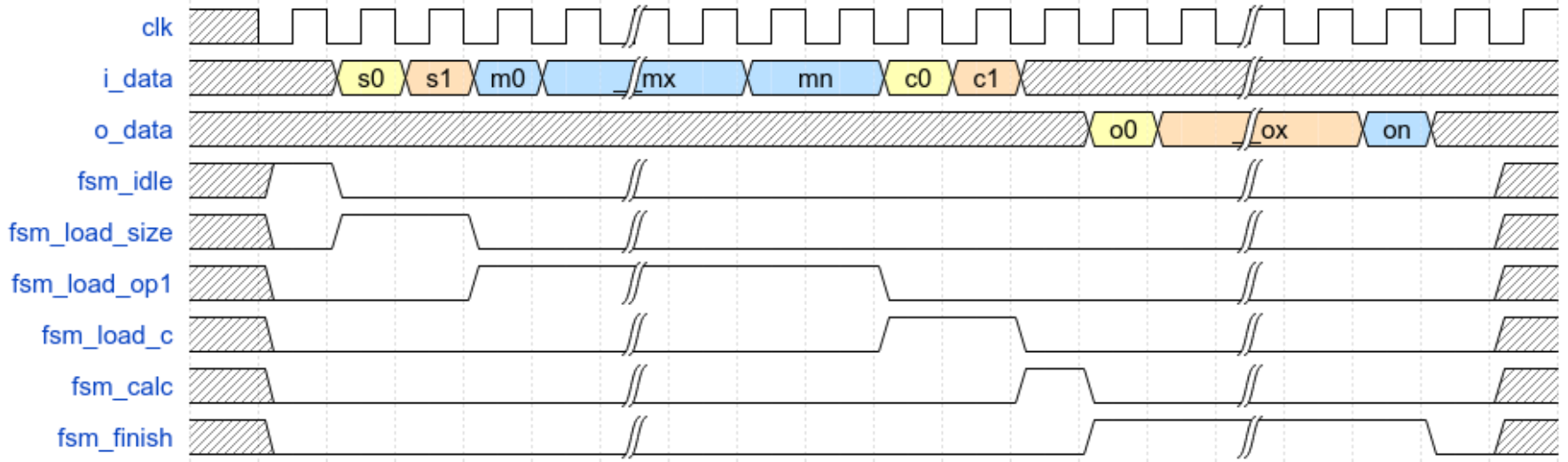


Рис. 3.21. Часова діаграма моделювання роботи блока множення матриці 16 × 16 на константу

ВИСНОВКИ ДО РОЗДІЛУ 3

1. Розроблені математичні моделі функціональних процесів дозволили оцінити непродуктивні витрати часу на всіх рівнях функціонування динамічно реконфігуровних комп'ютерних систем. Визначено, що адаптивне відображення завдань забезпечує інтенсивне скорочення непродуктивного часу за певного співвідношення параметрів обчислювальних алгоритмів і структури обчислювального середовища. Інакше непродуктивний час збільшується неконтрольованим чином, що в цілому зменшує ефективність апаратного пришвидшення.

2. Розроблено інтегральний критерій оптимізації, що дозволяє розв'язувати багатокритеріальну задачу оптимізації процесу обробки інформації на всіх рівнях функціонування динамічно реконфігуровних комп'ютерних систем.

3. Інтегральний критерій оптимізації ґрунтується на визначенні співвідношення параметрів обчислювальних задач і реконфігуровного обчислювального середовища, співвідношення непродуктивного і продуктивного часу виконання завдань, співвідношення параметрів обчислювальних задач і продуктивності реконфігуровного обчислювального середовища з урахуванням часових і апаратурних обмежень процесу обробки інформації.

4. Модифікований спосіб визначення непродуктивного часу дозволив визначити й оцінити непродуктивні витрати часу в процесі відображення завдань на реконфігуровне обчислювальне середовище з метою їх оптимізації. Формалізація модифікованого способу визначення непродуктивного часу дозволила виявити похибки відомого способу під час розподілення часу, що збільшило точність розрахунків непродуктивного часу.

5. Запропонований новий метод оптимізації процесу відображення задач на реконфігуровне обчислювальне середовище ґрунтується на визначенні стратегії обслуговування кожного завдання на підставі аналізу показника його апаратного пришвидшення із забезпеченням інтегрального критерію

оптимізації. Різні стратегії обслуговування завдань базуються на варіюванні непродуктивним часом і надлишковим використанням апаратурних ресурсів реконфігуровної комп'ютерної системи на підставі багаторівневого кешування конфігураційних даних. Застосування запропонованого методу загалом сприяє підвищенню користувацької ефективності РКС і зменшенню кількості відхилень виконання завдань у процесі їх динамічного надходження.

6. Із практичної точки зору метод оптимізації процесу відображення обчислювальних задач на реконфігуровне обчислювальне середовище забезпечує широкі класи задач керування ефективною цільовою обчислювальною структурою, зокрема, на час виконання яких накладаються жорсткі часові обмеження. Метод може бути використаний для підвищення ефективності високопродуктивних реконфігуровних обчислювальних систем під час розв'язання задач керування різноманітними технічними і технологічними процесами, а також для реалізації багатовимірних обчислень у складних інформаційних системах, побудованих на базі ПЛІС.

7. Визначено критерій швидкодії паралельного обчислювального середовища на ПЛІС, який ґрунтується на співвідношенні часу обчислення задачі на апаратурі ПЛІС, обсязі передаваних корисних даних, з урахуванням просторових обмежень обчислювального середовища і затримань поширення сигналів на фізичному рівні функціональних блоків, які лінійним чином збільшуються зі збільшенням «зерна» обчислень. Це дозволяє оцінити ефективність обробки інформації на фізичному рівні реконфігуровних обчислювальних систем у процесі розв'язання задач великої розмірності.

8. Для кількісної оцінки розробленого інтегрального критерію запропоновано нову стратегію організації процесу обробки інформації в РКС, яка заснована на взаємній адаптації розв'язуваних задач і обчислювального середовища, побудованого на ПЛІС. Реалізація нової стратегії базується на визначенні оптимальної зернистості обчислень. Оптимальна зернистість забезпечує екстремуми цільових функцій оптимізації процесу обробки інформації.

9. За результатами експериментальних досліджень застосування нової стратегії відображення завдань на реконфігуровне обчислювальне середовище дозволило підвищити користувацьку ефективність реконфігуровних комп’ютерних систем у середньому на 10 %.

10. Запропонована та реалізована на ПЛІС бібліотека функціональних ядер для розв’язання задач лінійної алгебри та матричних операцій дозволила забезпечити набір функціональних блоків з оптимальними характеристиками, що забезпечує можливість ефективного варіювання зернистістю обчислень згідно з визначеними в роботі критеріїв швидкодії паралельного обчислювального середовища на ПЛІС.

РОЗДІЛ 4
УДОСКОНАЛЕННЯ СТРУКТУРНОГО РІВНЯ РЕКОНФІГУРОВНИХ КОМП'ЮТЕРНИХ СИСТЕМ НА БАЗІ ПЛІС

4.1. Розроблення концептуальних принципів удосконалення структурного рівня динамічно реконфігуровних комп'ютерних систем

4.1.1. Розроблення та обґрунтування нових рівнів абстракцій динамічно реконфігуровних комп'ютерних систем. Визначення рівнів абстракцій, на яких зазвичай розглядають архітектуру обчислювальних систем, наведено у праці [51].

На підставі аналізу особливостей функціонування та концептуальних принципів організації керувальних засобів реконфігуровних обчислювальних систем, який виконано у праці [90], а також їх сучасної класифікації, розробленої у праці [20], визначено рівні абстракції, на яких розглядаються засоби керування паралельними обчисленнями. Відомі рівні абстракції розгляду керувальних засобів у РКС показано на рис. 4.1.

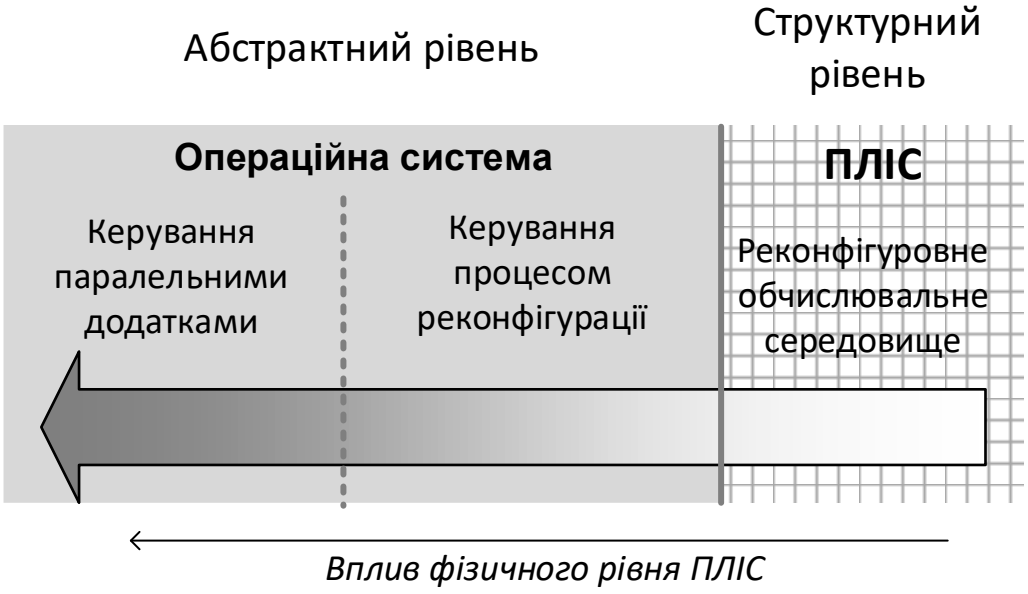


Рис. 4.1. Рівні абстракцій розгляду засобів керування в відомих реконфігуровних комп'ютерних системах

Реалізація керувальних засобів на рівні надбудови операційної системи не дає змогу використовувати ефективні технології паралельного програмування, ускладнює урахування фізичних параметрів ПЛІС, характеризуються високою спеціалізацією і значними накладними витратами. Із цієї причини відомі рівні абстракцій розгляду засобів керування обчислювальним процесом неефективні в динамічно РКС.

Для ефективної організації динамічного відображення задач та керування реконфігуровними обчислювальними ресурсами запропоновано новий підхід до розгляду засобів керування обчисленнями на трьох рівнях абстракцій (рис. 4.2).

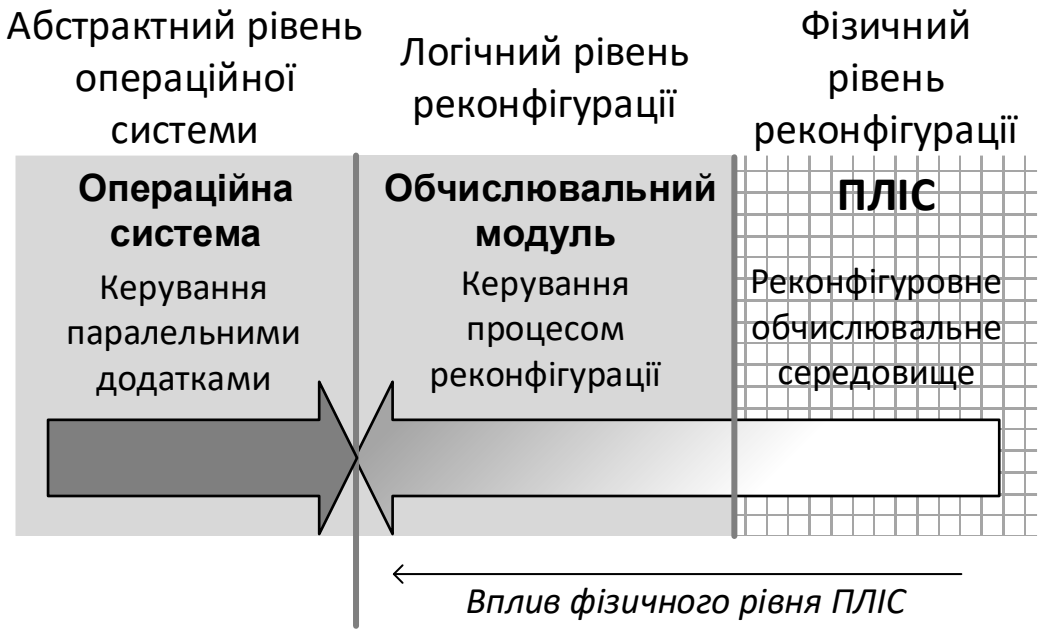


Рис. 4.2. Нові рівні абстракції розгляду засобів керування реконфігурацією обчислювального середовища у РКС

1. **Абстрактний рівень операційної системи**, на якому реалізовується загальноприйнятий підхід до керування паралельними обчисленнями програмними засобами операційної системи. Реконфігуровна обчислювальна структура на ПЛІС розглядається як віртуальна обчислювальна структура з визначеними характеристиками і можливостями.

2. **Логічний рівень реконфігурації**, на якому реалізовуються логічна послідовність реконфігурації програмно-апаратними засобами на рівні локальних процесорів або апаратними засобами на рівні апаратури обчислювального модуля.

3. **Фізичний рівень реконфігурації**, на якому реалізовується фізична послідовність реконфігурації – це апаратна реалізація фізичного процесу налаштування обчислювальних структур на ПЛІС на рівні апаратури обчислювального модуля.

У праці [90] оцінено ефективність нових рівнів абстракції аналогічним способом, який був використаний для оцінювання існуючих рівнів абстракції розгляду керувальних засобів у РКС, а саме: оцінювання I та оцінювання II (див. підрозділ 1.3.4 дисертаційної роботи).

Оцінювання III. Для розподілення за рівнями вибрано межу між логічним та фізичним процесами реконфігурації. Для реалізації керування реконфігурацією забезпечується можливість використання спеціалізованих засобів, які абстраговані від задач операційної системи з підтримання загальносистемного керування. Видимими об’єктами на локальному рівні керування є логічна структура локальної реконфігуровної ділянки. Реалізація фізичного процесу реконфігурації покладається на апаратний рівень обчислювального модуля, що є максимально наближеним до фізичного рівня обчислювального середовища на ПЛІС.

Із діаграми на рис. 4.3 видно, що апаратна реалізація зумовлює значне збільшення ефективності. Апаратна реалізація логічної послідовності реконфігурації зменшує навантаження на локальний процесор і пришвидшує реалізацію логічних алгоритмів керування. Локалізація процесу керування логічною послідовністю реконфігурації звужує їх специфічність, що певною мірою спрощує їх розроблення і застосування. У цьому контексті реалізація керувальних засобів на апаратному рівні дасть максимальний ефект від їх застосування.

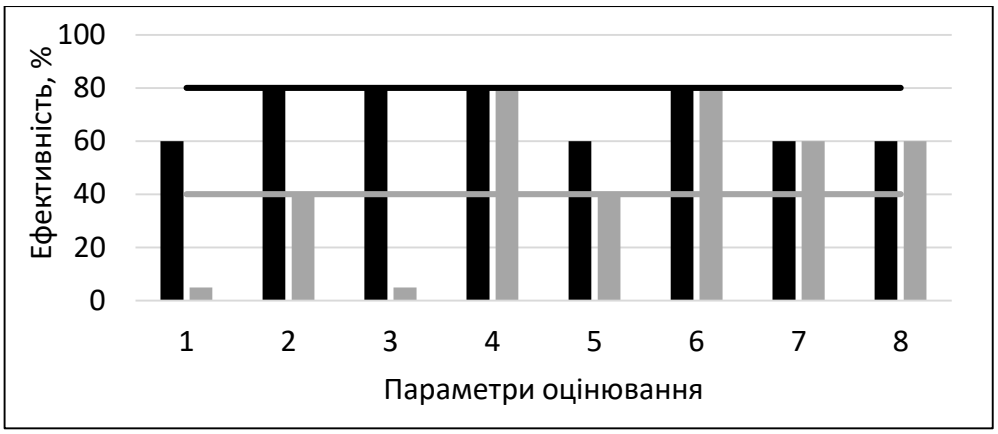


Рис. 4.3. Оцінювання ефективності реалізації керувальних засобів на нових рівнях абстракції розгляду РКС: ▒ – програмна реалізація; ▒ – середнє значення; ■ – апаратна реалізація; ■ – середнє значення

Діаграма порівняльного оцінювання ефективності реалізації засобів керування обробкою даних на нових і на існуючих рівнях абстракції показано на рис. 4.4. Для порівняння використано усереднені значення ефективності.

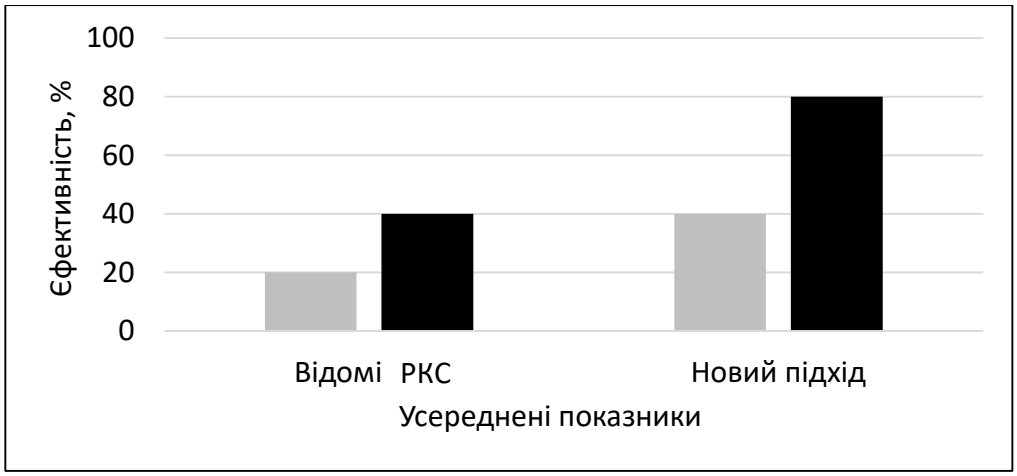


Рис. 4.4. Порівняльне оцінювання способів реалізації засобів керування в РКС: ▒ – програмна реалізація; ■ – апаратна реалізація

За абстрактною шкалою оцінювання ефективності, обґрунтованої у праці [90], видно, що реалізація додаткового рівня абстракції для розгляду логічного процесу реконфігурації забезпечує найбільш ефективні вирішення проблем підвищення ефективності функціонування РКС, пов’язані з організацією динамічної реконфігурації обчислювального середовища.

Розгляд засобів керування реконфігурацією обчислювального середовища на нових рівнях абстракцій лежить в основі нового концептуального принципу вдосконалення структурного рівня динамічно РКС, що розроблено й обґрунтовано у працях [90, 174]. Основним положенням запропонованої концепції є децентралізація процесу керування реконфігурацією обчислювального середовища та локалізація засобів керування реконфігурацією на рівні обчислювального модуля. Концепція побудови архітектури РКС на основі нових рівнів абстракції дає змогу локалізувати специфічні процеси на відповідних рівнях абстракції і реалізовувати на кожному рівні найбільш ефективні засоби організації обчислень.

4.1.2. Основні положення методології відображення обчислювальних задач на багаторівневу структуру реконфігуровних комп'ютерних систем. Властивості нових рівнів абстракції розгляду засобів керування обчислювальним процесом та реконфігуровними обчислювальними ресурсами [90] обґрунтовують певну подібність організації засобів керування на всіх рівнях абстракцій. Це дає змогу застосовувати методи та засоби, запропоновані в даній дисертаційній роботі, на всіх рівнях абстракції розгляду РКС. На цій підставі у праці [174] розглянуто розширення функціональних можливостей РКС через розв'язання задач зі змішаним типом паралелізму, у тому числі таких, що мають самоподібну структуру, наприклад, циклічні та фрактальні обчислення, а також задач, що включають високу кількість повторів однотипних функцій.

Граф відображення обчислювальних задач на багаторівневу структуру РКС, що ґрунтується на нових рівнях абстракції зображено на рис. 4.5. У праці [174] розроблено методологію відображення задач на багаторівневу структуру РКС, згідно з якою процес відображення паралельного алгоритму обчислювальної задачі поділяється на віртуально подібні етапи. Формалізовано також віртуальний підхід до відображення задач на всіх рівнях

абстракції розгляду РКС [174]. Етапи відображення алгоритму обчислювальної задачі зі змішаним типом паралелізму на багаторівневу структуру РКС показано на рис. 4.6.

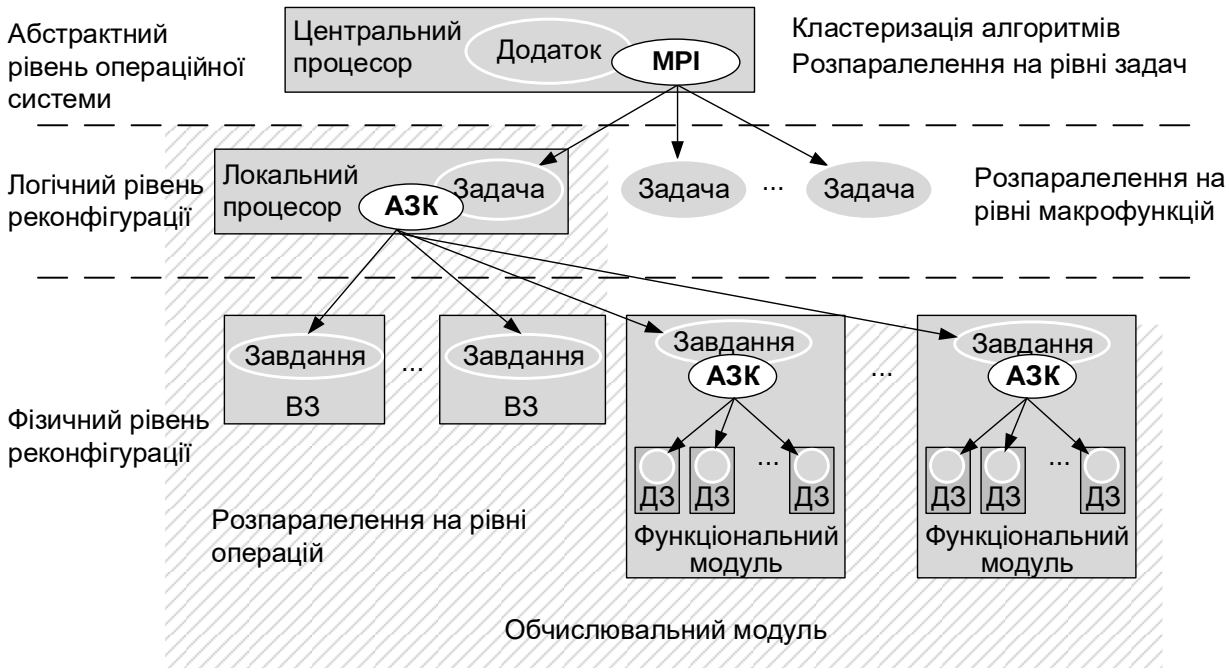


Рис. 4.5. Граф відображення задач на багаторівневу структуру РКС: *MPI* (*Message Passing Interface*) – традиційні засоби розпаралелення обчислювальних задач; АЗК – автоматичні засоби керування обробкою інформації; ВЗ – великозернисті ФБ; ДЗ – дрібнозернисті ФБ

Обчислювальні задачі на абстрактному рівні операційної системи розподіляються за критеріями мінімізації часу обчислення. Основний критерій розподілу завдань за рівнями – це реалізація мінімальної кількості зв’язків між рівнями і мінімальної кількості рівнів [156]. На наступному етапі (кластеризації) формується деяка множина підграфів – кластерів, реалізація яких згодом покладається на окремий обчислювальний модуль. Кластеризація виконується за критеріями мінімізації зв’язків між кластерами, що в сукупності з ефектом від першого етапу мінімізує час виконання обчислень через зменшення кількості міжмодульних зв’язків, внутрішньомодульних зв’язків, послідовно виконуваної реконфігурації обчислювального

середовища. Для реалізації розглянутих етапів можна використовувати відомі алгоритми і технології паралельного програмування [174], наприклад, «жадний» алгоритм розподілу завдань за рівнями, алгоритми кластеризації з максимальною кількістю зв’язків усередині кластера, метод пошуку мінімального перерізу графу, алгоритми пошуку ізоморфних структур.

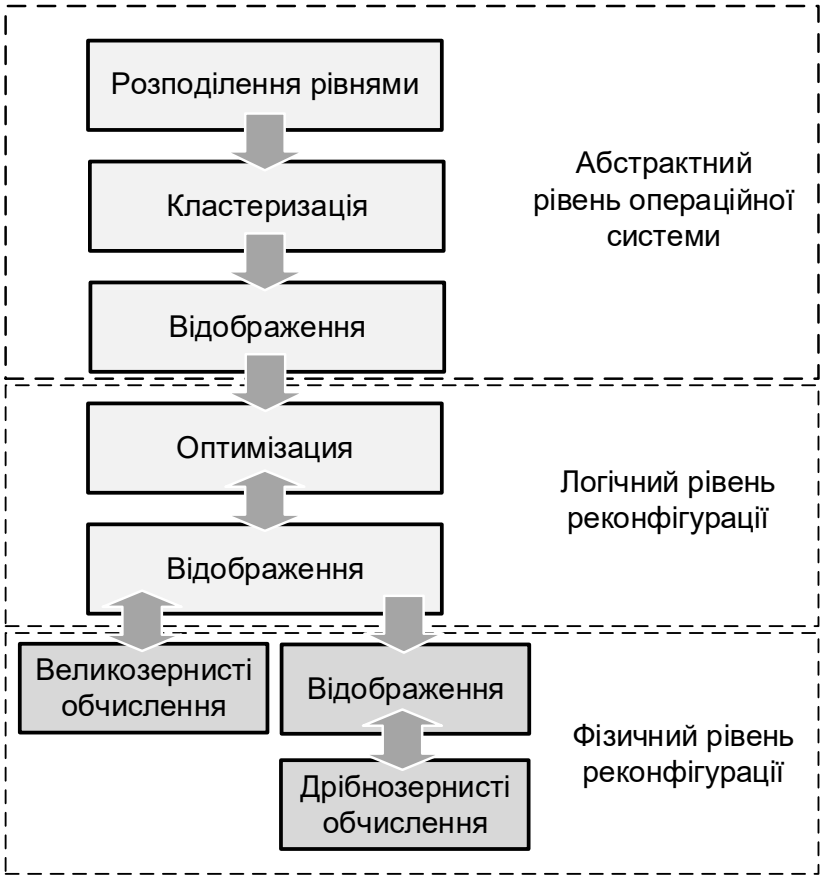


Рис. 4.6. Основні етапи відображення задач на архітектуру реконфігуровної комп’ютерної системи

Приклад реалізації абстрактного рівня для додатків зі змішаним типом паралелізму, поданих макрографами обчислювальних алгоритмів, показано на рис. 4.7, *а*. Приклад розподілу фрактального графу обчислювального алгоритму на рівні самоподібної архітектури реконфігуровної комп’ютерної системи показано на рис. 4.7, *б*.

Для оптимізації структури макрографів обчислювальних задач на абстрактному рівні використовується метод модифікації структури графів,

поданих в ЯПФ [155]. Цей метод ґрунтується на перерозподілі завдань між ярусами макрографу обчислювальної задачі, поданого в ЯПФ, що дає змогу мінімізувати час виконання обчислень в обмежених віртуальних ресурсах обчислювального модуля [174].

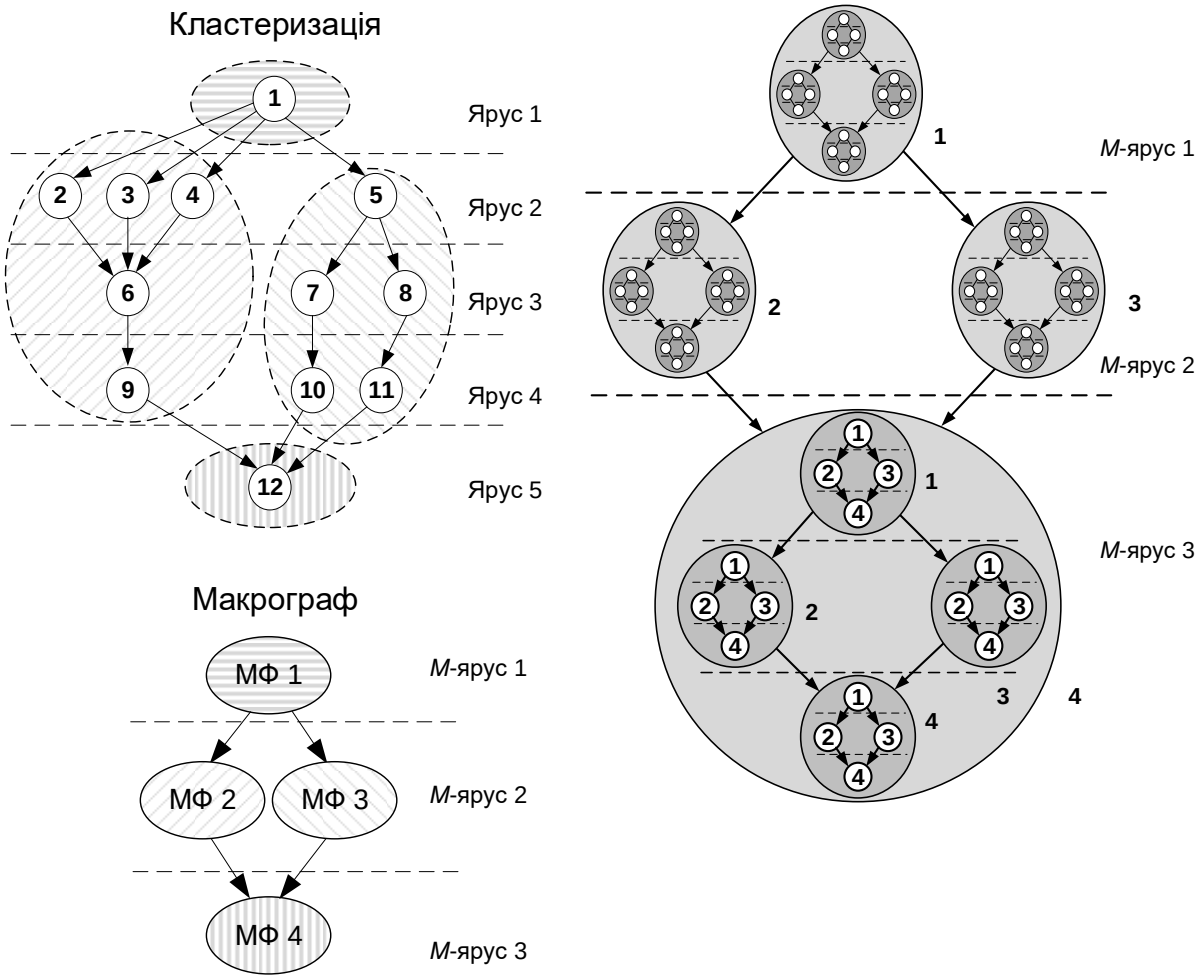


Рис. 4.7. Відображення графів обчислювальних алгоритмів на самоподібну архітектуру реконфігуровної обчислювальної системи: *а* – граф обчислювальної задачі зі змішаним типом паралелізму; *б* – фрактальний граф; *M*-ярус – ярус ЯПФ графу; МФ – макрофункція

Актуальним напрямом використання ПЛІС для організації високопродуктивних обчислень є хмарні обчислення на основі розподілених обчислювальних систем та мереж [175, 176]. У працях [177, 178] розглянуто розширення застосування реконфігуровних обчислювальних систем через

реалізацію багаторівневого відображення обчислювальних задач на віртуальну обчислювальну структуру розподілених обчислювальних мереж. У праці [177] визначено основні вимоги до методів та засобів підвищення ефективності технологій GRID за рахунок реалізації динамічних паралельних обчислень на основі розподілених неоднорідних архітектур на ПЛІС. У праці [178] запропоновано спосіб удосконалення технології GRID через реалізацію розподілених неоднорідних архітектур на ПЛІС для підвищення ефективності паралельних обчислень у класах задач з високою кількістю інформаційних обмінів.

4.2. Спосіб організації віртуальної пам’яті в динамічно реконфігуровних комп’ютерних системах на базі ПЛІС

4.2.1. Удосконалення структури обчислювального модуля динамічно реконфігуровних комп’ютерних систем. Загальний підхід до побудови архітектури РКС, обґрунтований на підставі класифікації реконфігуровних обчислювальних систем [20], зображено на рис. 1.6.

На підставі огляду особливостей структурної організації РКС (див. рис. 1.2, 1.6 і 1.7) наведемо структуру загальноприйнятої моделі обчислювального модуля (рис. 4.8), у склад якого зазвичай входять керувальний процесор, енергонезалежна пам’ять для зберігання локальних програм і даних та реконфігуровний модуль, реалізований на ПЛІС.

У праці [158] удосконалено структуру обчислювального модуля РКС. На відміну від відомої моделі реконфігуровного обчислювального модуля для розділення логічного і фізичного процесів реконфігурації і розвантаження керувального процесора застосовується спеціальний контролер реконфігурації, зображений на рис. 4.9. Контролер реконфігурації забезпечує реалізацію фізичного процесу реконфігурації обчислювального середовища на ПЛІС, зокрема, завчасної реконфігурації в процесі адаптивного відображення задач на реконфігуровне обчислювальне середовище (2.36), (2.37).

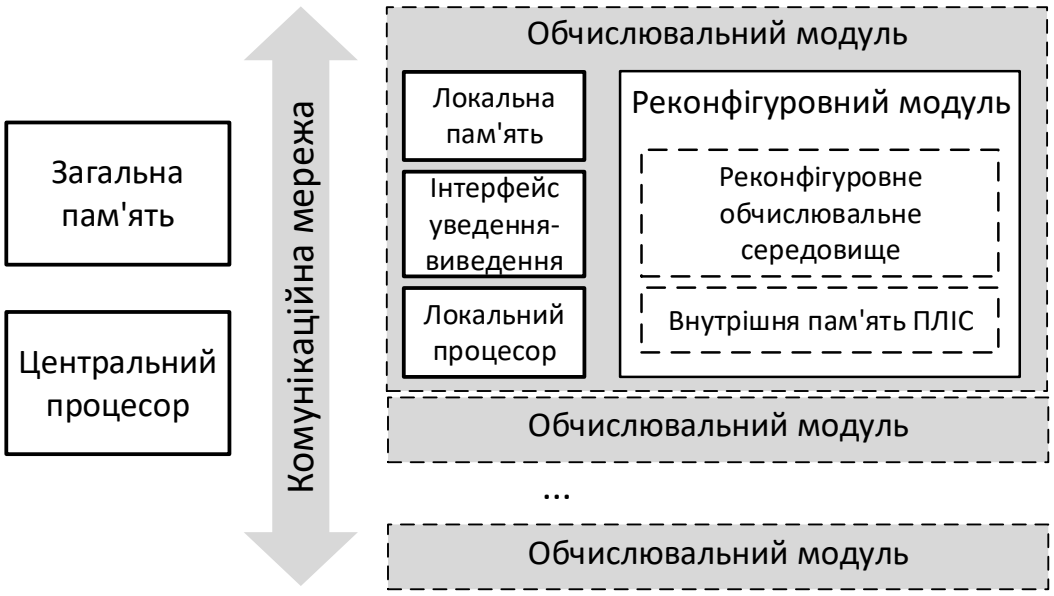


Рис. 4.8. Відома модель обчислювального модуля РКС

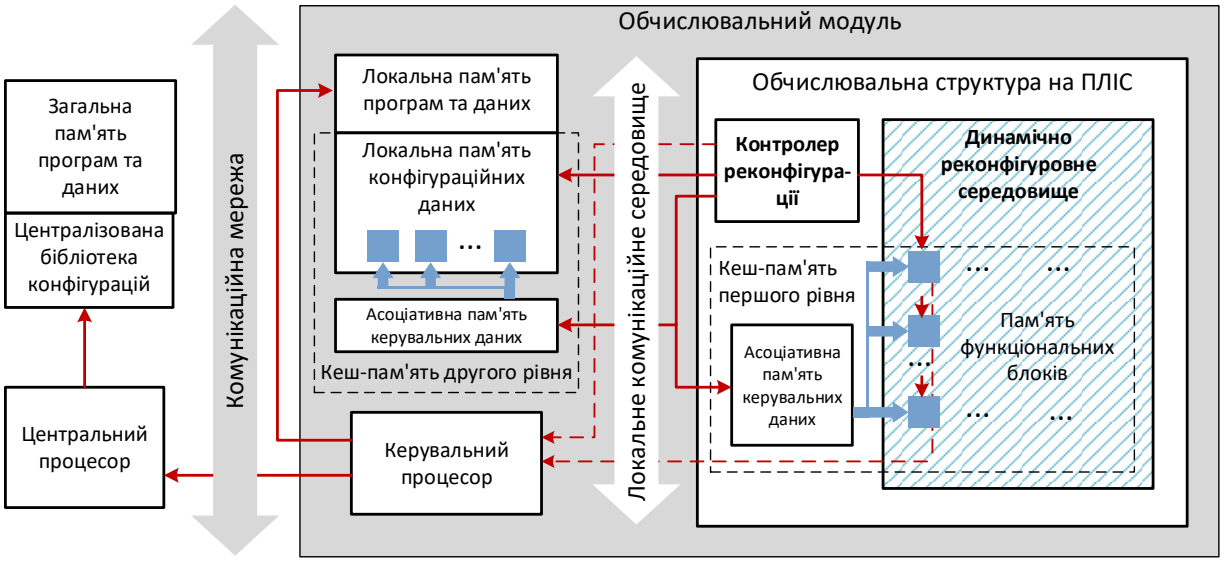


Рис. 4.9. Вдосконалена модель обчислювального модуля РКС

У РКС доступ до централізованих сховищ конфігураційних даних виконується засобами мережевих комунікаційних середовищ із залученням функціонального прошарку мережевого програмного забезпечення і ресурсів централізованої операційної системи (рис. 4.8). Математичний вираз (2.24) обґрунтовує, що первинне завантаження конфігураційних даних із централізованої бібліотеки значно уповільнює обчислювальний процес значними затриманнями під час передавання конфігураційних даних.

Для скорочення критичного часу розв’язання обчислювальних задач у праці [158] запропоновано спосіб багаторівневого кешування конфігураційних даних для динамічно РКС на основі багаторівневої багатофункціональної віртуальної пам’яті, яка розміщується між основною системною пам’яттю та внутрішньомодульними засобами керування реконфігурацією (рис. 4.9).

Основне призначення віртуальної пам’яті полягає в забезпеченні тимчасових сховищ конфігураційних даних у локальному запам’ятовувальному просторі обчислювального модуля, запобігаючи непродуктивним зверненням до централізованих сховищ конфігураційних даних.

Віртуальна пам’ять складається з двох рівнів. Кожний з цих рівнів має додатковий прошарок для зберігання керувальної інформації. Для реалізації віртуальної пам’яті запропоновано використання традиційних технологій кеш-пам’яті з асоціативним доступом, що забезпечує високу швидкість і простоту доступу до конфігураційних даних і керувальної інформації з використанням апаратних ресурсів ПЛІС малого обсягу.

Кешування даних на двох рівнях пам’яті забезпечує виконання різних стратегій пришвидшення реконфігурації обчислювального середовища, запобігаючи повторному завантаженню конфігураційних даних апаратних задач (2.25) – (2.30).

Для реалізації найбільш швидкодійного сховища запропоновано використання поверхні динамічної ділянки ПЛІС, де зберігаються ФБ апаратних задач у вже сконфігурованому вигляді. Це унеможливлює повторне передавання конфігураційних даних і забезпечує зберігання ресурсів внутрішньої пам’яті ПЛІС. На підставі цього реалізовано математичну модель послідовності завантаження ІІІ, що описується виразом (2.19). У статичній частині реконфігуровної області ПЛІС розміщується асоціативна пам’ять керувальних даних. Ця частина адресного простору обчислювального модуля являє собою кеш-пам’ять першого рівня.

Конфігурації апаратних задач зберігаються на поверхні реконфігуровної області за певним розкладом. Пам’ять керувальних даних використовується для обліку керувальної інформації і підтримання розкладу реконфігурації ФБ. Фактично пам’ять керувальних даних містить покажчики на ФБ (адресну інформацію), що зберігаються в динамічно реконфігуровній області ПЛІС, та інформацію щодо їх поточного стану.

Для реалізації послідовності завантаження ІІ відповідно до виразу (2.18) у межах способу багаторівневого кешування конфігураційних даних використовується другий рівень кеш-пам’яті, який фізично являє собою частину зовнішньої локальної пам’яті обчислювального модуля. Ця пам’ять не така критична до обсягу використання як внутрішня пам’ять ПЛІС, але на підставі порівняння виразів (2.29) і (2.30) спричиняє незначні накладні витрати під час реконфігурації обчислювального середовища за умови високого показника пришвидшення апаратного виконання завдання (2.12). Багаторівнева кеш-пам’ять використовується для реалізації різних стратегій обслуговування завдань під час застосування методу оптимізації відображення обчислювальних задач на реконфігуровне обчислювальне середовище на ПЛІС [165, 166].

Кеш-пам’ять другого рівня на основі локальної пам’яті обчислювального модуля також застосовується для тимчасового зберігання вивантажених з поверхні ПЛІС конфігурацій апаратних задач для продовження терміну їх зберігання на рівні обчислювального модуля. Пам’ять керувальних даних другого рівня кеш-пам’яті містить покажчики на конфігураційні файли апаратних задач та їх поточний стан.

4.2.2. Удосконалення моделі процесу обробки інформації в динамічно реконфігуровних комп’ютерних системах. Основний цикл виконання завдання на основі вдосконаленої структури обчислювального модуля РКС складається з п’яти етапів:

Етап 1. Програма-менеджер керування обчисленнями, що реалізовується керувальним процесором обчислювального модуля, виконує аналіз графу обчислювального алгоритму, на підставі якого формується розклад виконання обчислень. На підставі створеного розкладу визначається чергове завдання, що готове до виконання, за умови завершення процесу реконфігурації обчислювального середовища для його апаратного обчислення. Керувальний процесор відповідно до готового завдання формує макрокоманду, формат якої містить поле типу операції і адресну інформацію щодо вхідних даних до обчислення. На підставі типу операції керувальний процесор формує виконавчу адресу пошуку конфігураційних файлів у пам'яті і передає керування процесом реконфігурації контролеру реконфігурації.

Етап 2. Керувальний процесор виконує аналіз нащадків завдання, що надійшло до виконання, на підставі якого оновлює чергу реконфігурації і повертається на розклад виконання обчислень (на етап 1). Пошук конфігураційних даних виконується контролером реконфігурації на етапах 3 – 5.

Етап 3. Контролер реконфігурації на підставі сигналу наявності завдань у черзі реконфігурації одночасно формує виконавчу адресу на входи кеш-пам'яті першого і другого рівнів, на підставі чого виконується пошук конфігураційних даних ФБ. У випадку «успішного звернення» контролер реконфігурації ініціює процедуру розміщення і запуску апаратної задачі у ФБ на ПЛІС. Перехід на етап 4.

Етап 4. У випадку «неуспішного звернення» контролер реконфігурації формує сигнал переривання до керувального процесора, який за отриманою виконавчою адресою ініціює процедуру завантаження конфігураційних даних із централізованої БКД. Перехід за відповідним сигналом переривання на етап 4.

Етап 5. Контролер реконфігурації керує безпосередньо процесом прошивання ПЛІС і повертається на етап 3.

Модель стандартного процесу послідовного обробки інформації, що властива відомим реконфігуровним обчислювальним системам, зображено на рис. 4.10. На відміну від відомої моделі запропоновані засоби вдосконалення структури обчислювального модуля (див. рис. 4.9) дають змогу пришвидшити процес обробки інформації розпаралеленням послідовного процесу керування обчисленнями з урахуванням реконфігурації обчислювального середовища, як зображено на рис. 4.11. Програма-менеджер реконфігурації, що реалізується керувальним процесором, виконує аналіз графу обчислювальної задачі та формування черги завчасної реконфігурації, у той час як контролер реконфігурації виконує завчасну реконфігурацію обчислювального середовища на ПЛІС.

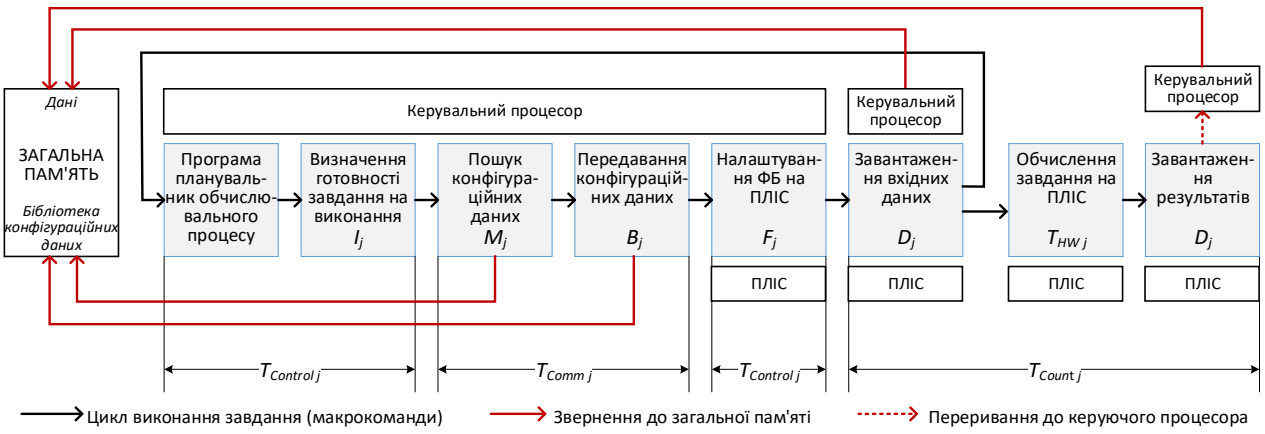


Рис. 4.10. Модель стандартного процесу обробки інформації в реконфігуровних комп’ютерних системах

Контролер реконфігурації обслуговує чергу завчасної реконфігурації незалежно від послідовності функціонування керувального процесора. Контролер реконфігурації визначає місце знаходження конфігураційних даних поточної апаратної задачі і розміщує конфігурацію на поверхні реконфігуровної області ПЛІС за певними алгоритмами просторового розміщення. Місце зберігання конфігураційних даних визначає керувальний процесор на підставі аналізу показника пришвидшення кожної апаратної задачі відповідно до виразу (2.12).

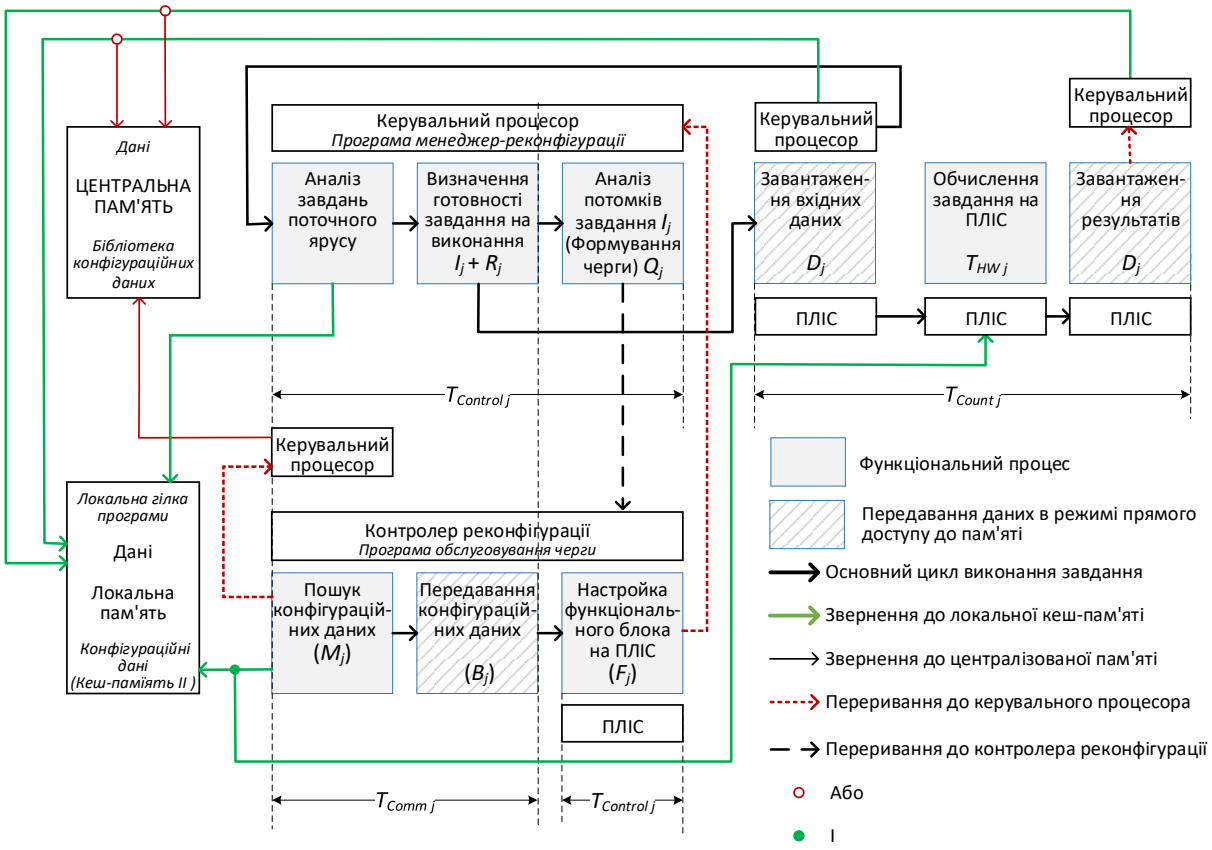


Рис. 4.11. Модель удосконаленого процесу обробки інформації на основі запропонованих програмно-апаратних засобів керування реконфігурацією

Усі пристрої системи зв’язано загальним комунікаційним середовищем. Керувальний процесор і контролер реконфігурації працюють у режимі децентралізованого керування в структурі локального обчислювального модуля. Синхронізація роботи керувального процесора і контролера реконфігурації відбувається в режимі переривань. Обмін даними між віртуальною пам’яттю обчислювального модуля і обчислювальною структурою на ПЛІС відбувається в режимі прямого доступу до пам’яті. Керування режимом прямого доступу до пам’яті виконується засобами контролера прямого доступу. Ефективність алгоритмів блокового передавання даних, що реалізовує контролер прямого доступу до пам’яті, визначається виразами (3.34). У праці [179] запропоновано технічні рішення для реалізації трансляційного передавання даних, що дає змогу одночасно завантажувати дані для обчислення в декілька ФБ на різних кристалах ПЛІС. Це підвищує ефективність обміну даними на рівні обчислювального модуля РКС за рахунок

зменшення кількості звернень до загальної шини. У праці [180] розроблено технічне рішення для реалізації динамічних пріоритетів під час опитування пристроїв у розподіленій системі переривань. У праці [181] запропоновано технічні рішення для зменшення часу затримання початку обробки переривань. У праці [182] розроблено засоби для підвищення ефективності синхронізації роботи процесорів на основі реалізації динамічних пріоритетів у багатопроцесорних обчислювальних системах на ПЛІС. Запропоновані засоби вдосконалення обробки переривань дають змогу підвищити ефективність розв'язання задач у РКС в режимі реального часу за рахунок зменшення часу очікування і гарантованої обробки переривань від усіх ФБ реконфігуровного обчислювального середовища.

У праці [183] запропоновано архітектурну концепцію побудови гетерогенного обчислювального середовища для реконфігуровних систем-на-кристалі. У праці [184] розроблено концепцію та апаратні засоби вдосконалення програмовного процесорного *RISC* (*Reduced Instruction Set Computer*) ядра, що дають змогу підвищити ефективність обробки інформації у гетерогенному обчислювальному середовищі через реалізацію реконфігуровної системи команд. У працях [185, 186] реалізовано на ПЛІС функціональні модулі програмовного RISC ядра.

4.2.3. Розроблення моделі пам'яті керувальних даних для автоматичного керування ресурсами реконфігуровного обчислювального середовища. Завдання визначаються унікальним ідентифікатором у вихідному графі $N_i \mid i = \overline{1, g}$, де g – кількість вершин графу обчислювальної задачі. Кожне завдання N_i асоціюється з параметром «тип операції» $I_j \mid j = \overline{1, m}$, який відповідає синтезованій в цифрову схему функції, де m – кількість функцій, реалізованих у бібліотеці конфігурацій. Синтезована в цифрову схему функція є апаратною задачею $Task_j$, яка визначається виразом (2.10). На поверхні ПЛІС апаратна задача подається ФБ $F_s \mid s = \overline{1, n}$, де n – кількість ФБ у

реконфігуровному обчислювальному середовищі на ПЛІС. Конфігураційні дані для реалізації апаратних задач заздалегідь створюються і зберігаються в централізованій БКД у вигляді конфігураційних файлів $M_j \mid j = \overline{1,m}$, для кожного параметра «тип операції» кількість файлів у БКД дорівнює m. Таким чином, параметр «тип операції» однозначно визначає конфігураційний файл у БКД і використовується як віртуальна виконавча адреса для пошуку конфігураційних даних апаратних задач у віртуальній пам'яті. Співвідношення між описаними параметрами зображено на рис. 4.12.

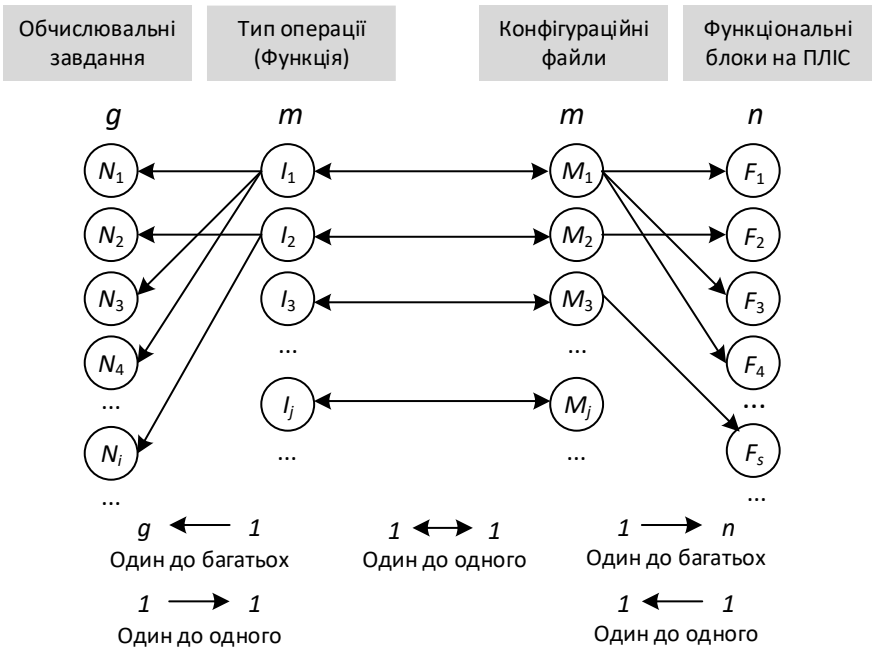


Рис. 4.12. Співвідношення між основними параметрами

Описані співвідношення важливі для формування виконавчої адреси пошуку конфігураційних файлів у віртуальній пам'яті обчислювального модуля. Кожний конфігураційний файл однозначно визначається типом операції. Декілька ФБ можуть виконувати одну і ту саму функцію відповідно до типу операції, але кожне обчислювальне завдання N_i у певний момент часу виконується в окремому ФБ F_s. Протягом часу в цьому функціональному блоці може бути виконана певна послідовність завдань обчислювального алгоритму, зважаючи на застосування технології повторного використання

обчислювальних ресурсів, а саме – ФБ апаратних задач. Приклад відображення трьох рівнів ГА обчислювальної задачі на реконфігуровне обчислювальне середовище і розподіл на нього ФБ зображено на рис. 4.13.

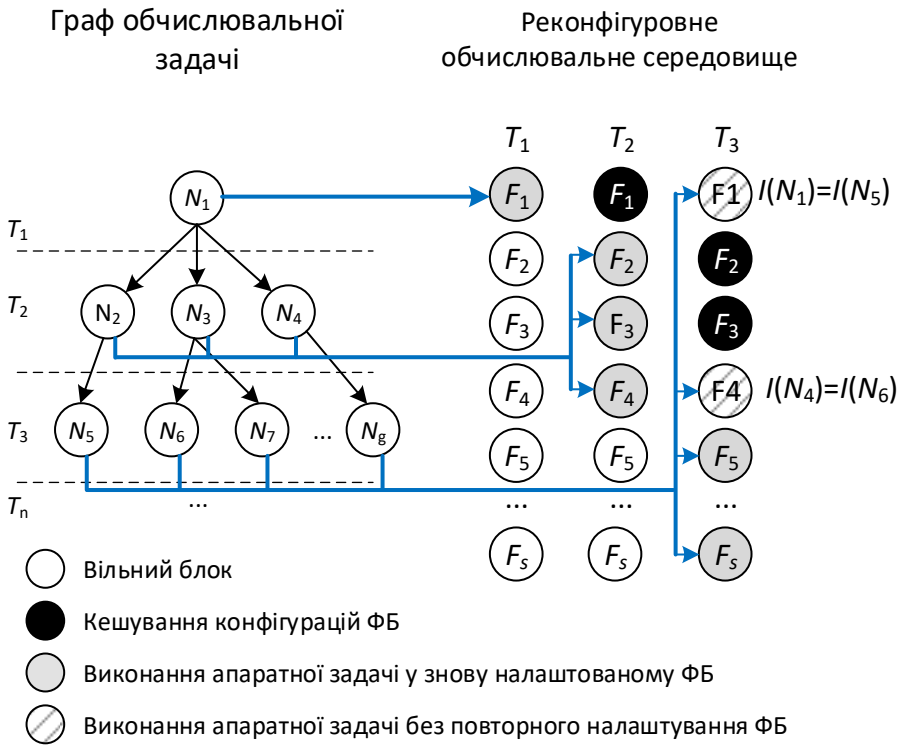


Рис. 4.13. Приклад відображення за рівнями обчислювальних завдань з видаленням повторного налаштування ФБ

Відповідно до викладеного конфігураційні дані кожної апаратної задачі $Task_j$ у бібліотеках з адресним доступом ідентифікуються віртуальним ідентифікатором – значенням типу операції I_j, а для ідентифікації розміщеного на реконфігуровній обчислювальній поверхні ПЛІС ФБ апаратної задачі $Task_j$ застосовується складений з двох полів віртуальний ідентифікатор $I_j.F_s$:

$$A_{Task\_j}^{Library} = [I_j],\ A_{Task\_j}^{FPGA} = [I_j.F_s]. \qquad (4.1)$$

Формування виконавчої віртуальної адреси для різних рівнів віртуальної пам’яті зображено на рис. 4.14.

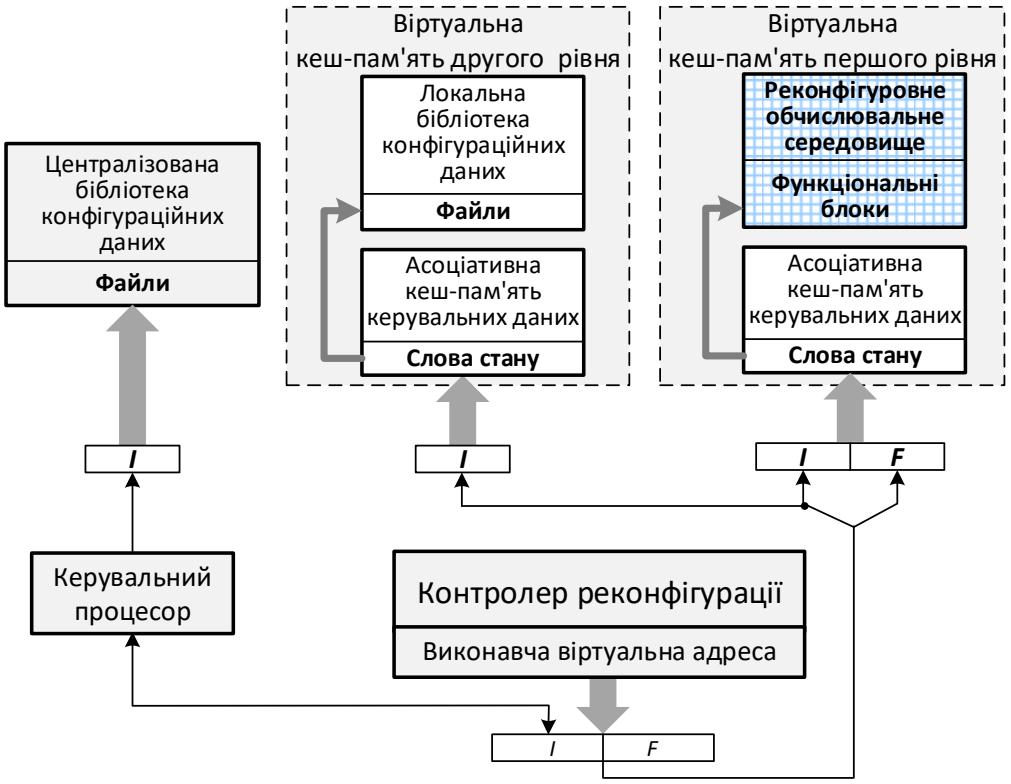


Рис. 4.14. Формат виконавчої адреси

Модель організації простору для зберігання ФБ апаратних задач на поверхні ПЛІС зображено на рис. 4.15.

Припустімо, що загальна кількість виконуваних операції $m = 64$, а загальна кількість ФБ на одній ПЛІС дорівнює $n = 63$. Тоді для зберігання слів стану кожного ФБ знадобиться 4 кбайт пам’яті з адресним доступом за умови мінімальної розрядності слова даних, що дорівнює 1 байт. Для збереження, наприклад, 4 байт керувальної інформації знадобиться 16 кбайт пам’яті з адресним доступом. Модель пам’яті з адресним доступом для розгляданого прикладу зображено на рис. 4.16. Зважаючи на проблему обмеженого обсягу внутрішньої пам’яті ПЛІС, яку описано у праці [158], розглянутий спосіб організації сховища керувальних даних призведе до непродуктивного використання значної частини пам’яті кристала ПЛІС.

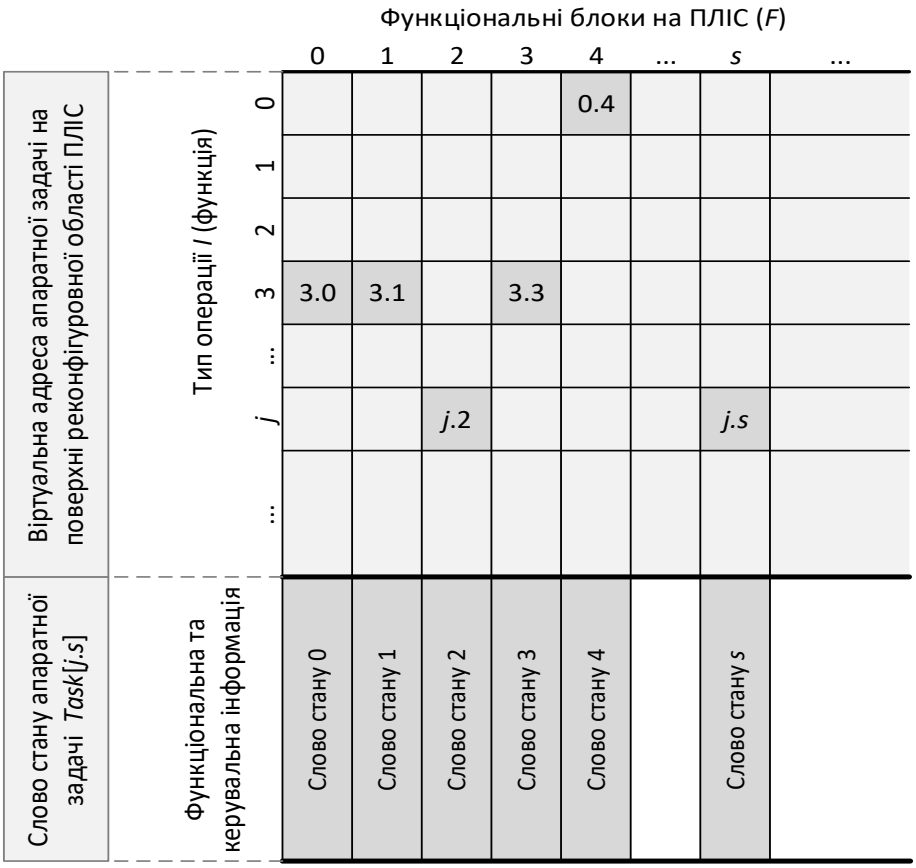


Рис. 4.15. Модель зберігання апаратних задач на поверхні реконфігуровної області ПЛІС

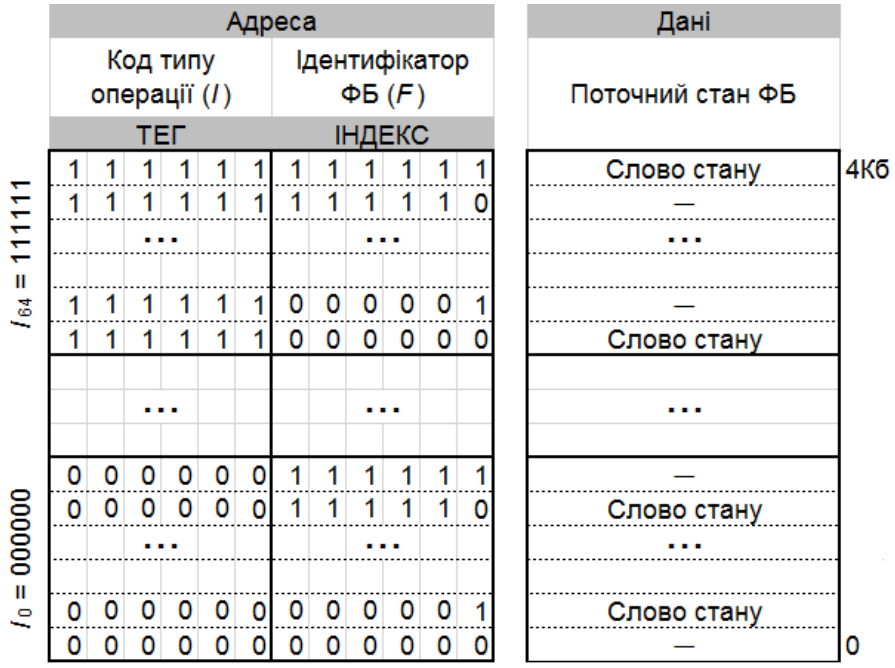


Рис. 4.16. Модель пам’яті з адресним доступом

Організація пам’яті керувальних даних у зовнішній пам’яті обчислювального модуля спричинить додаткові затримки під час налаштування обчислювального середовища на ПЛІС. Для подолання цієї проблеми запропоновано використання технології асоціативного пошуку на основі класичної технології кеш-пам’яті.

З огляду на те, що в розглянутому прикладі на рис. 4.16 тільки 64 слова стану з доступних 4 кбайт пам’яті з адресним доступом використовуватимуться для збереження слів стану, використання кеш-пам’яті з асоціативним пошуком інформації скоротить ємність пам’яті керувальних даних до 64 байт. Зі збільшенням довжини слова стану, наприклад, до 4 байт, ємність асоціативної пам’яті зросте лише до 256 байт, що потребуватиме доволі незначних запам’ятовувальних ресурсів мікросхеми ПЛІС і може бути використано для реалізації пам’яті керувальних даних безпосередньо на кристалі ПЛІС. Це звичайно пришвидшить процес пошуку та аналізу керувальних даних під час налаштування обчислювального середовища і виконання обчислень.

Функціональні особливості класичних технологій кеш-пам’яті, такі як локальність розміщення, прозорість для програмного шару, апаратна реалізація пошуку, швидкість пошуку, відсутність збиткових записів, мобільний розмір, вирішують ряд проблем, зумовлених накладними витратами реконфігурації і обмеженим ресурсом пам’яті ПЛІС.

Для побудови ефективного сховища керувальної інформації розглянемо особливості використання двох типових технологій організації кеш-пам’яті: асоціативну кеш-пам’ять і кеш-пам’ять з відображенням адрес.

Після формування виконавчої адреси в асоціативній пам’яті її старші біти, що утворюють тег, одночасно порівнюються з усіма тегами записів пам’яті. Якщо всі порівняння призводять до негативного результату, фіксується так званий «кеш-промах», в іншому випадку, коли знайдено шуканий запис, фіксується «кеш-успіх».

З іншого боку, повторне використання раніше сконфігурованих ФБ та підтримання їх розкладу зберігання потребує виконання пошуку в пам’яті станів шляхом перебирання всіх записів пам’яті. Звичайно, із застосуванням пам’яті з адресним доступом час виконання такого пошуку буде пропорційний кількості записів в пам’яті. Асоціативний пошук – це типовий засіб, спрямований на пришвидшення пошуку інформації. Для виконання пошуку в асоціативні пам’яті процесор видає в інтерфейс пам’яті спеціальне слово, яке називається тегом. Під час пошуку заданий тег порівнюється з тегом кожного запису асоціативної пам’яті. Таким чином, виконання пошуку зводиться до одного звернення до пам’яті з боку процесора, що пришвидшує процедуру пошуку і розвантажує комунікаційне середовище. Сама процедура асоціативного пошуку підтримується апаратно на рівні інтерфейсу пам’яті.

Якщо як тег використовується ідентифікатор адреси ФБ, що складається з двох полів $I.F$, отримаємо максимальне зменшення ємності пам’яті. Тоді, якщо видалити збиткові записи, ємність пам’яті дорівнюватиме 64 записи, по одній для кожного ФБ. При цьому пришвидшення пошуку досягатиметься лише за рахунок зменшення кількості записів. Пошук збереже всі ознаки пошуку в пам’яті з адресним доступом, а, отже, значне пришвидшення неможливе.

Пояснимо це таким чином. Між типом виконуваних операцій і ФБ, що сконфігуровані на ПЛІС, існує зв’язок один-до-багатьох, тобто можлива наявність певної кількості однотипних блоків. Метою пошуку є знаходження всіх ФБ на ПЛІС, що виконують певний тип операції. Технологія повністю асоціативної кеш-пам’яті, в свою чергу, дає змогу знайти лише один запис. У цьому випадку процедуру пошуку доведеться покласти на центральний процесор, що суперечить ідеї досліджень. Використання ж як тега лише поля типу операції можливе, але слова станів усіх однотипних блоків доведеться зберігати в одному рядку кеш-пам’яті. Це призведе до ускладнення й уповільнення процедури пошуку і потребуватиме додаткового вдосконалення інтерфейсу кеш-пам’яті. Окрім цього, типова апаратура повністю асоціативної

кеш-пам’яті будується на застосуванні компараторів, кількість яких дорівнює кількості записів кеш-пам’яті.

Наведемо також оцінювання апаратних ресурсів для реалізації асоціативної кеш-пам’яті. Відповідно до припущень потрібно 64 дванадцятибітні компаратори. Приблизне оцінювання затрат устаткування для реалізації одного компаратора складає 10 транзисторів на один біт компаратора [158], таким чином, знадобиться 7680 транзисторів для реалізації лише схеми порівняння в інтерфейсі кеш-пам’яті. Для реалізації схеми порівняння в типових реалізаціях пам’яті, що використовується в найбільш поширених процесорах, ємністю 256 кбайт з довжиною рядка 32 байт знадобиться 2 211 840 транзисторів.

На підставі викладеного визначимо, що поставленій меті щодо виконання ефективного пошуку конфігураційних даних найбільше відповідає асоціативна пам’ять з прямим відображенням адрес.

Рядки кеш-пам’яті адресуються асоціативним тегом, за який приймаємо тип операції, а ряд індексів, що переадресовується на кожний рядок пам’яті, є ідентифікаторами ФБ. Схему організації кеш-пам’яті першого рівні показано на рис. 4.17. За зроблених припущень кількість рядків, яку визначає розрядність тега, дорівнює 64.

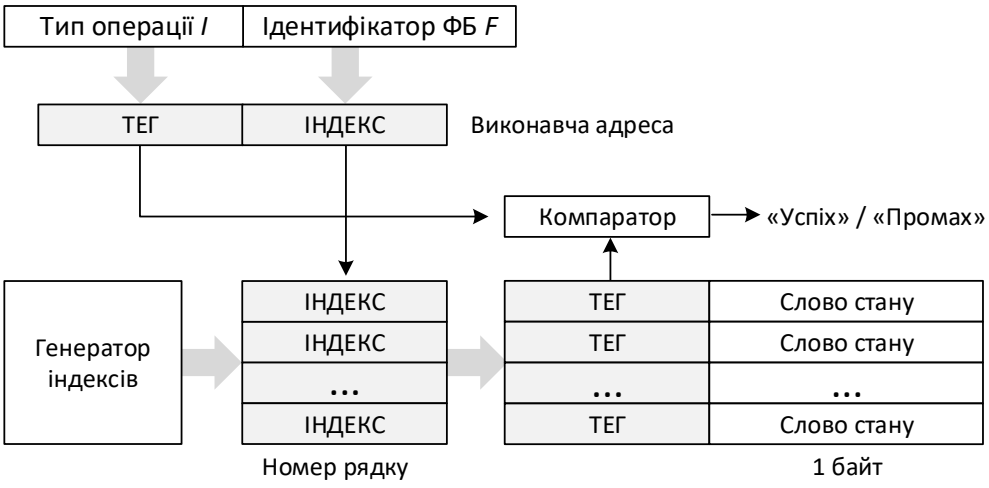


Рис. 4.17. Мнемонічна схема звернення до кеш-пам’яті першого рівня

Алгоритм функціонування наведено далі. На адресний вхід інтерфейсу пам’яті надходить виконавча адреса, що складається з полів [$I.F$] = [ТЕГ.ІНДЕКС]. Аналізується поле «ІНДЕКС», яке вказує на один із ФБ. Надалі старші розряди адреси («ТЕГ»), що вказують на шуканий тип операції, порівнюються з тегом, який зберігається в рядку. За збігу формується сигнал «кеш-успіху», інакше – «кеш-промаху». Така пам’ять потребує мінімальної кількості устаткування. Для порівняння знадобиться лише один компаратор, на вхід якого подається тег кеш-рядка обраного за полем «ІНДЕКС».

Таким чином, реалізовано таку логіку цільового пошуку. Фактично виникає потреба в перебиранні всіх ФБ і пошуку серед них тих, які виконують шуканий тип операції. Надалі розподілювач задач аналізує стан вибраних блоків і обирає серед них блок для завантаження чергової задачі.

Для звільнення процесора від виконання послідовного перебирання адрес ФБ до складу інтерфейсу пам’яті введено спеціальний пристрій – генератор адрес (рис. 4.17). Тоді виконавча адреса, що видається процесором на початку пошуку, складається з полів [I.0]. Генератор адрес послідовно видає індекси для пошуку, а значення коду операції порівнюється з тегами рядків.

Традиційний недолік застосування кеш-пам’яті з прямим відображенням адрес, коли неможливо одночасно зберігати два рядки з одним і тим самим індексом і різними значеннями тегів, не заважає розв’язанню кінцевої задачі, оскільки неможливо виконувати декілька операцій в одному ФБ.

Записи, що видаляються із кеш-пам’яті, потрапляють у кеш-пам’ять другого рівня. Відповідно до цього, конфігурації ФБ видаляються з поверхні реконфігуровної області, а конфігураційні дані зберігаються у локальній пам’яті обчислювального модуля.

Призначення кеш-пам’яті другого рівня полягає в наданні додаткового часу для зберігання конфігураційних даних на локальному рівні обчислювального модуля. Це сприяє додатковому зменшенню накладних

видатків реконфігурації за рахунок відсутності комунікаційних передач міжмодульного рівня.

Для реалізації такої кеш-пам’яті застосуємо повністю асоціативну кеш-пам’ять (рис. 4.18). Ємність кеш-пам’яті для зберігання конфігураційних даних обмежується декількома рядками, що зумовлює доволі незначні апаратурні витрати. Як асоціативний тег для пошуку застосовується поле типу операції, що однозначно визначає файл конфігурації ФБ.

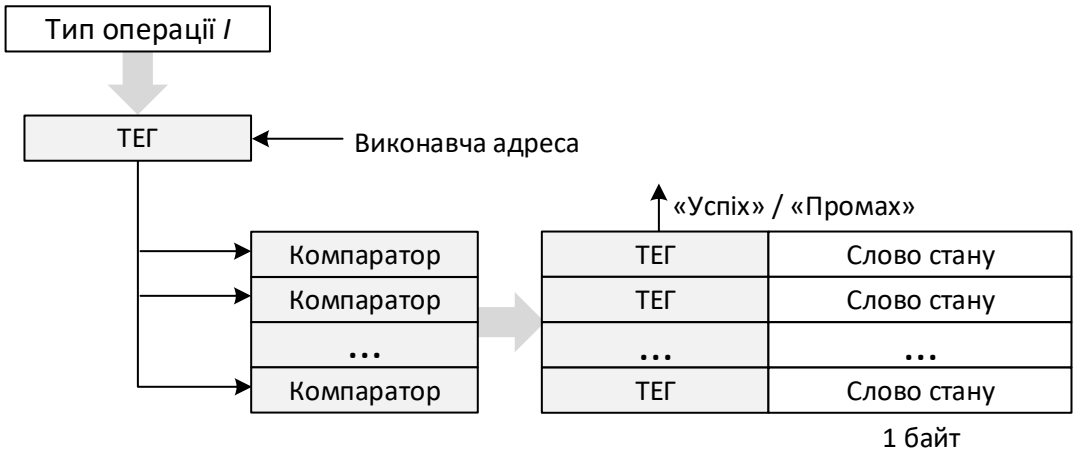


Рис. 4.18. Мнемонічна схема звернення до кеш-пам’яті другого рівня

Розклад зберігання конфігураційних файлів у кеш-пам’яті другого рівня підтримується автоматично стандартними засобами без участі планувальника. У цьому випадку доцільна стратегія обмеження кількості рядків пам’яті і витіснення з пам’яті найбільш давніх, або таких, що мають меншу ймовірність використання. Для цього використовуються відомі алгоритми: алгоритм заміщення на основі найбільш давнього застосування (*Least Recently Used, LRU*) або заміщення найменш часто використовуваного рядку (*Least Frequently Used, LFU*) [158].

4.2.4. Спосіб апаратного обліку та пошуку реконфігуровних обчислювальних ресурсів на основі багаторівневої віртуальної пам’яті. У межах відомих технологій більшість алгоритмів повторного використання ресурсів ФБ реалізовані таким чином, що апаратні задачі не видаляються з

поверхні реконфігуровної області після завершення їх виконання настільки довго, наскільки це можливо [44]. Засоби планування і розподілу завдань у відомих реконфігуровних обчислювальних системах зазвичай ґрунтуються на підтриманні списку вільного або зайнятого простору на поверхні ПЛІС на програмному рівні операційної системи [44].

Описана вище організація багаторівневої і багатофункціональної пам’яті, зокрема, швидкодійної пам’яті керувальної інформації, на відміну від відомих технологій, використовується для організації процесу планування та розподілу задач на апаратному рівні локального обчислювального модуля прозоро для програмного шару операційної системи. Для підтримання апаратних задач з урахуванням обмежень поверхні ПЛІС у праці [158] запропоновано новий спосіб обслуговування апаратних задач на поверхні реконфігуровної області ПЛІС, який дає змогу об'єктивно оцінити частоту використання кожного ФБ і на підставі цього визначити оптимальне місце зберігання конфігураційних даних з метою мінімізації накладних витрат реконфігурації з урахуванням апаратних обмежень обчислювального середовища на ПЛІС.

Запропонований спосіб обслуговування апаратних задач на поверхні реконфігуровної області ПЛІС ґрунтується на визначенні терміну зберігання кожного ФБ на поверхні ПЛІС на підставі механізму надання «бонусів». Для реалізації цього запропоновано протокол обслуговування ФБ апаратних задач на поверхні реконфігуровної області ПЛІС, основні етапи якого в змістовних термінах описано далі.

Етап 1 – надання «бонусів». Під час обчислювального процесу визначено готовність до виконання чергового завдання. Конфігураційні дані відповідної апаратної задачі містяться на поверхні реконфігуровної області ПЛІС. Нехай на поверхні ПЛІС є готова до розв’язання апаратна задача $Task_{[Ij.Fs]}$, яка характеризується ідентифікатором $[I_j.F_s]$ (4.1). При цьому апаратна задача, яка знову завантажена в поточному такті циклу виконання завдання у ФБ F_s, отримує максимальну кількість початкових «бонусів»

$CountB_{[Ij.Fs]}[s] := B_{\max}$, або, якщо задача $Task_j$ вже була на поверхні реконфігуровної ділянки для повторного використання, вона отримує додатковий «бонус» для продовження терміну зберігання на поверхні ПЛІС – $CountB_{[Ij.Fs]}[s] := CountB_{[Ij.Fs]}[s]+1$, де $CountB_{[Ij.Fs]}[s]$ – лічильник бонусів ФБ F_s.

Етап 2 – витрачання «бонусів». Для всіх неактивних апаратних задач, що є на поверхні реконфігуровної ділянки ПЛІС у режимі очікування, виконується операція декременту лічильника «бонусів». У випадку обнуління лічильника бонусів апаратної задачі $CountB_{[Ij.Fs]}[s] = 0$, задача вивантажується в зовнішню локальну пам'ять обчислювального модуля. На цьому етапі можуть надаватись додаткові «бонуси», якщо задача не має копій на реконфігуровній обчислювальній поверхні ПЛІС. Але у випадку нестачі місця для розміщення нових задач, така задача буде примусово вивантажена в зовнішню локальну пам'ять, навіть якщо її бонуси не будуть ще вичерпані.

Приклад часової діаграми надання та витрачання «бонусів» під час функціонування ФБ на поверхні реконфігуровної ділянки ПЛІС зображено на рис. 4.19.

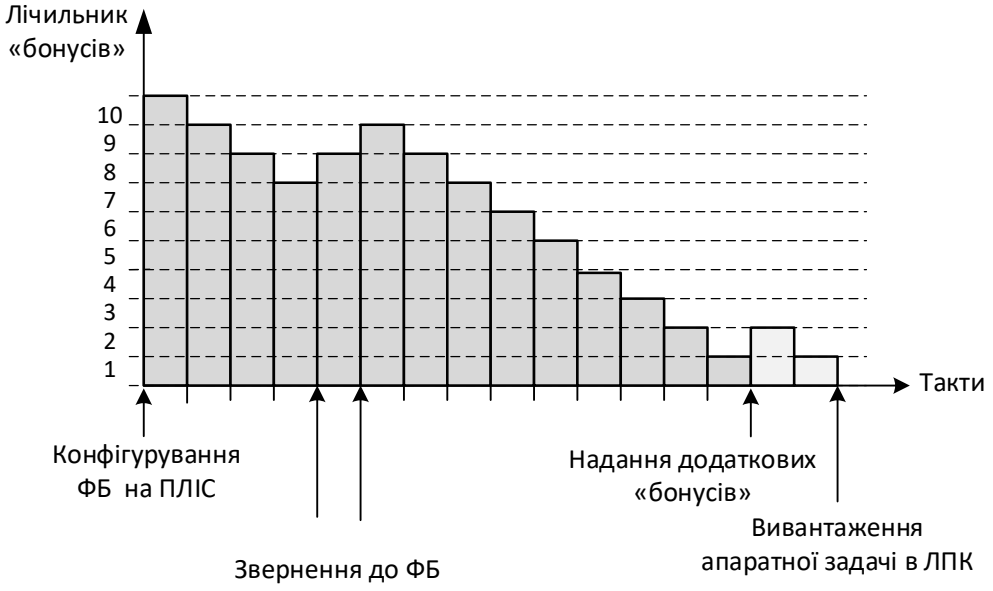


Рис. 4.19. Механізм надання «бонусів»
для підтримання ФБ на поверхні ПЛІС

Наведем псевдокод протоколу підтримання ФБ апаратних задач на поверхні ПЛІС.

```
procedure add_bonus;
// процедура надання «бонусів» для поточного ФБ апаратної задачі
begin
     if (CountB = = 0)  // ФБ вперше завантажений на поверхню ПЛІС
     CountB[s]:=CountB_max; // надання максимальної кількості «бонусів»
     else
     CountB[s]:=CountB[s]+1; // інкремент лічильника «бонусів»
end;


procedure Intr_run
// передавання керування керувальному процесору
begin
// встановлення прапорців
// передавання адреси ФБ керувальному процесору за перериванням
// завантаження вихідних даних керувальним процесором
// запускання виконання апаратної задачі в ФБ і активація лічильнику тактів
end;


procedure control_bonus;
// процедура перевірки стану лічильників «бонусів» усіх ФБ на ПЛІС

begin
// перебирання елементів масиву лічильників тактів CountT[s]
     for (s=1, s<=n, s++
     begin
     if (CountT = = 0) // апаратна задача в ФБ неактивна
          begin
          CountT[s]:=CountT[s]-1; // декремент лічильника «бонусів»
          if (CountB[s] = = 0)  // обнуління лічильника «бонусів»
          // прийняття рішення щодо видалення ФБ з поверхні ПЛІС
               procedure Search_FPGA;
               // процедура пошуку на ПЛІС копії ФБ апаратної задачі Task[j.s]
               // якщо знайдена неактивная копія, задача Task[j.s] видаляється
                    for (f=1, f<=n, f++
                    begin
                    if (hitI = = 1) // керувальний сигнал від кеш-пам'яті I
                    // є запис в кеш-пам'яті – є копія Task[j.f]
                         if (CountT[f] = = 0)  // копія неактивна
                         procedure load_localMemory;
                         // процедура вивантажування задачі Task[j.s] в ЛПК
                         // видалення відповідного рядка кеш-пам'яті I рівня
                         Hit_load:=1; // успішне видалення задачі Task[j.s]
                         return Hit_load;
                         end;
                    end;
               return Hit_load;
               end;
          if (Hit_load != 0)
          CountB[s]:=CountB[s]+2;
          // надання додаткових «бонусів» для ФБ задачі Task[j.s]
          end;
          end;
      end;
end;
```

4.2.5. Структура даних багаторівневої віртуальної пам’яті обчислювального модуля. Для формалізації процесу організації ефективного пошуку конфігураційної інформації у віртуальній пам’яті асоціюємо її з певною віртуальною багаторівневою структурою даних.

Набори даних, серед яких виконується пошук в апаратній базі даних, являють собою такі об’єкти:

– макрокоманди, що визначають виконувані в кожній вершині графу завдання;

– вихідні дані для виконання завдань;

– апаратні задачі;

– конфігураційні дані, що зберігаються на різних рівнях віртуальної пам’яті;

– сукупність керувальних даних, що зберігаються в асоціативних структурах віртуальної пам’яті.

Керування даними та пошук в апаратній базі даних виконується засобами процесорів, що входять у структуру обчислювального модуля. Ключами для пошуку конфігураційних даних є адресна інформація, що зберігається в адресних полях керувальних записів. Керувальні записи являють собою слова стану елементів даних.

Кожне завдання є деякою обчислювальною функцією, яка в процесі обробки інформації визначається словом макрокоманди або актором, відповідно до формальних визначень (2.8), (2.9). Формат слова макрокоманди наведено в табл. 4.1.

Таблиця 4.1

Формат слова макрокоманди

N_i	I_j
Унікальний номер завдання	Тип операції
Адресна інформація для визначення віртуальної адреси вхідних даних	Віртуальна адреса конфігураційних даних для налаштування обчислювальної структури на ПЛІС

Апаратна задача у вигляді конфігураційних даних розміщена в центральній бібліотеці конфігураційних файлів; перелік її параметрів наведено в табл. 4.2.

Таблиця 4.2

Слово стану апаратної задачі

I_j	T_{exp}	B_{exp}	D_j
Унікальний ідентифікатор апаратної задачі	Очікуваний час виконання	Розмір бітового потоку	Кількість слів вхідних даних

Під час виконання обчислень у слові стану кожного ФБ збирається функціональна інформація, що характеризує процес розв’язання апаратної задачі, та керувальна інформація, що використовується для підтримання розкладу зберігання апаратних задач на поверхні реконфігуровного обчислювального середовища. Ця інформація застосовується під час повторного використання апаратних задач і являє собою набір властивостей (табл. 4.3).

Таблиця 4.3

Слово стану ФБ

<table>
<tr><td rowspan="2">F_s</td><td>I_j</td><td>$T_{COUNT\ j}$</td><td>$A_{s\ j}$</td><td>B_s</td><td>C_j</td></tr>
<tr><td>Тип операції</td><td>Скоригований час виконання</td><td>Просторова інформація</td><td>Бонус підтримання</td><td>Таймери та лічильники</td></tr>
<tr><td>ФБ</td><td colspan="3">Функціональна інформація</td><td colspan="2">Керувальна інформація</td></tr>
</table>

Узагальнимо тепер структуру даних багаторівневого віртуального простору вдосконаленого обчислювального модуля (рис. 4.2) реконфігуровної комп’ютерної системи, як наведено на рис. 4.18.

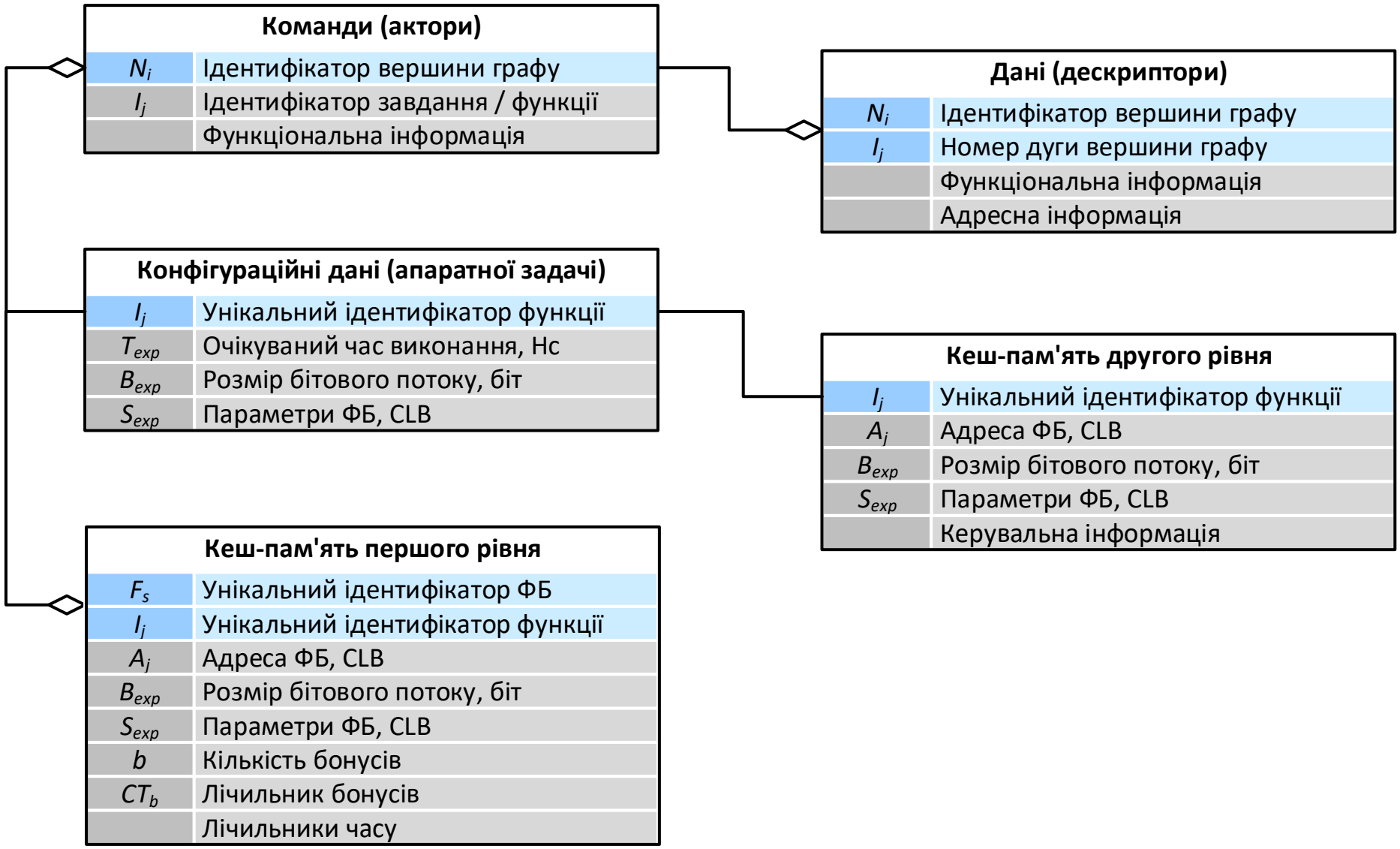


Рис. 4.20. Структура даних багаторівневої віртуальної пам’яті конфігураційних даних обчислювального модуля

ВИСНОВКИ ДО РОЗДІЛУ 4

1. Розгляд РКС на трьох нових рівнях абстракції дає змогу локалізувати процес керування відображенням обчислювальних задач на реконфігуровне обчислювальне середовище на структурному рівні обчислювального модуля, позбавити операційну систему від розв’язання неспецифічних задач і забезпечити на кожному рівні найбільш ефективну реалізацію відповідних процесів обробки інформації в РКС. Нові рівні абстракції забезпечують ефективну реалізацію методів та засобів, які запропоновані в дисертаційній роботі.

2. На основі нових рівнів абстракції розгляду засобів керування обчисленнями запропоновано нову концепцію структурної організації РКС, основними положенням якої є організація багаторівневого керування обробкою інформації та організація розподілених засобів керування відображенням задач на реконфігуровне обчислювальне середовище.

3. Удосконалено структуру і принцип функціонування обчислювального модуля. Удосконалення ґрунтуються на розділенні логічного і фізичного процесів реконфігурації і організації віртуального адресного простору на основі багаторівневої пам’яті. Запропонована концепція дає змогу зменшити часову складність процесу відображення задач на рівні обчислювального модуля РКС через реалізацію апаратного пришвидшення функціональних процесів.

4. Запропоновані прошарки пам’яті керувальних даних розширюють функціональні можливості багаторівневої кеш-пам’яті пам’яті і дають змогу виконувати процес керування реконфігурацією обчислювального середовища прозоро для операційної системи.

5. Запропоновані багаторівневі засоби керування обробкою інформації в РКС на основі нових рівнів абстракції мають властивості певної самоподібності на кожному з рівнів. Це дає змогу розширити функціональні можливості РКС через реалізацію циклічних і фрактальних моделей

динамічних процесів та структур, а також реалізацію розподілених неоднорідних обчислювальних структур на ПЛІС.

6. Розроблено новий спосіб апаратного обліку та пошуку конфігураційних даних. Підтримання та пошук ФБ на ПЛІС на відміну від відомих механізмів ґрунтується на визначенні терміну зберігання кожного ФБ на підставі показника частоти його використання, який визначається наданням «бонусів». Це дає змогу реалізовувати автоматичне керування даними на апаратному рівні обчислювального модуля прозоро для локального керувального процесора.

7. Застосування дворівневої кеш-пам’яті конфігураційних даних дозволяє зменшувати непродуктивні витрати часу в процесі обробки великих масивів даних. Реалізація автоматичного керування даними, що ґрунтується на застосуванні локальних керувальних засобів і швидкодійної асоціативної пам’яті керувальних даних, дозволяє прискорити процеси обліку та пошуку даних в дворівневій пам’яті і зменшити апаратурні витрати для реалізації пам’яті.

РОЗДІЛ 5
АПАРАТНІ ЗАСОБИ АВТОМАТИЧНОГО РОЗПОДІЛУ ЗАДАЧ У ДИНАМІЧНО РЕКОНФІГУРОВНИХ КОМП'ЮТЕРНИХ СИСТЕМАХ

5.1. Удосконалений метод автоматичного відображення задач у реконфігуровних комп'ютерних системах, керованих потоком даних

5.1.1. Модифікація математичної моделі формування заявок. Для автоматизації функції керування у РКС, керованих потоком даних, що описана математичними моделями (2.28) і (3.2), у дисертаційній роботі запропоновано засоби автоматичного відображення задач на реконфігуровне обчислювальне середовище, які ґрунтуються на вдосконаленні відомого методу автоматичного розпаралелення задач на рівні макрокоманд [153, 147]. Засоби автоматичного розпаралелення задач, розроблені для традиційних потокових обчислювальних систем, дають змогу спростити і скоротити цикл виконання макрокоманд, але не враховують апаратні і функціональні обмеження РКС. Це не дозволяє ефективно їх використовувати в обчислювальних системах, побудованих на сучасній динамічно програмовній елементній базі.

Механізм формування заявок у потокових обчислювальних системах, описаний у працях [51, 147, 148], ґрунтується на керуванні дескрипторами даних (2.9), що надходять у систему. У працях [147, 153] запропоновано метод автоматичного динамічного розпаралелення обчислень на рівні програмних модулів, команд і операцій у паралельних системах з неоднорідним доступом до пам'яті, керованих потоком даних. Цей метод пришвидшує реалізацію алгоритмів з неявним паралелізмом у багатозадачному режимі. Але метод розроблено для реалізації спеціалізованих потокових систем та обчислювачів з гомогенним обчислювальним середовищем, реалізованим на однотипних обчислювальних блоках без будь-яких функціональних та апаратних обмежень.

Для реалізації вдосконаленого способу автоматичного відображення задач на реконфігуровне обчислювальне середовище модифікуємо математичну модель формування заявки таким чином.

Обчислювальний процес подамо за допомогою макрографу потоку даних *MDG* [109, 155] $G_M = (N_M, D_M)$, де N_M – множина вершин; D_M – множина дуг. У вершинах графу розміщуються завдання (макрофункції) $N_i (i = \overline{1, g})$. Потік даних (дескриптор даних), який відповідає дузі D_{iv} графу, з'єднує *i*-у вершину з *v*-ю вершиною графу. Вихідні дані для кожного завдання готуються на підставі графу G_M і являють собою набір вхідних дескрипторів даних, а також бібліотеку апаратної реалізації функцій згідно із структурою, зображеною на рис. 3.17.

Далі наведено модифіковану математичну модель формування заявки з урахуванням процесу завчасної реконфігурації обчислювального середовища на ПЛІС.

Кожне завдання описується актором. Відомий формальний опис актора [51, 147, 148], що відповідає *i*-й вершині графу має вигляд

$$U_i = \{n_i, N_i, I_i, Q_i, \lambda_i, M_i, C_i\},$$

де n_i – ознака актора; N_i – ім'я актора; I_i – ідентифікатор завдання (команди, програми, процесу); Q_i – сумарна кількість потоків даних для завдання (вхідних дескрипторів); λ_i – рівень пріоритету завдання; M_i – множина імен акторів, для яких *i*-е завдання готує дані; C_i – множина імен потоків даних, які формуються під час виконання *i*-го завдання.

Модифікований формат актора ілюструє рис. 5.1. У його окремому полі зберігається кількість акторів *x*, для яких актор N_i готує дані (фактично кількість елементів множини C_i), при цьому немає потреби в передаванні на апаратний рівень пристрою формування заявок елементів множини M_i. У полі C_i актора міститься масив ідентифікаторів вихідних дескрипторів даних, які використовуються для адресації даних в пам’ять реєстрації під час

формування заявок. У полі ρ_i – показник апаратного пришвидшення виконання завдання, який обчислюється за виразом (2.12).

N_i	n_i	γ_i	Q_i	I_i	ρ_i	x_i	C_i

Рис.5.1. Формат актора

Дескриптор даних є множиною елементів

$$D_{oi} = \{n_{oi}, N_i, Q_i, A_{oi}\},$$

де n_{oi} – ім'я (номер) потоку даних (дуги, що входить до i-у вершину); $A_i = \{A_{oi}\}$ – елемент адресації даних, при цьому $o \mid o = \overline{1,x}$ – індекс попередньої вершини графу.

Формат дескриптора показано на рис. 5.2. Виходячи з того, що всі вершини графу мають унікальні імена N_i, елемент адресації визначає місце розташування даних у пам'яті обчислювального модуля, яка доступна для кожного локального процесора (рис. 4.9). Ім'я потоку даних (дескриптора) є ознакою, що відрізняє дескриптор даних від актора.

n_{oi}	N_i	Q_i	A_{oi}

Рис. 5.2. Формат дескриптора

З елементів акторів та дескрипторів відповідно до певної процедури λ формуються заявки на виконання i-го завдання, модифікована модель якої має вигляд:

$$\lambda(U_i, D_{vi}) \to Z_i = \{I_i, \Pi_i, M_i, C_i, A_i\}, \qquad (5.1)$$

де $\Pi_i = 0$ – прапорець, що визначає неефективність виконання завдання апаратними засобами реконфігуровного обчислювального середовища, інакше встановлюється $\Pi_i = 1$. У модифікованій моделі заявки процес завчасної реконфігурації врахований умовою формування заявки таким чином:

$$\omega_i(Z_i) = true \text{, якщо } \omega_i(Z_i) = \nu_i \,\&\, \alpha_{vi} \,\&\, [\varphi_{vi} | \rho_i > 1], \qquad (5.2)$$

де ν_i – умова отримання дескриптора *i*-го завдання; α_{vi} – умова отримання елемента адресації; φ_i – ознака виконаного налаштування ПЛІС для обчислення чергової задачі, яка формується за умови додатного значення показника пришвидшення апаратних обчислень ($\rho_i > 1$).

Формат слова заявки показано на рис. 5.3.

I_i	M_i	C_i	A_i

Рис. 5.3. Формат заявки

5.1.2. Розроблення структури і принципів функціонування вдосконаленого пристрою автоматичного розподілу завдань і синхронізації на ПЛІС. Структуру відомого пристрою автоматичного розподілу завдань подано у працях [51, 147, 148]. Мнемонічну схему, що зображує структуру вдосконаленого пристрою автоматичного розподілу завдань і синхронізації (ПАРС) та принцип формування заявок з реалізацією завчасної реконфігурації обчислювального середовища, описану у праці [187], показано на рис. 5.4.

У склад ПАРС на відміну від відомої реалізації [51, 147, 148] уведено контролер реконфігурації та буферну пам’ять черги реконфігурації, які застосовуються для керування процесом завчасної реконфігурації обчислювального середовища на структурному рівні обчислювального модуля. Наявність контролера реконфігурації забезпечує розпаралелення процесу виконання макрокоманд і процесу керування реконфігурацією обчислювального середовища. Запропоновані засоби функціонують на локальному рівні обчислювального модуля і керують паралельними алгоритмами і реконфігуровними апаратними ресурсами без участі операційної системи.

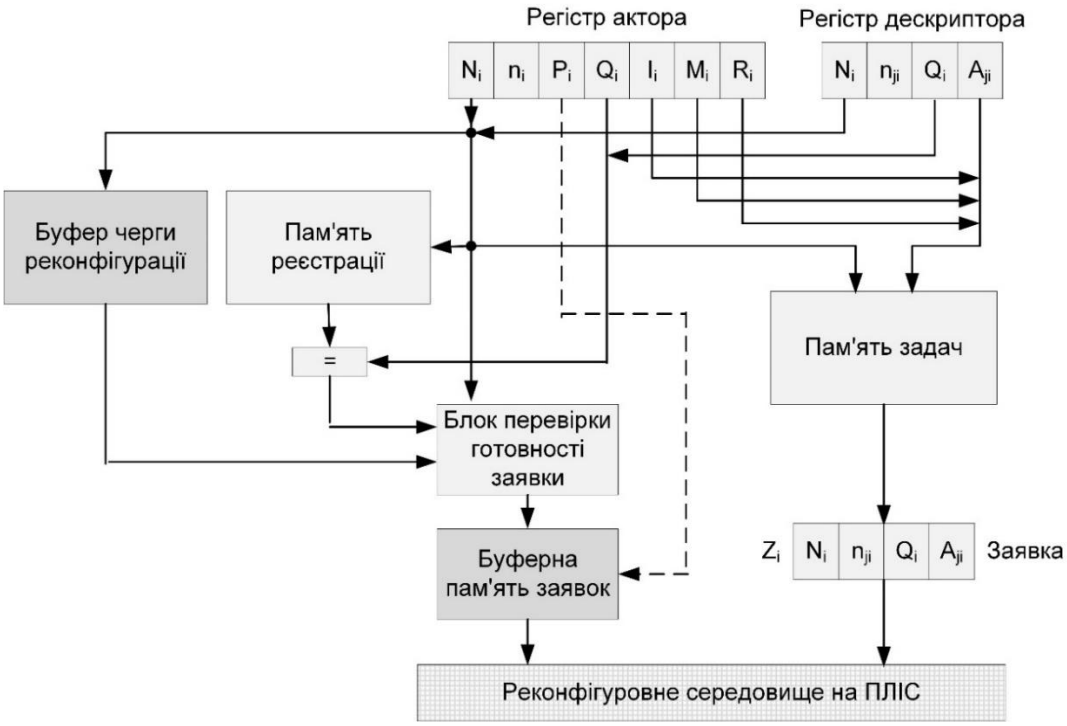


Рис. 5.4. Схема процесу формування заявки із завчасною реконфігурацією обчислювального середовища у вдосконаленому ПАРС

Удосконалений ПАРС розроблено і досліджено на базі ПЛІС *Cyclone II EP2C35F672C6* компанії *Altera*. Використано стандартні засоби проектування систем-на-кристалі SOPC *Builder*. Для реалізації керувального процесора та контролера реконфігурації застосовано вбудовані процесорні ядра *Nios II/s.* Модуль ПАРС реалізований як апаратна надбудова керувального процесорного ядра.

Керувальний процесор керує потоковими обчисленнями та формуванням заявок. Реалізовані на ПЛІС ФБ з точки зору керування є пасивними функціональними елементами. Дескриптори даних для кожного завдання формуються в різні моменти часу різними ФБ на ПЛІС. Контроль за надходженням даних для виконання чергового завдання виконує ПАРС.

Завчасна реконфігурація здійснюється таким чином. Процедура λ, що описана виразом (5.1), реалізовує автоматичну функцію керування процесом реконфігурації обчислювального середовища на підставі формальної моделі, зображеної на рис. 2.11. У результаті виконання цієї функції керування в

буфері черги реконфігурації (рис. 5.4) формується черга завчасної реконфігурації. Готовність заявки визнається формуванням ознаки готовності заявки відповідно до виразу (5.2) на керувальному виході блока готовності заявки (рис. 5.4).

Реконфігурація обчислювального середовища здійснюється згідно з параметром «тип операції I_i» у відповідному полі актора (див. рис. 5.1). Під час активації процесу реконфігурації відповідно до типу виконуваної операції з бібліотеки ФБ зчитуються конфігураційні дані для налаштування обчислювального середовища на ПЛІС для виконання чергового завдання.

У процесі ініціалізації системи, коли потік акторів завантажується в пам’ять задач [147], відбувається одночасне завантаження черги в буфер черги реконфігурації. Для забезпечення автономного виконання процесу реконфігурації в регістри буфера черги реконфігурації одноразово завантажується лише значення поля «тип операції I_i» (див. рис. 5.1). Із надходженням кожного наступного актора на компараторі, який входить у склад інтерфейсу блока черги реконфігурації (рис. 5.4), перевіряється наявність у черзі реконфігурації завдання на виконання, за відсутності якого нова функція ставиться у чергу для реконфігурації обчислювального середовища. Буфер черги реконфігурації побудовано на базі асоціативної пам’яті. Для реалізації буфера черги реконфігурації використовується структура асоціативної пам’яті, що зображена на рис. 4.17, де як індекс використовується поле ідентифікатора актора N_i, а як тег – поле типу операції I_i (див. рис. 5.1). Детально принцип функціонування такої пам’яті описано у праці [158]. Тегом для завантаження черги і виконання реконфігурації є поле типу операції I_i. У комірках пам’яті реалізовано лічильники для керування пріоритетами виконання реконфігурації, у цьому випадку за правилом «перший прийшов – перший обслугований» (*FIFO, First Input – First Output*), а також ознака налаштування відповідної апаратної задачі на поверхні ПЛІС.

З огляду на унікальність імен N_i дескрипторів для кожного завдання, умова готовності даних для виконання завдання формується на виході компаратора (див. рис. 5.1) таким чином:

$$\omega_i^* = \begin{cases} 1, \text{ якщ } c_i = Q_i, \\ 0, \text{ якщ } c_i \neq Q_i, \end{cases} \tag{5.3}$$

де c_i – кількість прийнятих у керувальний ОМ дескрипторів з іменем N_i.

Керувальний сигнал ω_i^* надходить на керувальний вхід кон'юнктура у складі блока перевірки готовності заявки, на другий керувальний вхід якого надходить ознака готовності налаштування обчислювального середовища відповідного завдання N_i. Якщо на кон'юнктурі формується умова

$$\omega = \omega_i^* \& \varphi,$$

готова для виконання заявка записується в буферну пам'ять заявок. Керувальний процесор ініціює виконання чергового завдання на поверхні ПЛІС. Порядок виконання формується у черзі буферної пам'яті заявок за принципом «перший прийшов – перший обслуговуваний»,

Дані з буфера черги реконфігурації вибираються за унікальними ідентифікаторами записів асоціативної пам'яті $[N_i].[I_i]$. Формат слова даних асоціативної пам'яті буфера черги реконфігурації зображено на рис. 5.5.

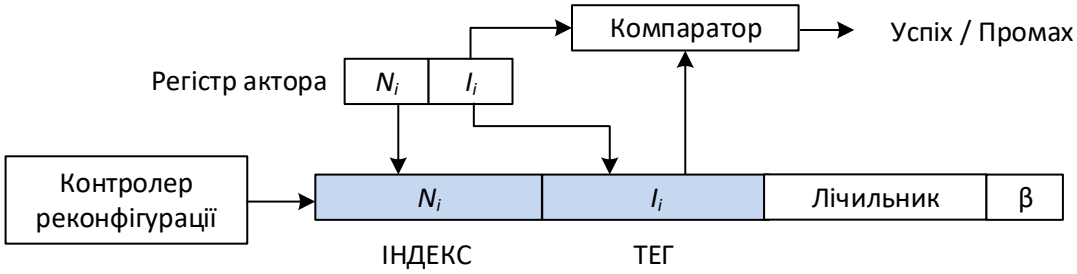


Рис. 5.5. Формат даних у буфері черги реконфігурації

Запис даних у пам'ять реєстрації та пам'ять задач, формування заявки на підставі готовності акторів та даних (5.3) виконуються згідно з відомою технологією, яку детально описано у працях [51, 147, 148].

У системі одночасно можуть розв'язуватися кілька задач у багатозадачному режимі роботи. У процесі реалізації багатопроцесорної

структури обчислювального модуля будь-який процесор може виконувати функції керувального та виконавчого процесора (навіть одночасно). Керувальний процесор зчитує із зовнішньої пам’яті даних потік акторів і вхідних дескрипторів. Виконавчий процесор зчитує готову для виконання заявку і ініціює запуск завдання на ПЛІС і звернення до пам’яті даних для отримання вхідних даних для виконання завдання. Після виконання завдання виконавчий процесор повертає дескриптор даних у пам’ять реєстрації, який бере участь у подальшому формуванні наступної заявки. Один керувальний процесор у структурі обчислювального модуля, описаний у праці [158], одночасно виконує функції керувального і виконавчого процесора.

Керувальний процесор самостійно отримує необхідну інформацію з блока пам’яті заявок для виконання завдання, визначає адреси даних на підставі значень N_i і Q_i. Безпріоритетна дисципліна обслуговування заявок, описана у праці [51], дозволяє формувати всі заявки в одній черзі, що значно спрощує механізм формування заявок, сприяє зниженню витрат апаратури і спрощує вирішення конфліктів під час доступу до спільних ресурсів. У праці [147] описано засоби керування формуванням заявок, що реалізовані на базі автомата із жорсткою логікою.

Модель реалізації обчислювального процесу на підставі вдосконалених апаратних засобів автоматичного розподілу завдань на динамічно реконфігуровне обчислювальне середовище зображено на рис. 5.6.

На відміну від традиційної моделі паралельної обробки інформації, описаної математичним виразом (2.36) (рис. 4.11), автоматичний розподіл завдань на реконфігуровне обчислювальне середовище, що реалізовує модель потокових обчислень, описаних виразом (2.37), дозволяє пришвидшити процес обробки інформації за рахунок ефективної реалізації прихованого паралелізму, скорочення циклу виконання завдань і зменшення непродуктивних часових витрат, у тому числі через спрощення процесу керування паралельною обробкою інформації і його апаратного пришвидшення.

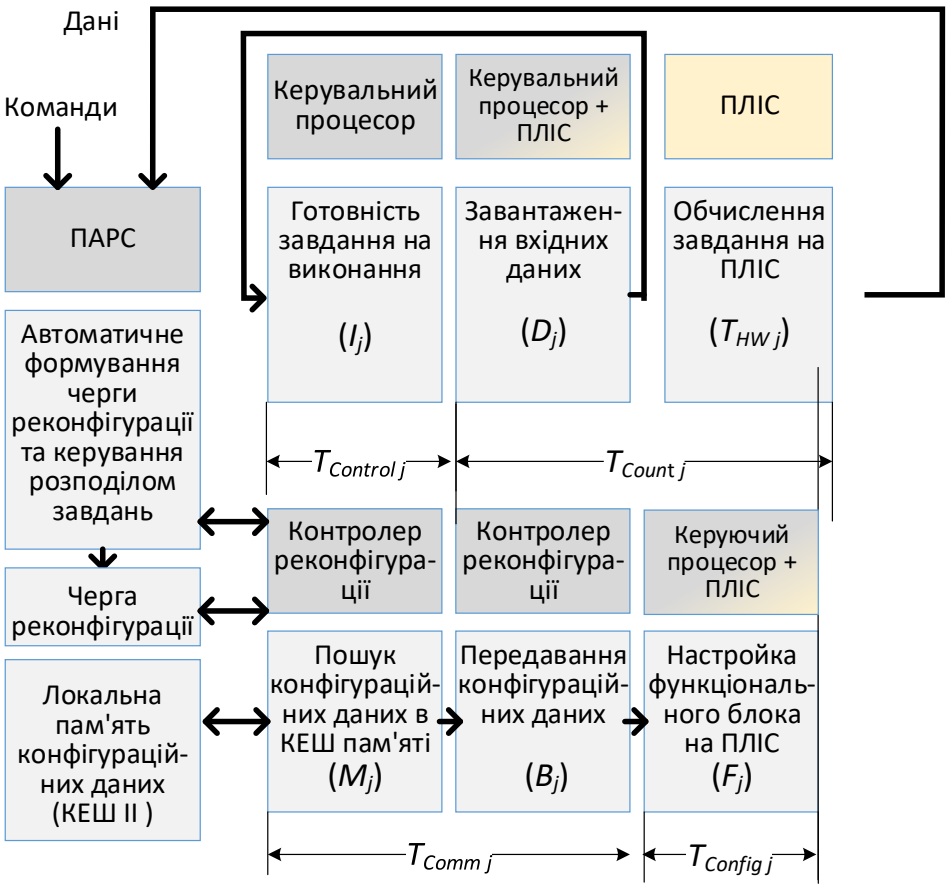


Рис. 5.6. Модель реалізації обчислювального процесу в системах, керованих потоком даних: КП – керувальний процесор, КП – контролер реконфігурації

5.1.3. Розроблення апаратних засобів реалізації автоматичного розподілу завдань. У працях [187, 188] розроблено апаратні засоби для реалізації завчасної реконфігурації на базі реалізації вдосконаленого способу автоматичного розпаралелювання задач. Основні особливості структурної та функціональної організації ПАРС описано далі.

Для опису процесу функціонування пристрою автоматичного розподілу завдань застосовують такі позначення (рис. 5.7):

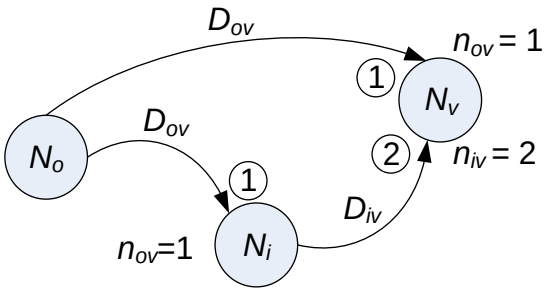


Рис. 5.7. Схема застосованих позначень

o, i, v – поточні індекси вершин графу, що визначають завдання (актори);

N_i – поточна вершина, щодо якої виконується опис;

N_v – наступна вершина, для якої актор N_i готує дані, актор-потомок;

N_o – попередня вершина, яка є джерелом даних для актора N_i, актор-джерело;

D_{oi}, D_{ov}, D_{iv} – дуги, що пов'язують відповідні вершини, дескриптори;

d_i – множина дескрипторів, що є вхідними для актора N_i: $d_i = \{D_{oi}\}$, $d_v = \{D_{ov}, D_{iv}\}$;

n_{di} – номер вхідного потоку даних у вершину N_i, номер дескриптора, де $d = \overline{1, Q_i}$: $n_{oi} = 1$, $Q_i = 2$; $n_{ov} = 1$, $n_{iv} = 2$, $Q_v = 2$.

Таким чином, дескриптор даних визначається номером актора N_i, для якого він визначає вхідний потік даних, та номером цього вхідного потоку даних n_{oi}: $D_{oi} = n_{oi} N_i$, $D_{ov} = n_{ov} N_v$, $D_{iv} = n_{iv} N_v$.

У склад кожного актора входить масив даних, що визначає ті актори-нащадки, для яких даний актор готує завдання. Масив нащадків M_i складається з дескрипторів, що готує актор N_i для своїх нащадків для виконання подальших обчислень: $M_i = \{n_{iv} N_v\}$, $M_o = \{n_{ov} N_v, n_{oi} N_i\}$.

Актор однозначно визначається ідентифікатором актора n_i і номером актора N_i, де $N_i = \overline{1, \Psi}$; Ψ – максимально можлива кількість акторів у системі.

Під час розподілення адресного простору пам'яті заявок адреси акторів і дескрипторів даних формуються таким чином:

– адреса актора: $[N_i].[n_i = 0]$;

– адреса дескриптора даних: $[N_i].[n_{oi}]$.

Схему формування заявок зображено на рис. 5.1. У процесі обробки інформації у пам'яті заявок нагромаджуються дані, необхідні для формування заявок. Розглянемо приклад формування заявок у ПАРС. У системі формування вихідних задач кожного керувального процесора можлива

максимальна кількість акторів $a = 2^5 = 32$ і максимальна кількість дескрипторів даних $d = 2^4 - 1 = 15$, що обмежується адресним простором пам'яті заявок.

Формат адреси пам'яті заявок складається з таких полів (рис 5.8):

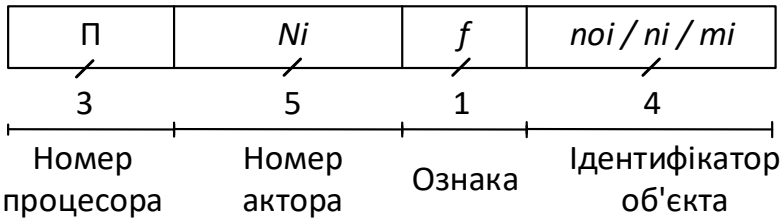


Рис. 5.8. Формат адреси акторів: П – номер процесору

Для восьми керувальних процесорів номер процесора змінюється в діапазоні $\Pi = \overline{000,111}$. Тоді кожний з них має основу з 32 номерів акторів, які він може розподіляти для створення графів задач. ПАРС має основу із 256 номерів акторів. Під час видачі керувальним процесором слів акторів та дескрипторів старші розряди адресного поля формуються автоматично в інтерфейсі керувального процесора. Загальна ємність пам'яті заявок для даної моделі $V_M = 2^3 = 8$ кбайт. Структурну схему формування адреси зображено на рис. 5.9.

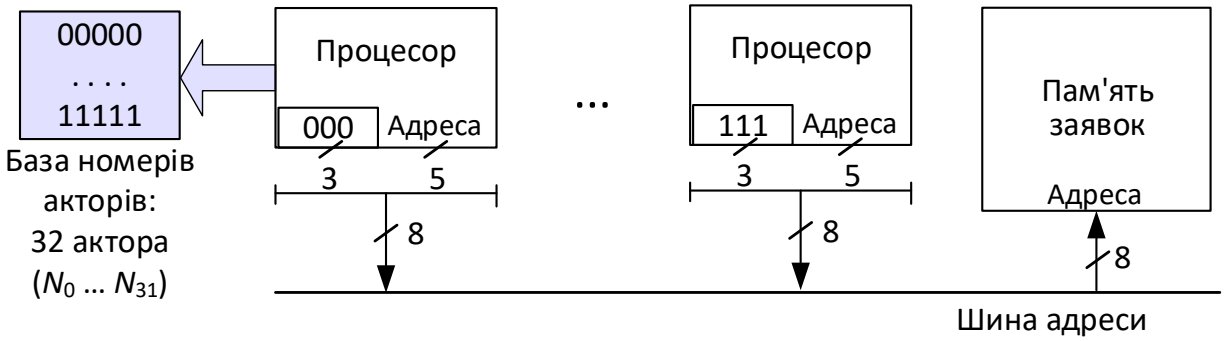


Рис. 5.9. Схема формування адрес у пам'яті задач

Шина адреси, що поєднує процесори та пристрій формування заявок, восьмирозрядна. Процесор послідовно у програмному режимі передає дані (актори та дескриптори). За шиною адреси вісім старших розрядів адреси надходять на адресний вхід пам'яті задач. Інші п'ять молодших розрядів

адреси формуються автоматично в інтерфейсі пристрою формування заявок під час надходження відповідних слів даних. Тобто інформація, що ідентифікує різні типи об'єктів (актори, дескриптори або масив потомків) і міститься у відповідних розрядах слів даних, надходить шиною даних на вхід інтерфейсу ПАРС і застосовується для формування молодших розрядів адреси об'єкта в пам'яті задач. Зазначена ідентифікаційна інформація в пам'яті заявок взагалі не зберігається. На структурній схемі (див. рис. 5.4) показано фрагмент інтерфейсу ПАРС, який пояснює принцип формування адреси пам'яті заявок. Виходячи із розрядності адреси ємність пам'яті задач: $A = 2^{13} = 8$ кбайт.

Ознака $f = 0$ є ознакою елемента масиву нащадків та актора, $f = 1$ є ознакою дескриптора. Номер вхідного потоку змінюється в діапазоні $n_{oi} = \overline{001,111}$, тобто можна ініціалізувати 15 вхідних дескрипторів. Наприклад, якщо значення у полі дорівнює 0000, то ознака актора $n_i = 0000$. Принцип формування адрес зберігання акторів та дескрипторів даних у пам'яті заявок для задачі, поданої графом на рис. 5.10, пояснимо на такому прикладі.

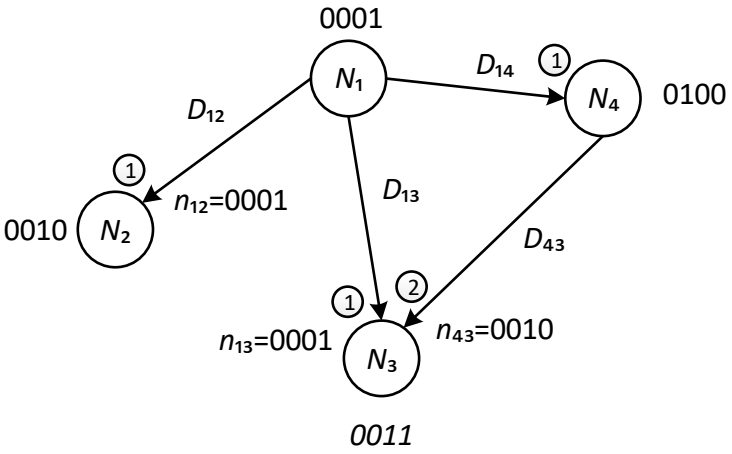


Рис. 5.10. Фрагмент графу обчислювальної задачі

Якщо задача надійшла з процесора $П = 000$, тоді актори N_1 і N_4 будуть записані в пам'яті заявок за адресами, указаними у табл. 5.1. Кожний актор у своєму складі містить масив нащадків, які будуть записані у пам'яті заявок за адресами, наведеними у табл. 5.1.

Таблиця 5.1

Приклад формування адрес акторів і дескрипторів у пам’яті задач

Ім’я об’єкта	Адреса у пам’яті заявок				Уміст комірки пам’яті
Актор					
	Номер процесору	Номер актора	Ознака	Ознака актора	$U_i = \{N_i; I_i; Q_i; \gamma_i\}$
N_1	000	00001	0	0000	$U_i = \{N_1; I_1; Q_1; \gamma_1\}$
Масив нащадків					
	Номер процесора	Номер актора	Ознака	Номер елемента масиву	$W_i = \{M_i; C_i; A_i\}$
$N_2^d n_{12}^d$	000	00001	0	0001	000 00010 0001 *A*
$N_3^d n_{13}^d$	000	00001	0	0010	000 00011 0001 *A*
$N_4^d n_{14}^d$	000	00001	0	0011	000 00100 0001 *A*
Актор					
N_4	000	00100	0	0000	$W_4 = \{N_4; I_4; Q_4; \gamma_4\}$
Масив нащадків					
$N_3^d n_{43}^d$	000	00100	0	0001	000 00011 0010
Дескриптор					
	Номер процесора	Номер актора	Ознака	Номер вхідного потоку	$D_{ji} = \{Q_i; A_{ji}\}$
$D_{14} = N_4 n_{14}$	000	00100	1	0001	$D_{14} = \{Q_4; A_{14}\}$

Актор складається з функціональної частини та масиву акторів-нащадків, для яких даний актор готує завдання. У форматі вдосконаленого ПАРС актор подамо масивом слів даних: перше слово – функціональна

частина актора, кожне наступне слово – елемент масиву потомків. Кожне слово має довжину шістнадцять розрядів. Масив слів актора передається за один спеціалізований цикл запису даних у пам'ять, який складається з одного звернення до шини адреси і (x + 1) звернень до шини даних, де x – кількість елементів масиву потомків. На початку циклу запису масиву слів актора у пам'яті заявок на шину адреси видається адреса актора, яка зберігається там протягом усього циклу запису, поки всі елементи масиву не будуть завантажені у пам'ять. Запис даних відбувається у програмному режимі.

Формат функціональної частини актора показано на рис. 5.11.

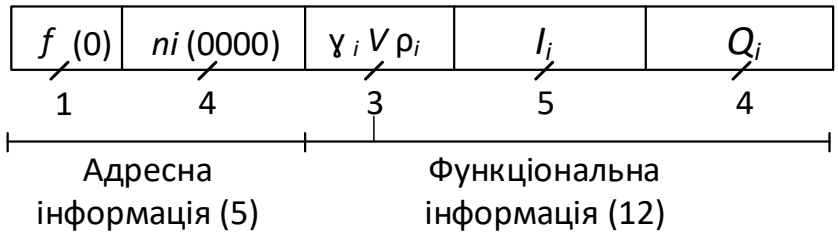


Рис. 5.11. Формат функціональної частини актора

Формат слів масиву потомків показано на рис 5.12. Формат слова дескриптора зображено на рис 5.13.

Кожне слово масиву містить пари $N_v n_{iv}$ – номер актора-нащадка і номер дуги. Чотирирозрядне поле m_i містить номер елемента масиву, дозволяє ідентифікувати п'ятнадцять акторів-нащадків (a = 15), для яких актор готує завдання. Ідентифікатори акторів і елементів масиву потомків беруть участь у формуванні адреси для запису відповідної інформації в пам'яті заявок. Функціональна частина актора записується у пам'яті заявок за адресою $N_i 00000$, елементи масиву потомків – за адресами $N_i, 0, m_i$, де $i = \overline{1,k}$.

Завантаження дескриптора відбувається за один стандартний цикл запису. Старша частина адреси дескриптора у пам'яті заявок відповідає номеру актора, для якого даний дескриптор є вхідним. Молодша частина адреси формується на підставі адресної інформації, що входить у склад слова дескриптора. Адресне поле складається з ознаки дескриптора $f = 1$ і номеру

вхідного потоку n_{ov}. Функціональна частина дескриптора записується у пам'яті заявок за адресою $N_i, 1, m_{oi}$. Адресна інформація у пам'яті заявок не зберігається.

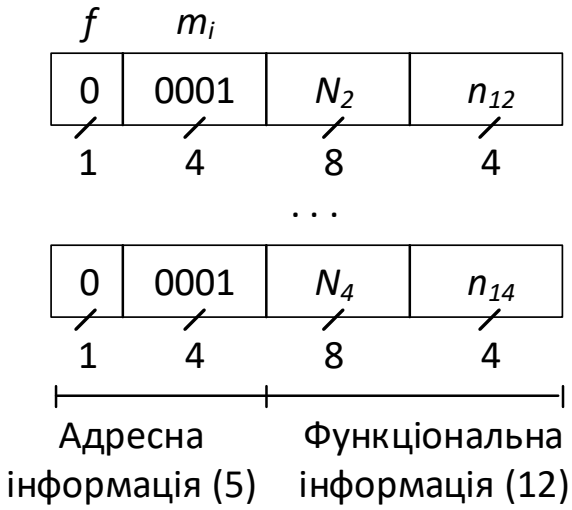


Рис. 5.12. Формат слів потомків

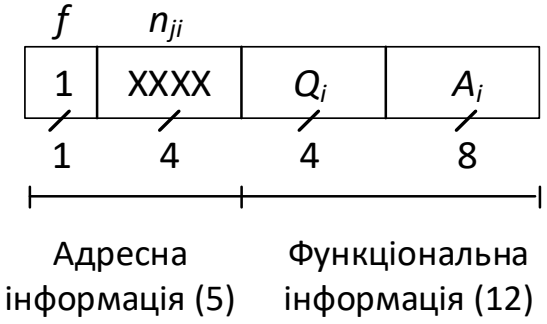


Рис. 5.13. Формат слова дескриптора

Структурна схема інтерфейсу пам'яті заявок зображена на рис. 5.14. Запис у пам'ять заявок виконується у програмному режимі керувальним процесором. Запис дескрипторів виконується за стандартним протоколом запису даних, алгоритм якого наведено у праці [189]. Для організації блокового передавання акторів пропонуємо модифікований протокол запису даних у пам'ять заявок, алгоритм якого показано на рис. 5.15.

Процесори і блок пам'яті заявок сполучені між собою системною шиною. Для забезпечення безконфліктного доступу до загальної пам'яті вдосконалимо систему арбітражу таким чином. Під час запису дескрипторів у пам'ять заявок процесор видає на шину адресу актора, яка фіксується в інтерфейсі пам'яті заявок у спеціальному регістрі адреси. Далі процесор видає на шину даних слово дескриптора, яке за сигналом процесора *write* записується у пам'ять заявок. Процесор отримує від інтерфейсу пам'яті заявок сигнали зворотного зв'язку *RD* (*ready*), що означає запис інформації в пам'ять, та *sta_a* (*strobe address*), що означає стробування адреси в регістр адреси.

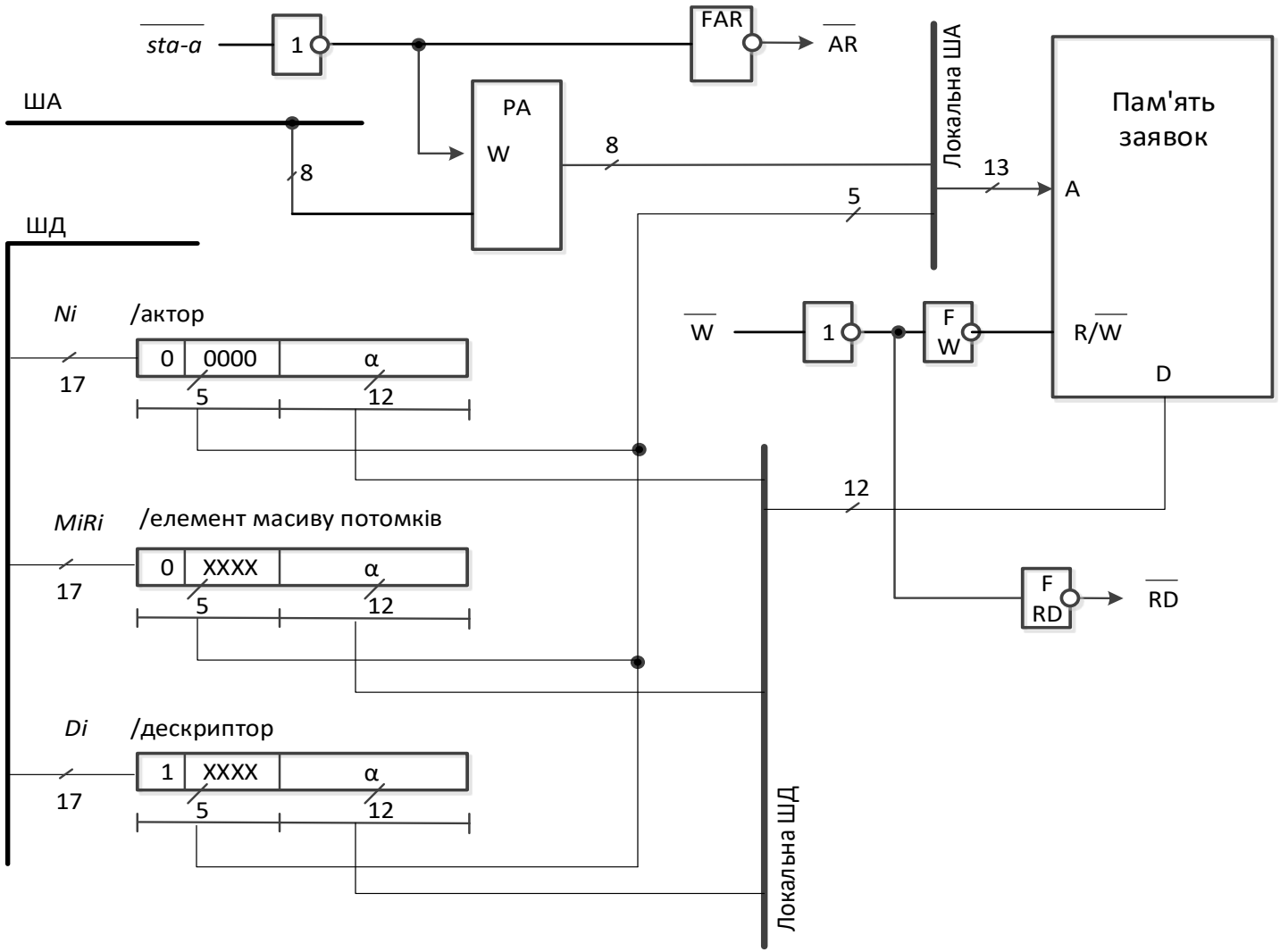


Рис. 5.14. Структурна схема інтерфейсу пам’яті заявок: ША – шина адреси; ШД – шина даних; ЛША – локальна шина адреси; ЛШД – локальна шина даних; РА – регістр адреси; *F* – формувач керувальних сигналів

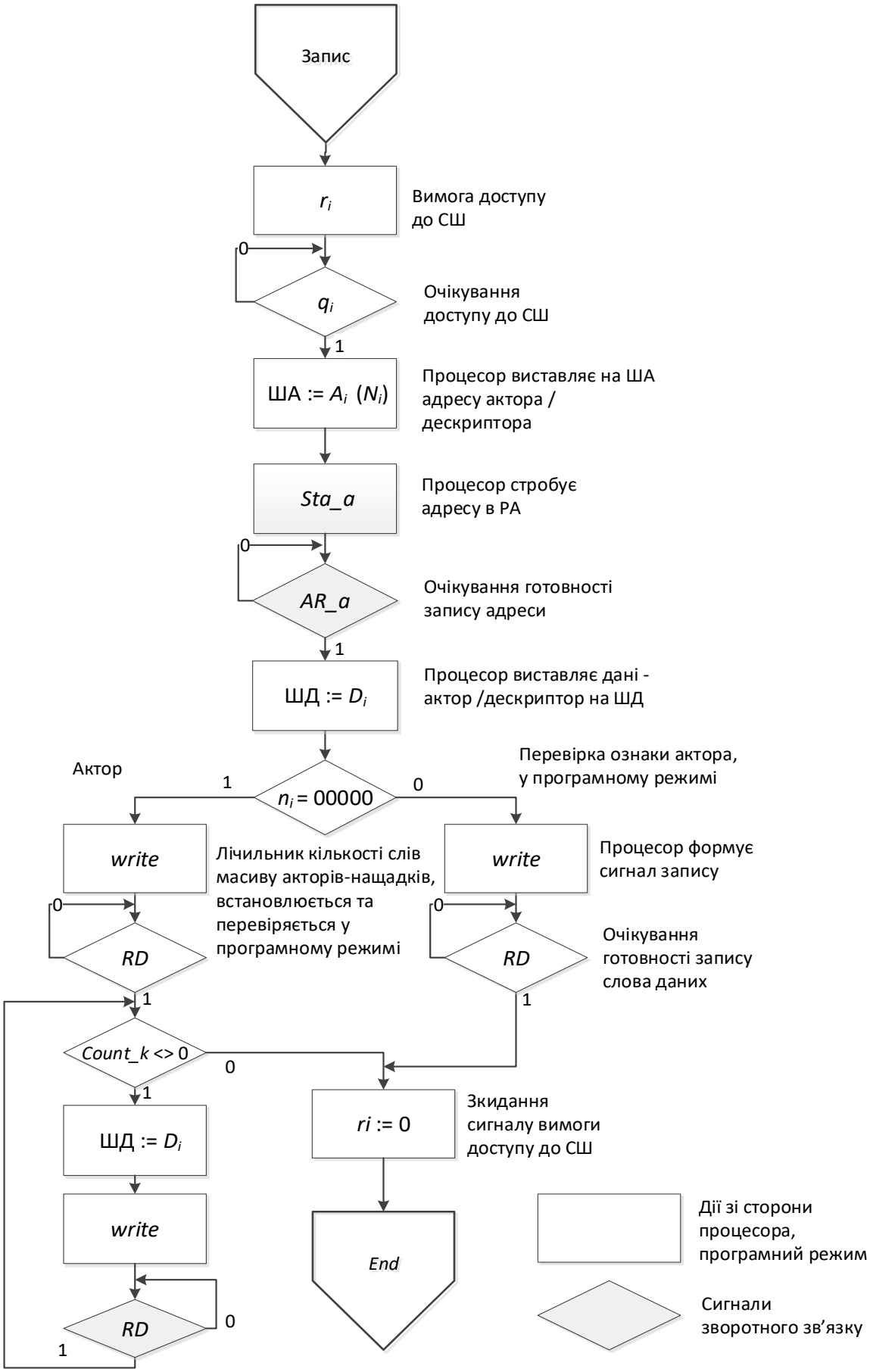


Рис. 5.15. Модифікований алгоритм запису масиву слів актора в пам'яті заявок

Процесор підключається до системної шини через блок централізованого арбітра. При цьому процесор генерує сигнали вимоги доступу r_i, який надходить на вхід арбітра; якщо шина вільна, арбітр видає сигнал дозволу доступу q_i. Процесор, який отримав доступ до системної шини виконує модифікований протокол запису (рис. 5.15), під час якого адреса актора фіксується в інтерфейсі пам'яті заявок у спеціальному регістрі адреси. Після цього процесор видає на шину даних масив слів актора, який за послідовністю керувальних сигналів *write* записується в пам'яті заявок. Під час запису масиву слів актора процесор здійснює одне звернення до шини адреси та (x + 1) звернень до шини даних. Процесор отримує від інтерфейсу пам'яті заявок послідовність сигналів зворотного зв'язку *RD* (*ready*), що визначають закінчення операції запису чергового слова даних у пам'ять. Запис слів актора відбувається у програмному режимі. Рішення щодо припинення передачі даних приймає процесор на підставі вмісту програмного лічильника *Count_x*, що підраховує кількість елементів масиву потомків на підставі значення поля *x* слова актора.

Структурну схему організації модифікованої системи арбітражу в ПАРС, показано на рис. 5.16. У функції централізованого арбітра (рис. 5.16) входить стандартна процедура перевірки зайнятості системної шини і видавання сигналу дозволу доступу до неї: низький рівень керувального сигналу на шині bs визначає, що шина зайнята, а високий рівень сигналу – вільна. Алгоритм керування, що реалізує керувальний автомат у складі централізованого арбітра наведено у праці [51].

У пам'яті реєстрації формуються змінні CT_i, що визначають кількість акторів і дескрипторів, які належать до одного завдання. Під час завантаження акторів та дескрипторів порівнюються значення змінної CT_i, яка зберігається у пристрої реєстрації за адресою N_i, і значення Q_i, яке надходить зі словами акторів та дескрипторів.

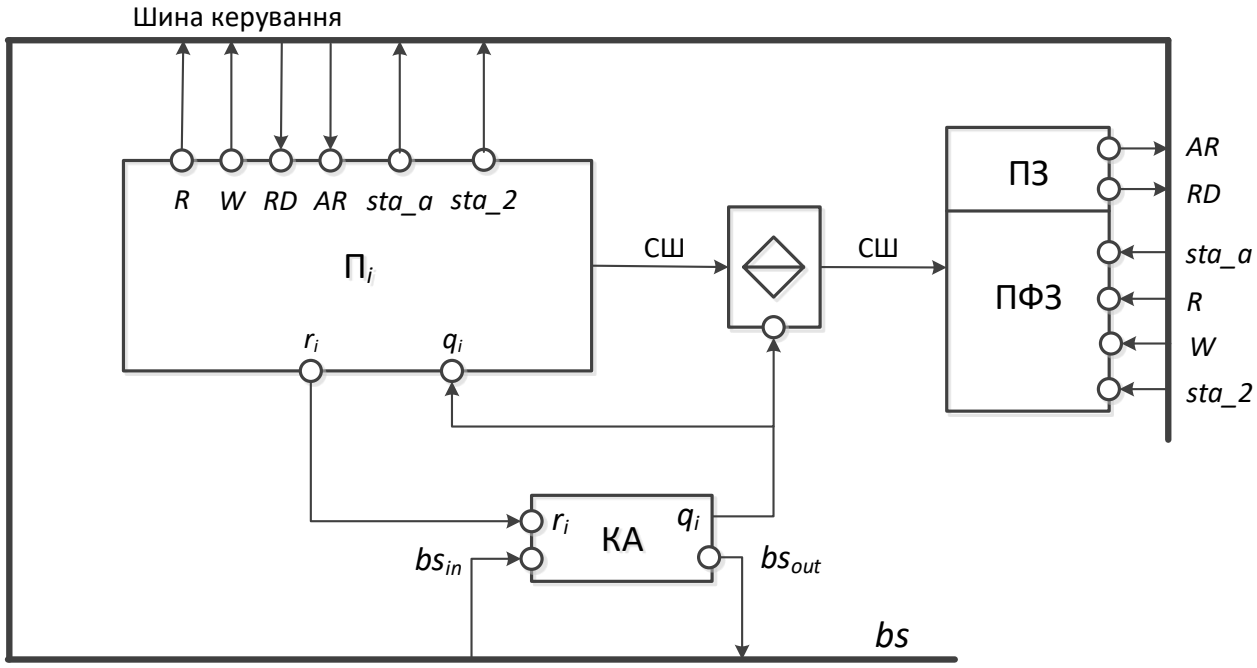


Рис. 5.16. Функціональна схема підключення процесора до пам’яті заявок з арбітражем на системній магістралі: П – процесор; ПЗ – пам’ять заявок; ПФЗ – пристрій формування заявок; КА – керувальний автомат; СШ – системна шина

Під час першого звернення до пам’яті реєстрації за адресою актора у відповідній комірці міститься значення нуля, яке надходить на схему порівняння. На другий вхід схеми порівняння надходить значення Q_i, що фактично дорівнює кількості дескрипторів для даного актора. Якщо значення не однакові, відбувається інкремент вмісту відповідної комірки пам’яті. Таким чином, у комірці пам’яті реалізований лічильник. У разі передавання останнього дескриптора у відповідній комірці пам’яті буде зберігатися значення Q_i, при цьому схема порівняння видасть сигнал рівності значень лічильника і значення Q_i, яке надійшло з останнім дескриптором. Рівність значень Q_i і CT_i означає, що вся інформація для формування заявки отримана, після цього ініціюється процедура передачі заявки у буфер *FIFO*. Функціональні схеми завантаження пам’яті реєстрації та блока формування заявок зображено на рис. 5.17 і 5.18 відповідно.

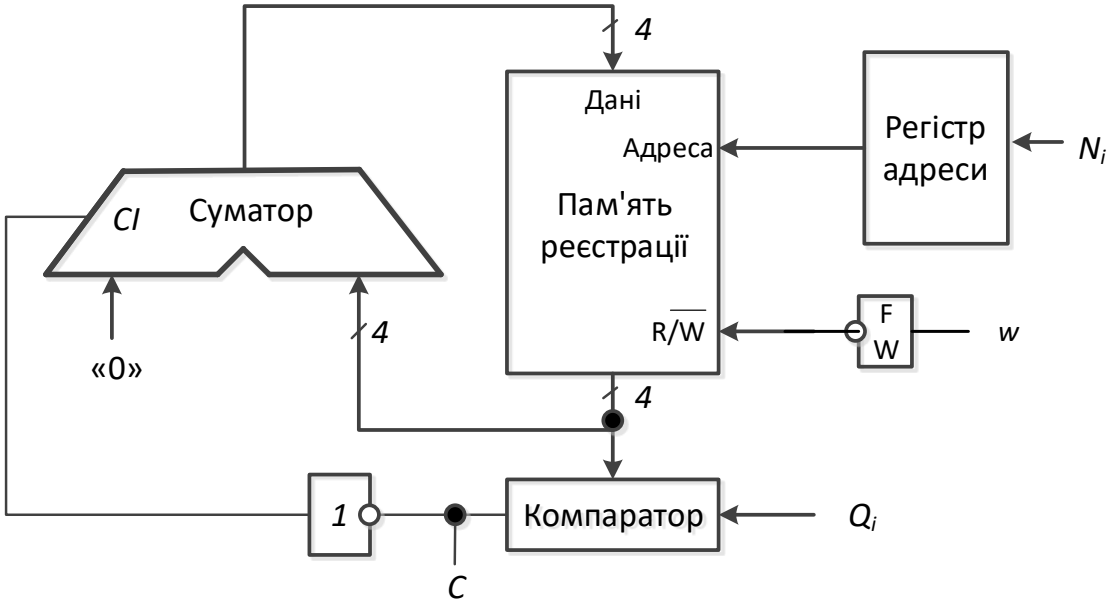


Рис. 5.17. Функціональна схема завантаження пам’яті реєстрації

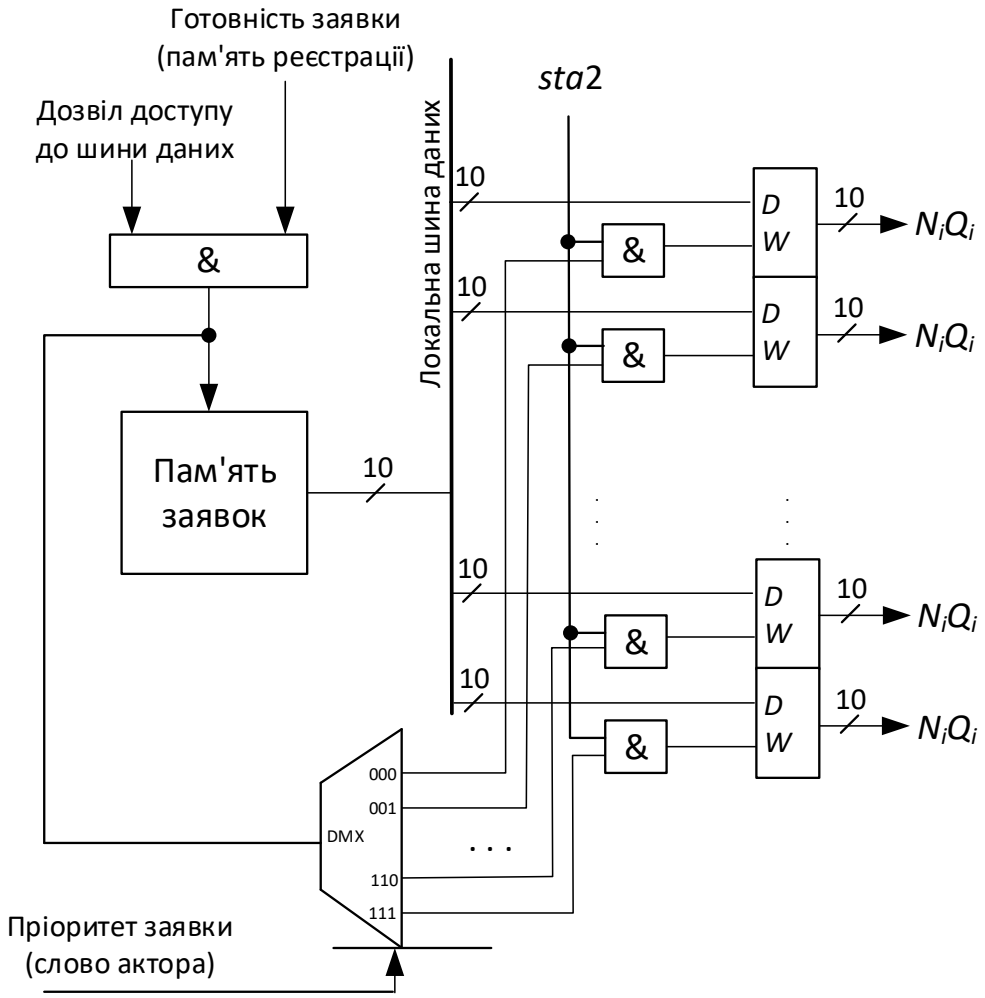


Рис. 5.18. Функціональна схема блока формування заявок

5.1.4. Моделювання вдосконаленого пристрою автоматичного розподілу завдань та синхронізації. Імітаційна модель методу адаптивного відображення задач на реконфігуровне обчислювальне середовище, що описаний математичною моделлю (2.37), та імітаційна модель РКС, керованої потоком даних, описані у працях [162, 163, 187]. На базі запропонованих імітаційних моделей досліджено процеси завчасної реконфігурації і повторного використання ресурсів ФБ. Програмні засоби розроблено кросплатформенною мовою *Java*. Імітаційна модель включає в себе такі блоки, реалізовані як програмні модулі, які емулюють функції відповідних апаратних функціональних елементів: *DataFlow* – головний модуль, що емулює функціонал всієї системи і об'єднує підлеглі модулі: *AppFormingService* – модуль, що відтворює роботу ПАРС і містить функціонал контролера реконфігурації; *FPGA* – реконфігуровне обчислювальне середовище; *Core* – модуль, що виконує роль програмно-апаратного ядра для обчислення задач і може масштабуватися в налаштуваннях системи; *Actor, Descriptor, Task, Application, Data* – модулі, що виконують роль відповідних структур у потоковій системі і забезпечують збереження даних у пам'яті.

Досліджено серію обчислювальних алгоритмів, поданих потоковими графами, з різною кількістю однотипних задач та різним ступенем зв'язності. Графік залежності часу виконання обчислювальних алгоритмів з різною кількістю повторів однотипних задач зображено на рис. 5.19. На рис. 5.20 показано усереднену залежність часу виконання реконфігуровних обчислень для потокових алгоритмів від різних стратегій пришвидшення реконфігурації, а саме повторного використання реконфігуровних ресурсів (T_G) [157, 159] та завчасної реконфігурації на базі автоматичного розподілу завдань (T_R) [162, 187]. Для оцінювання ефективності завчасної реконфігурації у РКС, керованих потоком даних, використано коефіцієнт пришвидшення $K = T_G / T_R$, де T_G – час виконання потокового алгоритму без будь-якого пришвидшення реконфігурації; T_R – час виконання обчислювального алгоритму із

застосуванням методу адаптивного відображення задач на реконфігуровне обчислювальне середовище, що описаний виразами (2.28) і (3.2).

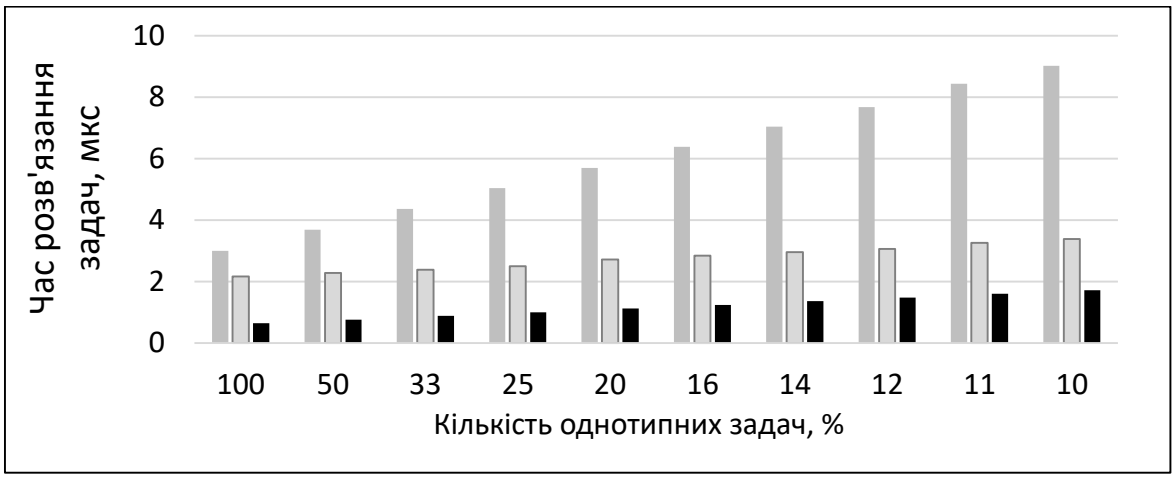


Рис. 5.19. Залежність комунікаційних затримок від кількості повторів однотипних функцій в обчислювальному алгоритмі: ■ – реконфігурація з повторним використанням ресурсів ФБ; ■ – завчасна реконфігурація; ■ – час виконання завдань на апаратурі ПЛІС

Рис. 5.20. Залежність коефіцієнта пришвидшення від кількості повторів однотипних задач

Повторне використання ресурсів ФБ апаратних задач на ПЛІС дозволяють збільшити швидкість процесу реконфігурації в середньому на 63%, що показано у праці [157]. При цьому ефективність методу адаптивного пришвидшення реконфігурації залежить від кількості однотипних задач в обчислювальному алгоритмі. Із графіка дослідження пришвидшення

обчислювального процесу із застосуванням автоматичного розподілу завдань на базі моделі обчислень, керованих потоком даних, отримуємо середній коефіцієнт пришвидшення, що дорівнює 2 (рис. 5.20). Завчасна реконфігурація дозволяє видалити майже весь непродуктивний час реконфігурації для будь-яких обчислювальних алгоритмів незалежно від кількості однотипних задач. Часові діаграми, отримані під час моделювання підтвердили теоретичне оцінювання часу реконфігурації, виконане у працях [157, 159], а також наведену в попередніх розділах дисертаційної роботи теоретичне оцінювання часу реконфігурації на підставі математичних моделей (2.24) – (2.27), (2.36) і (2.37) (див. рис. 2.8 і 2.9)

5.2. Удосконалення засобів синхронізації процесів у реконфігуровних комп'ютерних системах

5.2.1. Удосконалення базової архітектури обчислювального модуля на ПЛІС. Особливості застосування уніфікованої технології проектування від компанії *Altera* на базі інструментального середовища *SOPC Builder* (*System on a Programmable Chip Builder*) для автоматичної генерації обчислювальних систем на кристалі ПЛІС описано у працях [146, 190]. Основними функціональними елементами для створення таких систем на ПЛІС є модулі процесорних ядер *Nios* II, загальна шина з обміном даними через загальну пам'ять *Avalon* та набір *HDL*-модулів (*Hardware Description Language*), що включають контролери пам'яті, інтерфейси, периферійне устаткування тощо.

Для розв'язання задачі синхронізації процесів у системах, створених на базі засобів *SOPC Builder*, у праці [146] досліджується застосування механізму *Mutex*, який є суто програмною реалізацією і потребує попередньої підготовки програмного коду, та альтернативного засобу від компанії *Altera* – апаратних блоків *MailBox*, які застосовуються для обміну інформацією між кожною парою процесорів. У праці [146] визначено, що для забезпечення атомарності операції зміни вмісту *MailBox* застосовується механізм *Mutex*, що реалізовується програмно на рівні операційної системи і не може бути перспективним засобом

для реалізації апаратних засобів керування обчисленнями в системах на ПЛІС. Для реалізації *MailBox*, окрім того, використовується внутрішня пам’яті ПЛІС. Зі збільшенням кількості процесорів стрімко зростає кількість блоків *MailBox*. У системі з шістьма процесорами для обміну масивами даних з максимальною ємність 2 кбайт реалізація *MailBox* потребує 20% внутрішньої пам’яті ПЛІС.

Проблемою, що впливає на швидкодію синхронізації процесів із застосуванням традиційних механізмів синхронізації процесів під час доступу до загальних ресурсів є необхідність установлення прапорців у загальному адресному просторі [51, 146], що непродуктивно збільшує кількість звернень до загальної шини. Кількість таких звернень до загальної шини може коливатися в широких межах: від двох звернень для перевірки та встановлення прапорця до невизначеної кількості звернень у стані очікування дозволу передавання даних [51].

Для вдосконалення наявних засобів синхронізації процесів у праці [191] вдосконалено архітектуру із загальною шиною, на якій ґрунтується уніфікований процес проектування *SOPC Builder Altera*. У межах вдосконалення базових технологій проектування реалізовано ОМ РКС, який реалізує модель обчислень, керованих потоком даних. Для автоматичного керування обчисленнями і синхронізацією процесів до складу ОМ РКС, на відміну від базової технології, додано апаратний ПАРС, який реалізовано як апаратну надбудову центрального керувального процесорного ядра. Функціонування ПАРС ґрунтується на вдосконаленому методі автоматичного розподілу задач для РКС, керованих потоком даних [162, 163], який заснований на вдосконаленні механізму формування заявок, керованого дескрипторами даних [51, 147, 148]. Централізовані засоби керування на рівні операційної системи перетворюють вихідний обчислювальний алгоритм у потік акторів та вхідних дескрипторів. ПАРС забезпечує автоматичний розподіл завдань між процесорними ядрами.

ПАРС реалізований на апаратному рівні у вигляді окремого модуля, який підключається до загальної шини і виконує функції автоматично й автономно від процесу функціонування програмного шару операційної системи. Готові

заявки, які автоматично формуються в загальному середовищі формування заявок, зберігаються в локальній буферній пам’яті заявок блока і виконуються вільними процесорами. Узагальнену структуру ПАРС та його функціональні зв’язки з основними складовими ОМ РКС зображено на рис. 5.21. У загальному адресному просторі ПАРС являє собою пару адрес регістра адреси та регістра даних. Загальна кількість звернень до загальної шини, що відбувається під час розв’язання деякої задачі, дорівнює

$$V_{BUS} = W_{IN} + D_{IN} + W_{IN}(1 + D_{N\,i} + 2),$$

де W_{IN} і D_{IN} – кількість акторів та вихідних дескрипторів відповідно; $D_{N\,i}$ – кількість вхідних дескрипторів кожного актора $N_i \,|\, (i = \overline{1, W_G})$; W_G – кількість вершин ГА обчислювальної задачі.

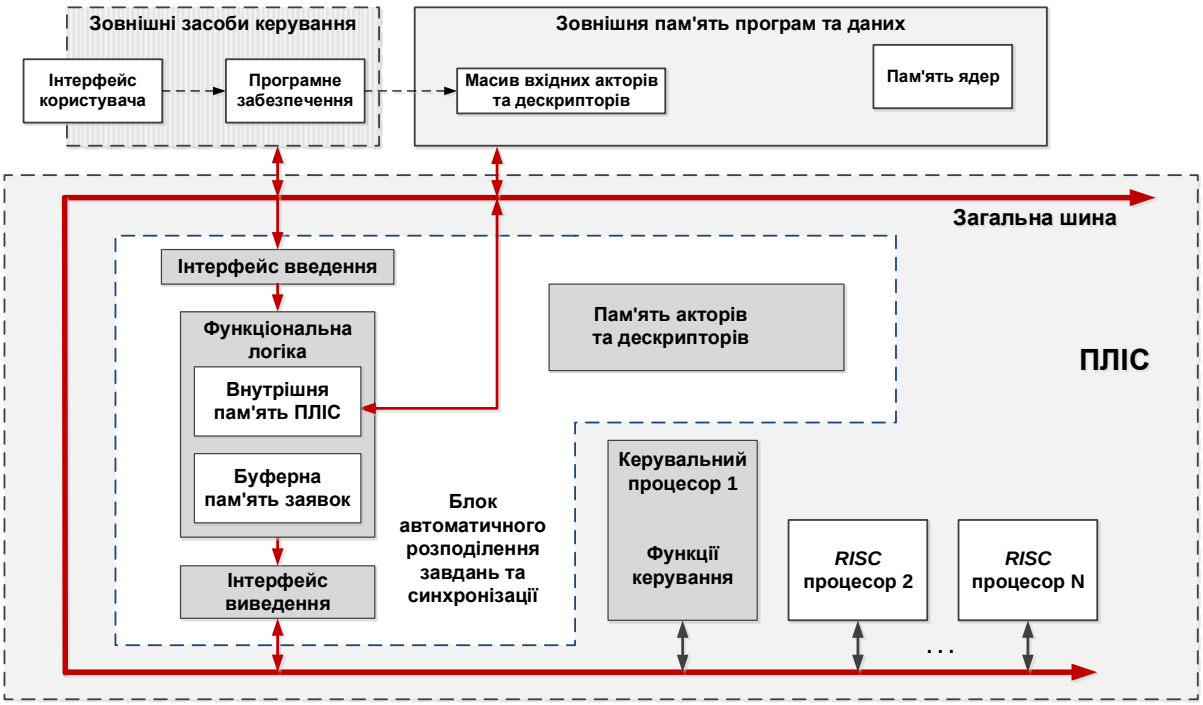
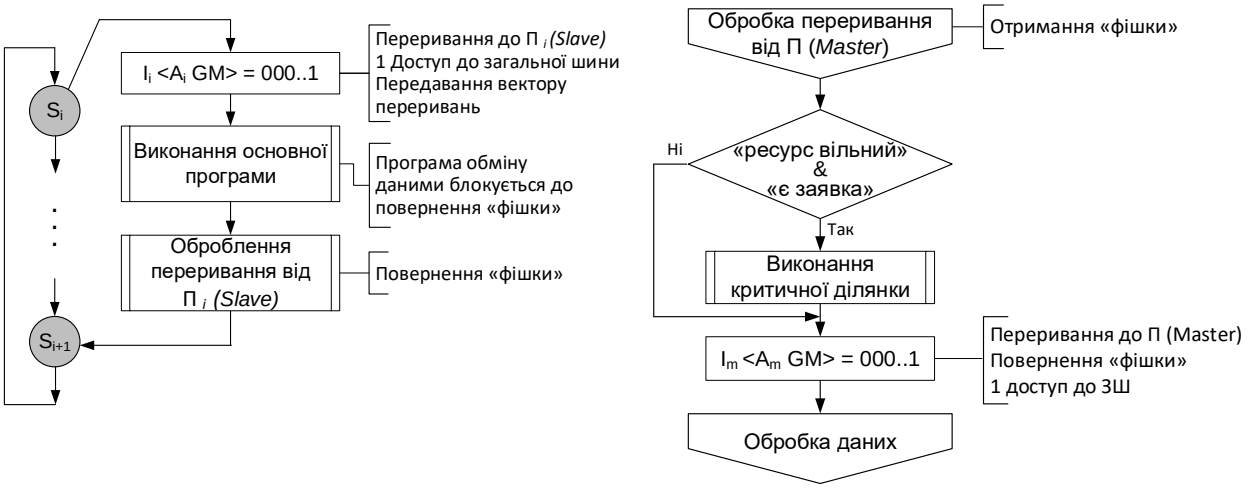


Рис.5.21. Структура обчислювального модуля РКС

5.2.2. Удосконалення засобів автоматичної синхронізації. Для підвищення ефективності обміну даними у запропонованому ОМ РКС у праці [191] вдосконалено спосіб автоматичної синхронізації, описаний у праці [51]. Цей спосіб ґрунтується на пересуванні «фішки», яка інтерпретується доступністю ресурсу S, через вершини S_i графу $(i = \overline{1, (N-1)})$, де П – кількість

процесорів у системі. Якщо «фішка» міститься у вершині S_i і є вимога доступу до загального ресурсу від процесора P_i, виконується операція виділення ресурсу. Процесор отримує доступ до виконання критичної ділянки, після виконання якої повертає «фішку», звільнюючи ресурс S_i. Функції автоматичної синхронізації процесів виконує розроблений ПАРС (рис. 5.21). Програма синхронізації виконується засобами керувального процесора. Програмний режим опитування спричиняє витрати продуктивності, пов'язані з безперервною перевіркою наявності «фішки» навіть із застосуванням комунікаційної пам'яті. За реалізації керування потоком даних у централізованій системі, коли підлеглі процесори рівноправні і лише обробляють дані, необхідно додатково синхронізувати сигнали вимоги доступу до загального ресурсу і готовності заявки на виконання, інакше заявка може бути виконана іншим процесором, який раніше отримає «фішку» від керувальної програми. Окрім того, реалізація відомого способу потребує наявності локальної пам'яті у складі кожного процесорного модуля, що не передбачено архітектурою засобів проектування *SOPC Builder*. Для вирішення означених проблем запропоновано модифікацію відомого способу автоматичної синхронізації для забезпечення ефективної синхронізації в межах вдосконаленої архітектури. Алгоритм реалізації модифікованого способу зображено на рис. 5.22 *а, б*. Код програми реалізації модифікованого способу автоматичної синхронізації для пристрою автоматичного розподілу завдань та синхронізації наведено в додатку В.2.

Модифікований спосіб забезпечує синхронізацію керувальних сигналів вимоги доступу та наявності заявки і функціонує в режимі керування каналами зв'язку, що запобігає накладним витратам продуктивності процесорів. Обмін керувальними даними вимагає лише двох звернень до загальної шини, при цьому відсутні непродуктивні використання ресурсу процесорів у стані очікування. Реалізація модифікованого способу ґрунтується на застосуванні базової системи переривань процесора *Nios* II *Altera*.



а *б*

Рис. 5.22. Модифікований спосіб автоматичної синхронізації: *а* – алгоритм, виконуваний керувальним процесором; *б* – алгоритм, виконуваний підлеглим процесором; П – процесор

5.2.3. Розроблення архітектури та програмного забезпечення обчислювальної системи на ПЛІС. Обчислювальний модуль РКС розроблено та досліджено на базі ПЛІС *Cyclone II EP2C35F672C6* компанії *Altera* загальною логічною ємністю близько 33 216 логічних комірок та вбудованою пам’яттю ємністю близько 59 кбайт. Використано стандартний потік проектування SOPC *Builder*. Для реалізації обрано процесорні ядра типу *Nios II/s* (*Small core size*), які мають незначні вимоги до обсягу апаратних ресурсів до 4% логічного ресурсу кристала, і високу продуктивність. Для кожного ядра виділено 10 кбайт загальної пам'яті зі складу вбудованої пам’яті *RAM*. Для виконуваних програм виділено 256 кбайт зовнішньої пам’яті *SDRAM*. ОМ РКС з одним процесорним ядром займає 27% логічного ресурсу мікросхеми та 20% вбудованої пам’яті (близько 12 кбайт). Пристроєм автоматичного розподілу завдань та синхронізації керує керувальний процесор, підключений до шини *Avalon*. Пристрій синтезовано мовою *Verilog* як апаратна надбудова керувального процесорного ядра.

Найбільша проблема, що впливає на обчислювальну потужність досліджуваного ОМ РКС, це обмежена ємність внутрішньої пам’яті кристала,

яка не дозволяє розмістити на одному кристалі велику кількість процесорних ядер. Завдяки розміщенню пам’яті процесорів і пам’яті проміжних даних розрахунків на зовнішній пам’яті *SRAM* (*Static Random Access Memory*) значно зменшено задіяну процесорами ємність убудованої пам’яті. При цьому, зважаючи на можливість забезпечення достовірності результатів експериментів, а також на однаковий підхід до всіх досліджуваних архітектур, можна нехтувати негативною складовою часу, яку вносить застосування зовнішньої пам’яті.

Для виконання розрахунків використано вбудовані стандартні апаратні засоби для обчислень з плаваючою точкою (*Floating Point Hardware – FPH*). Проведені порівняльні дослідження в цьому напрямі показали збільшення швидкодії на порядок та підвищення точності результатів.

5.2.4. Моделювання та дослідження часових характеристик обчислювального модуля реконфігуровної обчислювальної системи. Як практична задача для дослідження та експериментів розв’язується актуальна задача галузі керування технічними системами – СЛАР від великої кількості змінних. Як метод для реалізації на паралельних цифрових системах обрано блоковий метод розв’язання СЛАР методом Гауса [105]. Якщо функції, на які розгалужується задача, є достатньо елементарними і обраховуються швидше ніж обмін даними, виникають проблеми зі швидкодією навіть за достатньої кількості процесорів. Для зведення розмірності задачі до відповідності структурі обчислювального середовища ОМ РКС використовується спосіб блокового розпаралелювання алгоритму розв’язання СЛАР методом Гауса. Під блоком розуміють об’єднання декількох рядків в один блок [105]. Матриця розподіляється між процесорними ядрами поблоково відповідно до їх кількості.

Дослідження проводились для різних архітектур ОМ РКС на ПЛІС з кількістю процесорних ядер від трьох до шести. Порівняно дві архітектури: базову архітектуру із загальною шиною, побудовану засобами середовища *SOPC Builder* [146], та вдосконалену архітектуру на базі моделі обчислень, керовану потоком даних [191]. У першому випадку керування розподілом задач

та синхронізація виконуються програмно-апаратними засобами *MailBox* і *Mutex Core*, у другому – засобами розробленого ПАРС.

На діаграмі, показаній на рис. 5.23, зображено загальну позитивну динаміку збільшення ефективності функціонування досліджуваних конфігурацій системи під час виконання обчислень від різної кількості аргументів СЛАР. Це природно для реалізації розгалуження обчислювального процесу. Показник ефективності E розраховується як $E = K_i / N \cdot 100\%$, де K_i – показник пришвидшення обчислення ($K_N = T_N / T_1$), T_N; T_1 – час обчислення; N – кількість процесорних ядер.

Для аналізу інтенсивності приросту ефективності застосуємо показник абсолютного приросту ефективності: $\Delta E_i = E_i - E_{i-1}$, де $i-1$ – показник попереднього вимірювання. Для оцінювання середніх показників приросту ефективності застосуємо усереднені відносно кількості процесорів показники ефективності. Із діаграми, показаній на рис. 5.24, видно, що застосування автоматичної синхронізації надає більш інтенсивного приросту ефективності в середньому на 2%.

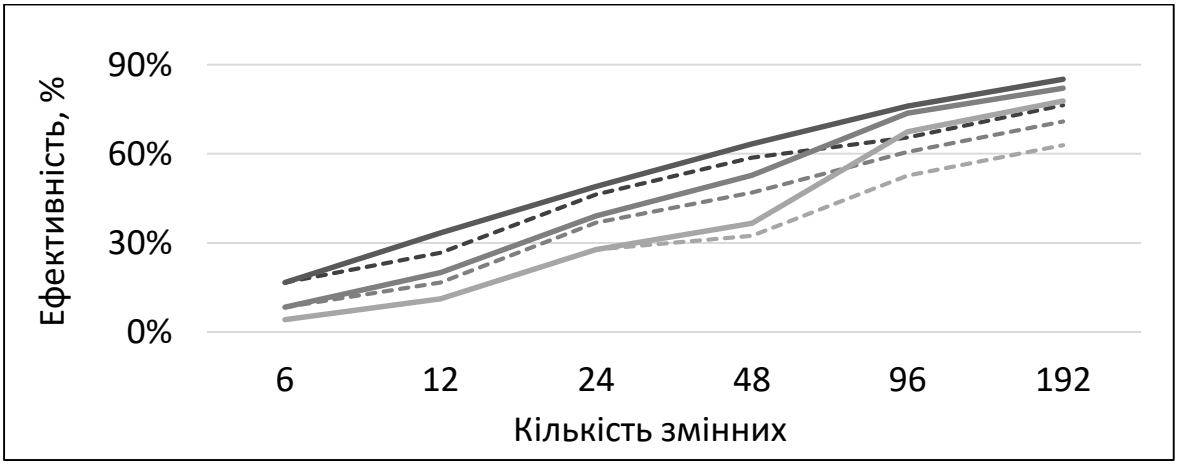


Рис. 5.23. Ефективність традиційних засобів синхронізації в багатопроцесорній системі-на-кристалі: – 3 процесорні ядра; – 4 ядра; – 6 ядер і апаратних засобів синхронізації ПАРС: – 3 ядра; – 4 ядра; – 6 ядер

Як видно, система забезпечує менший приріст ефективності у випадку неефективного використання ресурсів процесорних ядер або переважної кількості операцій обміну даними. За ідеальних умов, коли кількість операцій передавання даних менша або порівняна з кількістю обчислювальних операцій за рівномірного завантаження процесорних ядер, інтенсивність приросту ефективності з використанням автоматичної синхронізації становить 5%.

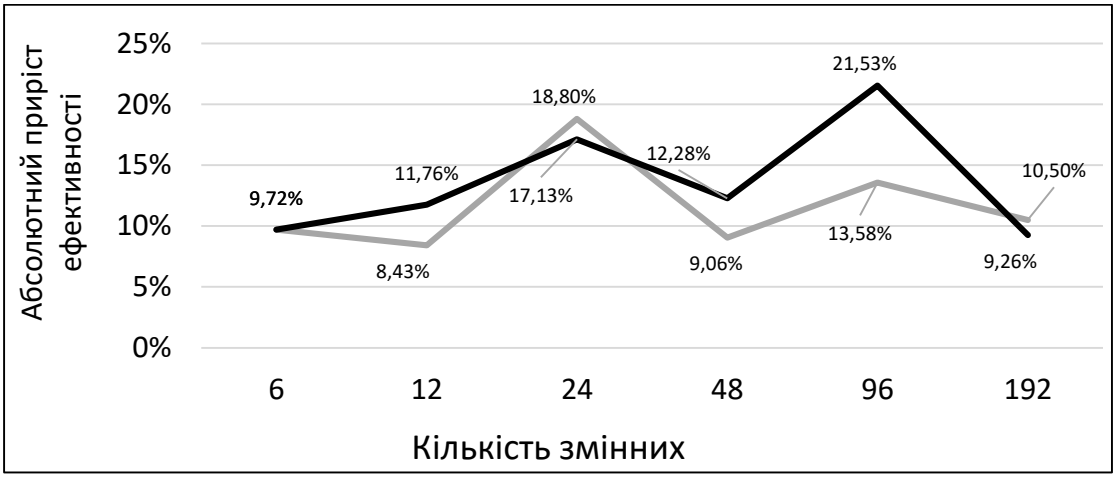


Рис. 5.24. Абсолютний приріст ефективності досліджуваних засобів синхронізації: ▬ – традиційні засоби синхронізації в багатопроцесорній системі-на-кристалі; ▬ – апаратні засоби синхронізації ПАРС

Розглянуто також показники відносного пришвидшення обчислення, розраховані як $K_i = (K_i / K_{i-1}) \times 100\%$. З отриманих графіків залежностей (рис. 5.25) видно, що інтенсивність пришвидшення обчислення із застосуванням засобів автоматичної синхронізації в середньому більша на 65% ніж із застосуванням відомих засобів синхронізації.

Абсолютний приріст ефективності досліджуваних підходів, розрахований за виразом $\Delta E_{AS} = E_{SOPC} - E_{AS}$, зображує позитивну динаміку збільшення ефективності (рис. 5.26), при цьому прослідковуються критичні ділянки, на яких рівень ефективності зменшується, що відповідає збільшенню кількості операцій обміну даними або невідповідності розмірності задачі та кількості процесорних ядер. Відносний показник приросту ефективності,

розрахований за виразом $K_{AS} = (E_{SOPC} - E_{AS}) / E_{AS} \cdot 100$, зображує приріст ефективності із застосуванням запропонованого способу автоматичної синхронізації у середньому на 10,25%.

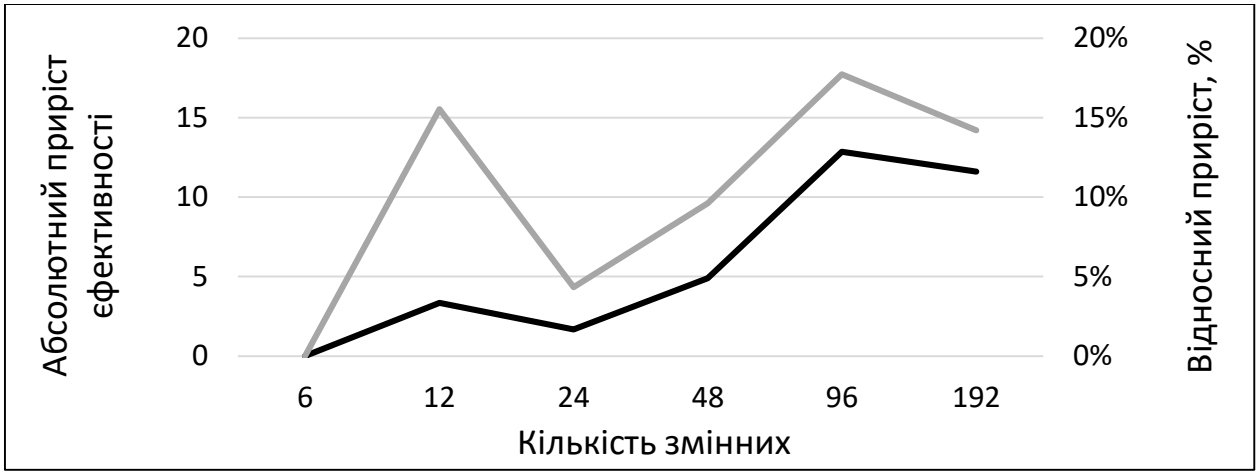


Рис. 5.25. Показники приросту ефективності: : – абсолютний приріст ефективності; – відносний приріст ефективності

Дослідження розроблених засобів показують, що за сумірної кількості процесорів та розмірності задачі спостерігається збільшення показників ефективності. Зі збільшенням кількості процесорів на ефективність впливає ступінь їх завантаження обчислювальними операціями.

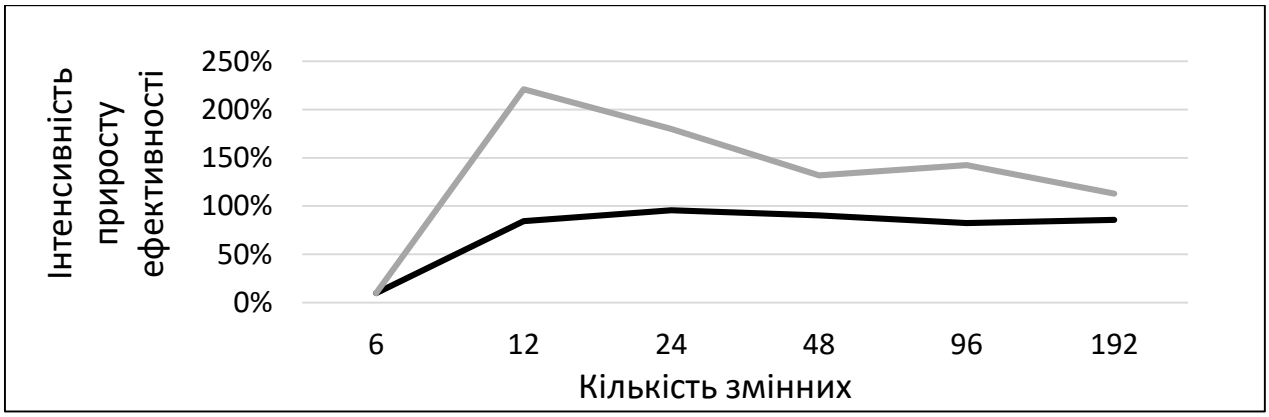


Рис. 5.26. Дослідження інтенсивності приросту ефективності: – традиційні засоби синхронізації в багатопроцесорній системі-на-кристалі; – апаратні засоби синхронізації ПАРС

Абсолютний показник, що зображує динаміку зміни ефективності, зростає на ділянках з ідеальною організацією обчислювального процесу – сумірними параметрами обчислювальної системи та розв’язуваної задачі, і зменшується в противному випадку. При цьому зберігається позитивна динаміка збільшення ефективності на всьому проміжку дослідження. Найбільшим негативним фактором впливу на ефективність у досліджуваній системі є операції обміну даними, коли час їх виконання перевищує час обчислення. Дослідження показують різке зниження ефективності за великої кількості невідомих з причини значного збільшення обсягу оброблюваних даних і відповідно часу їх передавання.

5.3. Спосіб побудови паралельного середовища формування команд у потокових функціональних модулях на ПЛІС

5.3.1. Розроблення структури паралельного середовища формування команд для потокових функціональних модулів. Потокові обчислювачі, що реалізують послідовне формування і розподіл команд у загальному середовищі формування команд, побудовані на базі послідовних СФК, розглянуто у працях [51, 192 – 194,]. Застосування сучасних апаратних методів пришвидшення виконання операцій на швидкодійній елементній базі ПЛІС [34] приводять до того, що інтенсивність формування команд у таких середовищах формування команд стає недостатньою для ефективного завантаження швидкодійного обчислювального середовища. Це зумовлено тим, що час формування команди стає більшим від часу її виконання.

Безпосереднє дублювання СФК для реалізації декількох потоків формування команд ефективне лише у статичних обчислювальних системах, для реалізації такого дублювання в паралельних обчислювальних системах виникає необхідність статичної підготовки програми, а саме: попереднього розділення ГА на підграфи, виділення потоків і ручного призначення акторів та ідентифікаторів потоків [195]. Способи автоматичного формування акторів за наявності декількох середовищ формування команд у потокових обчислювачах

описано у праці [196], які ґрунтуються на послідовному пересиланні даних між обчислювальними модулями, зв’язаними між собою кільцевою організацією потокових обчислень. Недоліком таких систем є затримання на пересилання даних між модулями системи і проблеми відмовостійкості, пов’язані з послідовними зв’язками між модулями системи. Метод автоматичного розподілу акторів і даних, що описано у праці [195], дозволяє автоматично формувати паралельні потоки команд для їх реалізації в паралельному СФК, але даний метод ефективний для графів обчислювальних алгоритмів без розгалуження потоків даних.

Для підвищення ефективності розв’язання задач керування в режимі реального часу, які потребують реалізації дрібнозернистих обчислювальних алгоритмів великої розмірності, а також різної розмірності в багатозадачному режимі обчислень, графи яких характеризуються великою шириною та високим ступенем зв’язності, у праці [27] запропоновано використання потокових функціональних модулів (ПФМ), побудованих на динамічно реконфігуровних ПЛІС, як спеціалізованих обчислювальних вузлів РКС. Пришвидшення обчислень у даному випадку досягається за рахунок використання швидкодійної елементної бази [27, 34], апаратного пришвидшення виконання операцій і розпаралелювання алгоритмів на рівні операцій [51], реалізації прихованого паралелізму, одночасної реалізації декількох алгоритмів без витрат на синхронізацію процесів і розподілення пам’яті [51]. Розширення функціональних можливостей потокових обчислювачів забезпечується шляхом багаторазового динамічного перепрограмування певних ділянок кристалів ПЛІС [13 – 17].

Потоковий функціональний модуль, що складається з керувального ядра і обчислювального середовища, організований за принципом відкритих систем, коли всі функціональні елементи з’єднуються з загальним комунікаційним середовищем засобами однотипних інтерфейсів [18, 27, 51]. Обчислювальне середовище складається з набору реконфігуровних ФБ і регістрів команд (РК) у складі кожного ФБ. Це забезпечує передумови для масштабування ПФМ (рис. 5.27). Керувальне ядро реалізовує модель обчислень, керованих потоком

даних, відповідно до якої здійснює функцію автоматичного розподілу команд в обчислювальному середовищі і керує процесом реконфігурації ПФМ, що виконується за адаптації до розмірності розв’язуваних задач або в разі відмов устаткування.

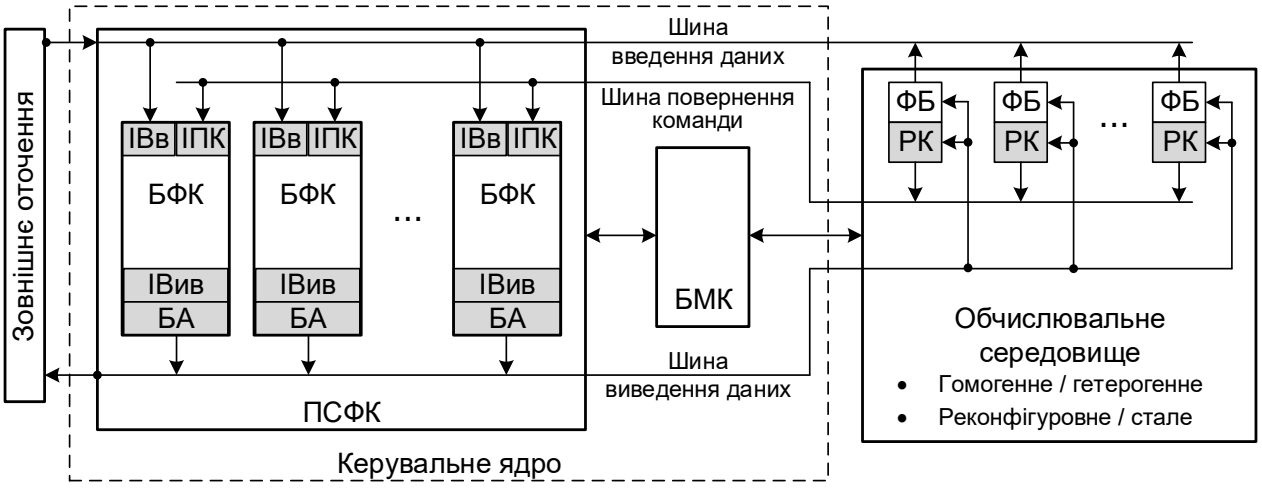


Рис. 5.27. Структура потокового функціонального модуля: блок арбітра, ІВв – інтерфейс введення; ІПК – інтерфейс повернення команди; ІВив – інтерфейс виведення; БА – блок арбітра; РК – регістр команди

У дисертаційній роботі запропоновано спосіб побудови паралельного середовища формування команд (ПСФК) на базі модифікації відомого методу паралельного формування потоків команд для традиційних потокових обчислювальних систем, що допускає галуження потоків даних [196]. Запропонована модифікація створює передумови для реконфігурації структури ПСФК за адаптації до розмірності розв’язуваних задач або в разі відмов устаткування.

Паралельне середовище формування команд складається з h однотипних блоків формування команд (БФК). До складу кожного з яких входять: блок буферної пам'яті даних (БПД), блок буферної пам’яті команд (БПК), регістр даних, регістр адреси, пам'ять операндів, пам'ять керувальних слів, блок арбітра, інтерфейс введення, інтерфейс повернення команди, інтерфейс виведення, блок керування формуванням команд (БКФК) (рис. 5.28).

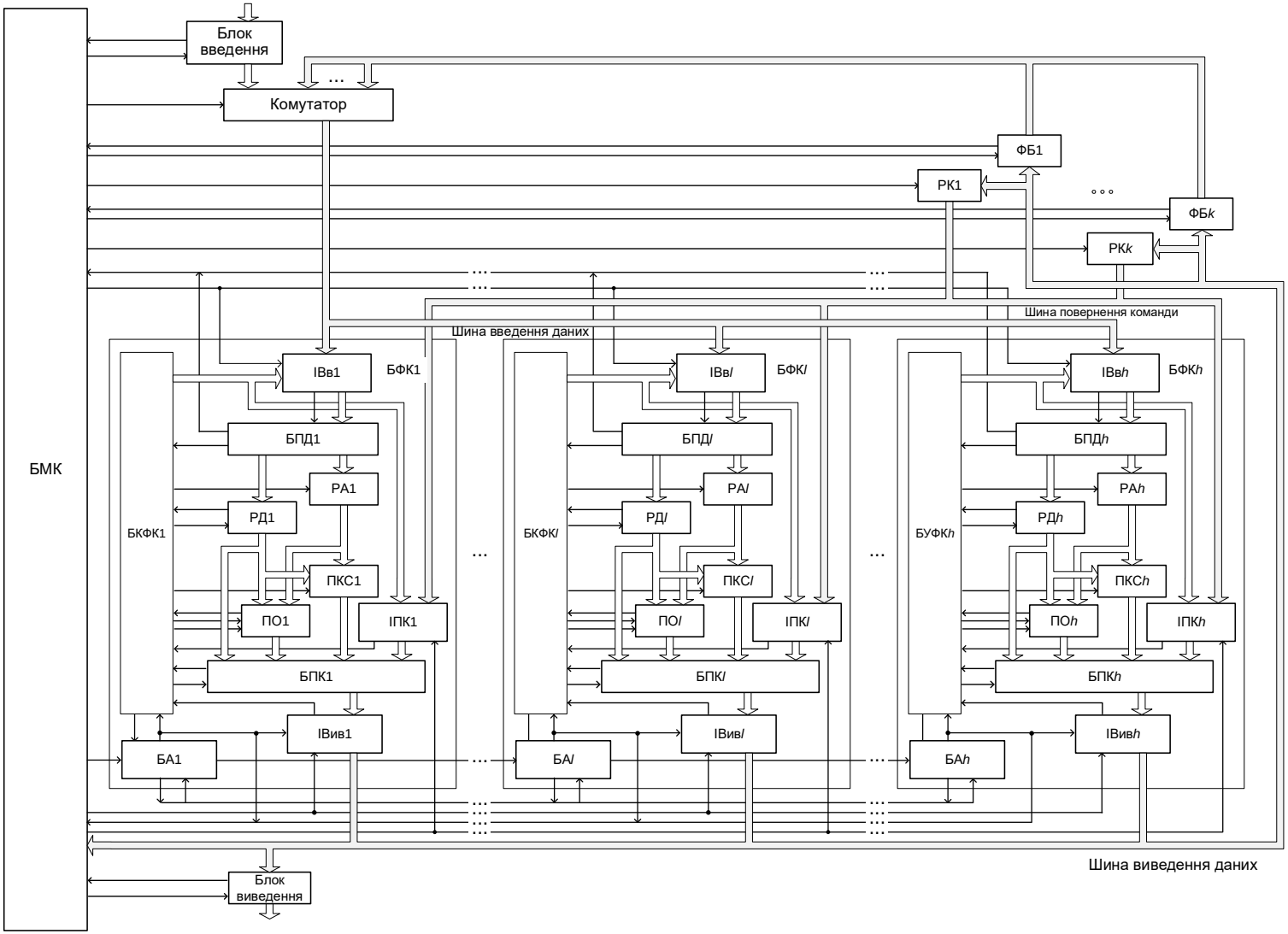


Рис. 5.28. Структурна схема потокового обчислювального модуля з паралельним середовищем формування команд

Реконфігурація структури ПСФК стосується адаптації його розмірності (кількості БФК) до кількості паралельних потоків формування команд, що визначається структурою ГА обчислювальної задачі. Це можливо на підставі застосування принципу відкритих систем [27, 51]. У склад інтерфейсу кожного БФК входять блоки інтерфейсів введення даних та виведення команди, які сполучають кожний БФК із загальними шинами введення даних та виведення команди відповідно, і блок інтерфейсу повернення команди, через який БФК сполучається із загальної шиною повернення команди. ФБ у структурі загального обчислювального середовища зв'язані з інтерфейсами введення всіх БФК через загальну шину введення даних (рис. 5.28).

Таким чином, для розв'язання задач синхронізації паралельних процесів формування команд і доступу до загальних системних ресурсів розроблено таку структуру інтерфейсів БФК.

Для розв'язання задач синхронізації паралельних процесів формування команд розроблено структуру інтерфейсу введення даних БФК, структуру інтерфейсу повернення команди та структуру інтерфейсу виведення команди (рис. 5.29 і 5.30).

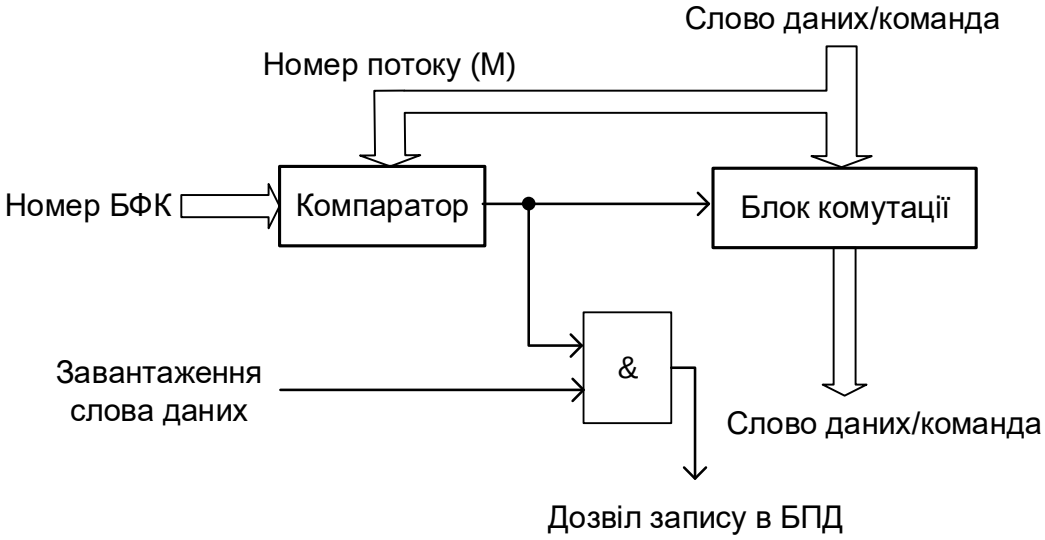


Рис. 5.29. Структура інтерфейсу введення даних та інтерфейсу повернення команди

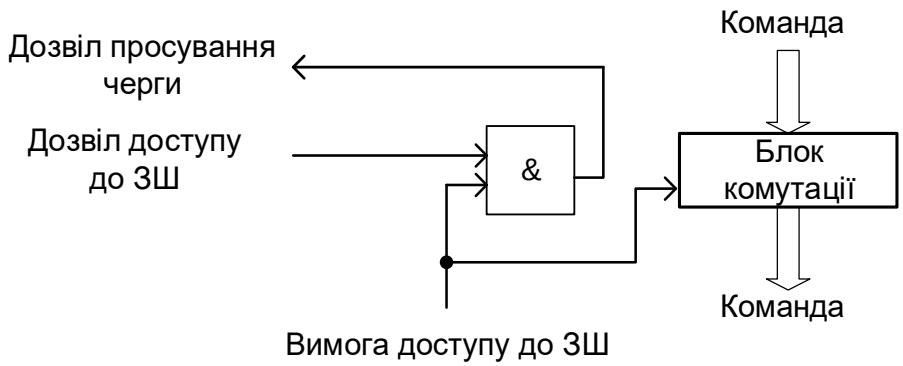


Рис. 5.30. Структура інтерфейсу виведення команди: ЗШ – загальна шина

Для забезпечення синхронізації доступу до загального обчислювального середовища у склад інтерфейсу виведення даних кожного БФК входить блок розподіленого арбітра загальної шини (рис. 5.31). Принцип функціонування розподіленого арбітра загальної шини наведено у працях [51, 189]. У склад кожного блока арбітра входить блок керувального автомата, граф функціонування якого наведено у праці [51, рис. 5.11].

Засоби синхронізації роботи БФК у ПСФК забезпечують синхронізацію на рівні апаратних засобів без участі централізованих і децентралізованих засобів керування обчислювальним процесом.

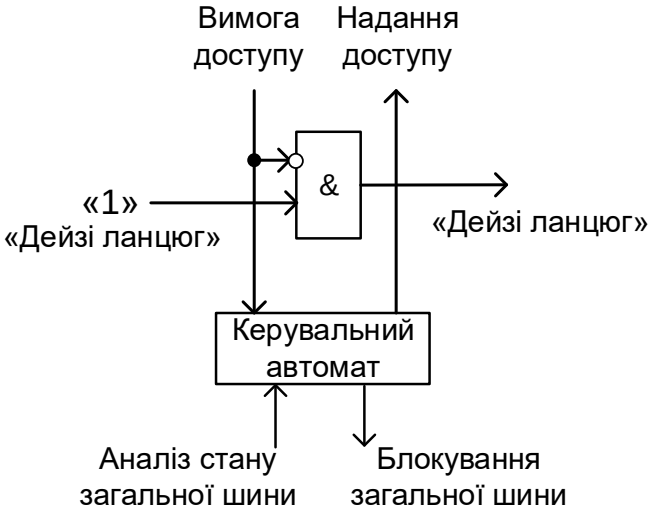


Рис. 5.31. Структура блока розподіленого арбітра доступу до загальної шини

Принцип формування команди в кожному БФК, а також формат керувального слова (актора) і слова операнда (дескриптора) описано у працях [196, 197]. Керувальне слово (актор), містить q-розрядне поле коду операції, розряд номера обчислюваного операнда (дескриптора), m-розрядне поле номера операції (актора), f-розрядне поле номера наступної операції (наступного актора), розряд ознаки типу інформації, де q, m, f – цілі числа. Операнд (дескриптор) містить поле номера обчислюваного операнда, m-розрядне поле номера операції, розряд ознаки типу інформації, поле значення операнда.

Для визначення кількості потоків формування команд використовується формальний спосіб розподілу потоків команд, запропонований у праці [196].

Для реалізації ПСФК у склад керувального слова і слова даних, на відміну від відомого [196, 197], введено d-розрядні поля номера потоку L_u і один розряд b ознаки типу даних, яка для всіх керувальних слів має значення «1», для слів операндів – значення «0» (рис.5.32, рис. 5.33). Інформація, що записується в інші поля керувального слова і слова даних, визначається заданим графом обчислювальної задачі і не залежить від кількості БФК.

Номер потоку, d	Номер операції/ актора, m	Код операції, q	Наступна операція/актор		Номер операнда	Тип даних
			Номер потоку, d	Номер операції, f		
L_u	N_i	I_j	L_r	N_v	n	$B_i = 1$

Рис. 5.32. Формат керувального слова (актора)

Розрядність d поля номера потоку визначається кількістю потоків паралельних обчислень p і дорівнює $]\log_2 p[$. Значення номерів потоків L_u і L_r визначають відповідно належність операції і наступної операції до визначеного потоку паралельних обчислень $(u, r = \overline{1, p})$ [196].

Призначення операції/актора		Номер операнда	Значення операнда	Тип даних
Номер потоку, *d*	Номер операції/ актора, *m*			
L_r	N_v	n	X_{vn}	$b_v = 0$

Рис. 5.33. Формат слова даних (дескриптора)

5.3.2. Розроблення алгоритмів керування формуванням команд у потоковому функціональному модулі. Під час функціонування ПФМ можна виокремити два етапи процесу керування.

1. Централізований, керований блоком мікропрограмного керування (БМК):

– уведення інформації в блок буферної пам'яті даних з блока введення даних і з обчислювального середовища, що складається з функціональних блоків ФБ1...ФБ*k* (у випадку справної роботи ФБ1...ФБ*k*) (рис. 5.34);

– розподіл команд між функціональним блоками ФБ1...ФБ*k* (для подальшого оброблення) і блоком виведення даних (рис. 5.35).

1. Децентралізований, що відбувається в кожному БФК, керованому блоком керування формуванням команди: процес формування команди і занесення її в чергу на виконання у блок буферної пам'яті команд.

Децентралізований процес керування формуванням команд у кожному БФК відбувається незалежно від системного оточення, зокрема від кількості БФК і структури обчислювального середовища. Децентралізований процес формування команди і занесення її в чергу на виконання у блок буферної пам'яті команд описано у працях [196, 197].

Процес введення даних у БФК за нормальної роботи ПФМ, коли ФБ є в працездатному стані, ілюструється алгоритмом, показаним на рис. 5.34.

Для кожного з блоків ФБ1…ФБ*k* використовується таймер, який формує два сигнали, один з яких відповідає інтервалу часу t_1, а другий – більшому інтервалу часу t_2.

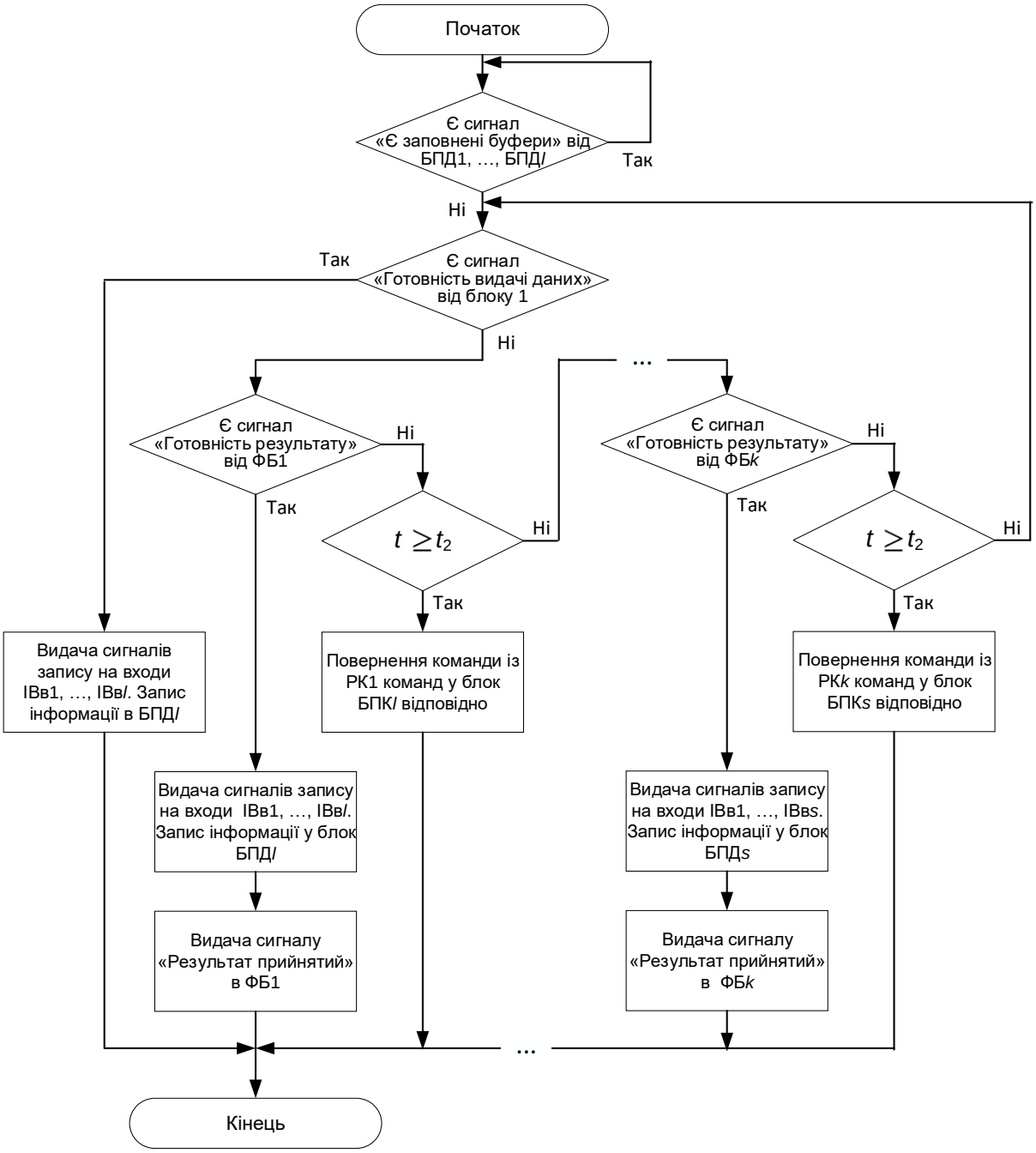


Рис. 5.34. Алгоритм керування введенням даних у блок формування команди: ІВв – інтерфейс введення

Алгоритм керування введенням даних у БФК реалізує БМК, який аналізує сигнал «Є заповнені буфери», що надходить на його керувальний вхід

загальною лінією від БПД1…БПД*h*, і визначає, що один або декілька БПД на своєму керувальному виході виставили сигнал «Буфер зайнятий», сигналізуючи про наповненість. Без указаного сигналу, коли в блоки БПД можна записати інформацію, блок БМК перевіряє сигнал на виході ознаки даних блока введення даних і сигнали на виходах ознаки завершення операції блоків обробки даних ФБ1...ФБ*k*. Наявність першого із сигналів свідчить про те, що блок уведення даних прийняв дані з інформаційного входу ПФМ. У цьому випадку за керувальними сигналами БМК інформація з блока введення через комутатор надходить на інформаційні входи всіх інтерфейсів уведення. Блок мікропрограмного керування видає сигнал запису слова даних на керувальні входи запису даних всіх інтерфейсів введення. Слово даних буде записане тільки в той блок БФК, який призначений для виконання *g*-го потоку паралельних обчислень. Слова даних завантажуються в БФК відповідно до номера потоків паралельних обчислень, що закодовані в полі *L* слова даних (див. рис. 5.32 і 5.33); при цьому для кожного значення номера потоку визначено відповідний номер БФК.

Якщо кількість БФК дорівнює 2^w, то вони нумеруються *w*-розрядними кодами і до інформаційних входів компараторів інтерфейсів уведення (див. рис. 5.29) підключено *w* молодших розрядів поля *L* номера потоку, що надходить з комутатора. Аналогічним чином *w* молодших розрядів поля *L* номера потоку слова команди, що надходить на повторне виконання з регістрів команд, підключені до інформаційних входів компараторів інтерфейсів повернення команди (див. рис. 5.29).

Підключення кожного БФК до складу оточення під час розподілу і завантаження слів даних здійснюється через інтерфейс уведення. Номер БФК формується у програмному режимі на інформаційному виході БКФК і надходить на інформаційний вхід компаратора, який за збігу інформації на першому і другому інформаційних входах установлює на керувальному виході сигнал «Вибір блока», що надходить на вхід логічного елемента І блока інтерфейсу введення і на керувальний вхід «Дозвіл підключення» блока

комутації, що комутує слово даних через інформаційний вихід блока інтерфейсу введення на інформаційний вхід блок буферної пам’яті даних. Із БМК надходить сигнал запису слова даних на другий вхід логічного елемента І, який за наявності сигналу «Вибір блока» встановлює на керувальному виході інтерфейсу введення керувальний сигнал запису слова даних, за яким слово даних завантажується в БПД. За відсутності сигналу «Вибір блока» сигнал «Запис даних» від БМК блокується логічним елементом І інтерфейсу введення.

Аналогічним чином відбувається запис у БПД інформації із ФБs (де $s=\overline{1,k}$) у випадку готовності результату. В останньому випадку в ФБ.j передається сигнал «Результат прийнятий» із БМК і ФБs знімає сигнал ознаки завершення операції. Для запобігання ситуації, коли один або декілька блоків БПД заповнені, що може призвести до невизначеного часу очікування можливості запису, використовуються алгоритми з резервуваннями комірок пам’яті для запису результатів обчислення [51].

Якщо ФБ.s не виставляє сигнал ознаки завершення операції, то БМК перевіряє умову ($t > t_2$), де t_2 – інтервал часу, за який результат повинен бути отриманий обов'язково, якщо ФБs не вийшов з ладу. Коли ФБs працює правильно, тобто умова ($t > t_2$) не виконується, перевіряється сигнал ознаки завершення операції в наступному ФБ.(s+1).

Якщо в блоці БПК є хоча б одна готова для виконання команда, з керувального виходу вимоги доступу блока керування формуванням команди на керувальний вхід відповідного блока арбітра видається керувальний сигнал «Вимога доступу», який розриває «дейзі-ланцюжок», тимчасово відключаючи блоки арбітрів усіх інших блоків формування команд, запобігаючи підключення до загальної шини. Блок арбітра підключеного до загальної шини даних (див. рис. 5.28) із надходженням сигналу «Вимога доступу» аналізує наявність одиничного сигналу на керувальному вході «дейзі-ланцюжка» і сигналу «Шина вільна» на керувальному вході загальної лінії зайнятості шини

даних. У разі їх встановлення виставляє на керувальному виході сигнал «Надання доступу» і блокує загальну шину сигналом «Блокування шини». Інакше, за відсутності хоча б одного із зазначених сигналів, блок арбітра переходить у стан очікування звільнення загальної шини. Керувальний сигнал «Надання доступу» надходить на відповідний керувальний вхід БКФК на лінію готовності команди, що підключена до відповідного керувального входу готовності команди БМК, і на керувальний вхід дозволу видачі команди інтерфейсу виведення даних (див. рис. 5.30), який при цьому комутує слово команди зі свого інформаційного входу на інформаційний вихід, зв’язаний із загальною шиною даних.

Якщо команда обчислювальна, передавання її у блок ФБ*s* здійснюється відповідно до алгоритму, наведеному на рис. 5.35.

Якщо на керувальному вході готовності команди БМК установлений сигнал «Є готові команди для виконання», то блок БМК аналізує поле типу даних на загальній шині даних, що підключена до його інформаційного входу, і забезпечує передачу інформації в блок виведення даних або в один із ФБ*s* шляхом формуванням відповідних керувальних сигналів.

Готовий до приймання команди ФБ*s* після перевірки працездатності вбудованими схемами контролю або тестом видає в блок БМК сигнал «Готовність прийому команди». Блок мікропрограмного керування по черзі аналізує готовність кожного ФБ1...ФБ*k*, здатного виконати дану команду, по черзі аналізується БМК, який готовому ФБ*s* передає керувальний сигнал «Прийняти команду». Команда із загальної шини даних завантажується у ФБ*s* для подальшого виконання. Одночасно з цим за відповідними сигналами БМК команда завантажується в регістр команди РК*s*.

Далі БМК запускає таймер ФБ*s* на інтервали часу t_1 і t_2, а ФБ*s* скидає сигнал «Готовність прийому команди» і починає виконання своєї команди. Після зняття сигналу «Готовність прийому команди» БМК формує сигнал «Просунути чергу» на загальну лінію, що надходить на керувальні входи просування черги інтерфейсів виведення, в одному з яких за наявності сигналу

«Надання доступу» від арбітра формується сигнал просування, що надходить на відповідний керувальний вхід БКФК. Інакше, якщо сигналу «Надання доступу» не було, інтерфейс виведення блокує сигнал «Просунути чергу» від БМК.

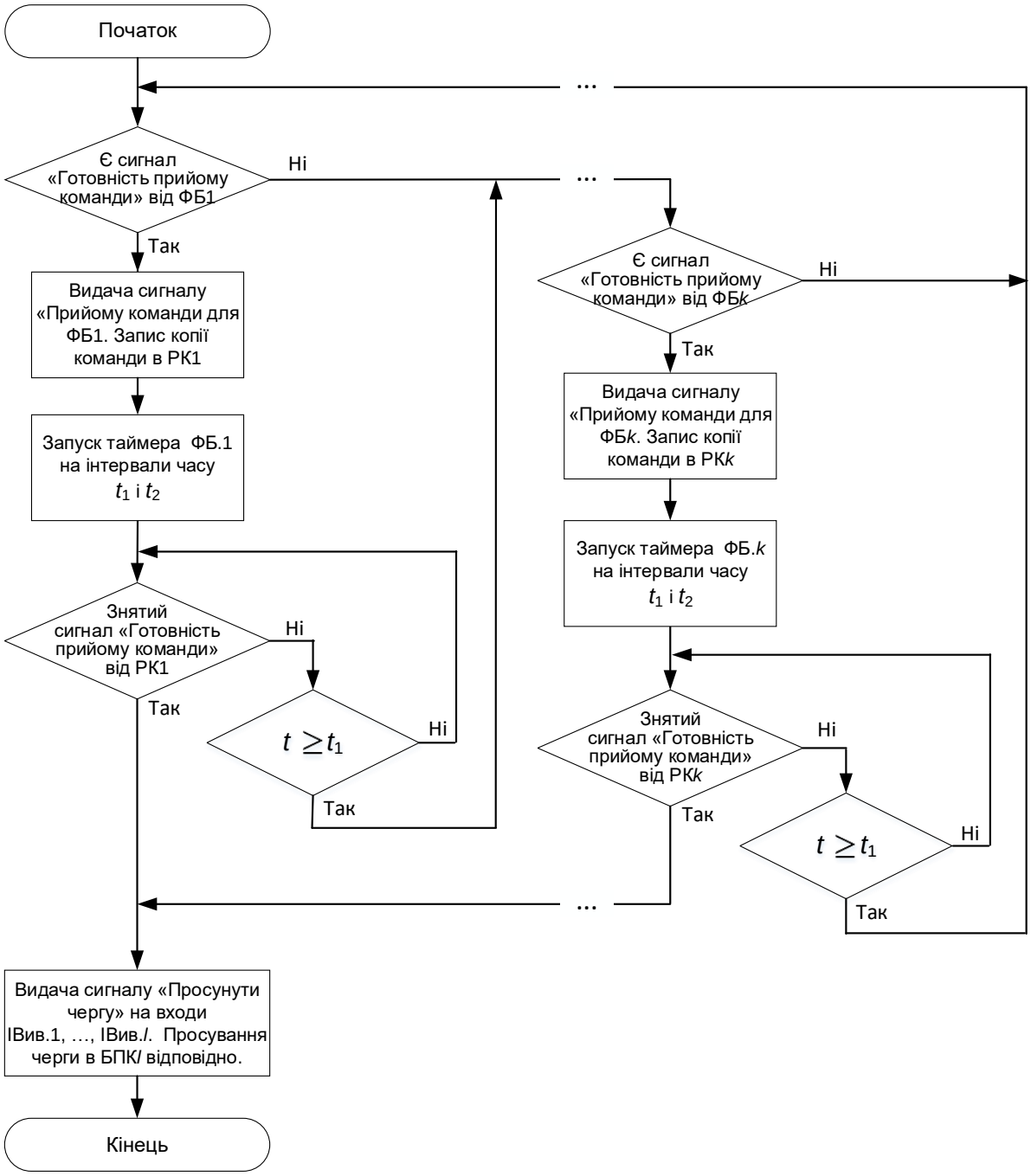


Рис. 5.35. Алгоритм розподілу команд між функціональними блоками і блоком виведення даних: ІВив – інтерфейс виведення

Отримавши сигнал «Просунути чергу», БКФК видає сигнал просування черги на керувальний вхід БПК і скидає сигнал вимоги доступу до арбітра, який у свою чергу знімає сигнали «Блокування шини» і «Надання доступу» на відповідних виходах із загальної лінії готовності команди. Це відключає БФК від загальної шини. Описаний процес формування команди відбувається в усіх БФК, а інтерфейси виведення забезпечують підключення їх до загальної шини для видачі чергової готової команди.

У процесі оброблення команди кожен ФБ*s* обмінюється з БМК двома вхідними і двома вихідними керувальними сигналами. Одна пара сигналів (вхідний і вихідний) використовується для введення команди, а друга для видачі результату після виконання команди.

У роботі ПФМ може виникнути збій в роботі одного або декількох ФБ. Збої можуть виникати після виставлення сигналу «Готовність приймання команди», або після його зняття. Для визначення несправності ФБ використовуються алгоритми, описані у працях [196, 197]. Виконання цих алгоритмів приводить до необхідності повторення команди, що повинна була виконуватися або виконувалася в несправному ФБ в середовищі формування команд для повторного виконання.

Завдання визначення, у якому саме БФК була раніше сформована команда і повернення її саме в той БФК вирішується на рівні інтерфейсів повернення команди на підставі значення поля *M* слова команди (див. рис. 5.32 і 5.33).

Повторний запис команди в БФК здійснюється за допомогою сигналів «Видати дані з РК*s*» і «Повернути команду», які надходять із БМК відповідно на керувальний вхід РК*s* і на загальну лінію повернення команди. Надходження сигналу «Видати дані з РК*s*» забезпечує видачу слова команди на загальну шину повернення команди, що сполучає блоки РК1…РК*k* і інформаційні входи інтерфейсів повернення команди. Керувальний сигнал «Повернути команду» із загальної лінії повернення команди надходить на керувальні входи всіх інтерфейсів повернення команди. В інтерфейсі

повернення команди в одному з БФК у разі збігу значення поля *M* номера потоку слова команди (рис. 5.32, рис. 5.33) і номера БФК слово команди комутується інтерфейсом повернення команди і надходить на інформаційній вхід БПК. Сигнал «Повернути команду» з керувального виходу інтерфейсу надходить на керувальний вхід запису поверненої команди БКФК.

Таким чином, у разі відмови будь-якого ФБ відбувається повернення команди у відповідний номер потоку паралельних обчислень БФК, у якому вона була сформована. Невиконана команда повторно записується в буфер готових команд незалежно від інформації, наявної в БФК у даний момент часу і буде виконана в працездатному ФБ.

Оцінимо ефективність ПФМ з ПСФК за способом оцінювання продуктивності обчислювальних систем, що виконується на стадії проектування і використаний у праці [197]. Порівняємо ПФМ з різною кількістю БФК за усередненим часом виконання команд, розрахованим за виразом $T_{\text{К}} = \sum_j P_j t_j$, де t_j – час виконання команди *j*-го типу; P_j – частота команди *j*-го типу в програмі. Оцінювання ефективності проведено на підставі моделювання тестових обчислювальних задач, поданих бінарними деревами, в імітаційній моделі ПФМ з ПСФК. Як часові параметри команд прийнято час виконання команд, змодельований на реальних апаратних моделях ПФМ, розроблених на ПЛІС у праці [135]. Отримані на підставі моделювання тестових задач залежності коефіцієнта відносного пришвидшення обчислень від кількості БФК і кількості ФБ зображено на рис. 5.36. Коефіцієнт відносного пришвидшення обчислень розрахований за таким виразом [197]:

$$T_{\text{П}} = \frac{T_1 - T_h}{T_1} \cdot 100\%,$$

де T_1 – час виконання обчислень у ПФМ з одним БФК, T_h – з *h* БФК.

З отриманих графіків видно, що зі збільшенням кількості БФК час виконання алгоритму змінюється за логарифмічним законом. Характер кривих залежить, по-перше, від співвідношення кількості БФК та кількості ФБ, що у

випадку реалізації реконфігуровного обчислювального середовища обмежено обсягом апаратних ресурсів кристалів ПЛІС, по-друге – від ширини ГА, поданого ЯПФ, що визначає ступінь паралелізму обчислювальної задачі. Із діаграм видно, що навіть застосування двох БФК надає пришвидшення обчислення близько 40%. Ефективність запропонованих апаратних рішень підвищується пропорційно збільшенню кількості БФК і ширини графу обчислювального алгоритму, але таке збільшення ефективності обмежується можливостями обчислювального середовища і непродуктивними витратами часу, зумовленими затриманнями під час передавання даних, синхронізації у процесі доступу до загальних ресурсів і процесом реконфігурації ПСФК та обчислювального середовища. Отримані графіки також зображують взаємну залежність структури алгоритму обчислювальної задачі і структури обчислювального середовища. Використання чотирьох БФК у складі ПСФК пришвидшує обчислення в середньому більш ніж на 60%.

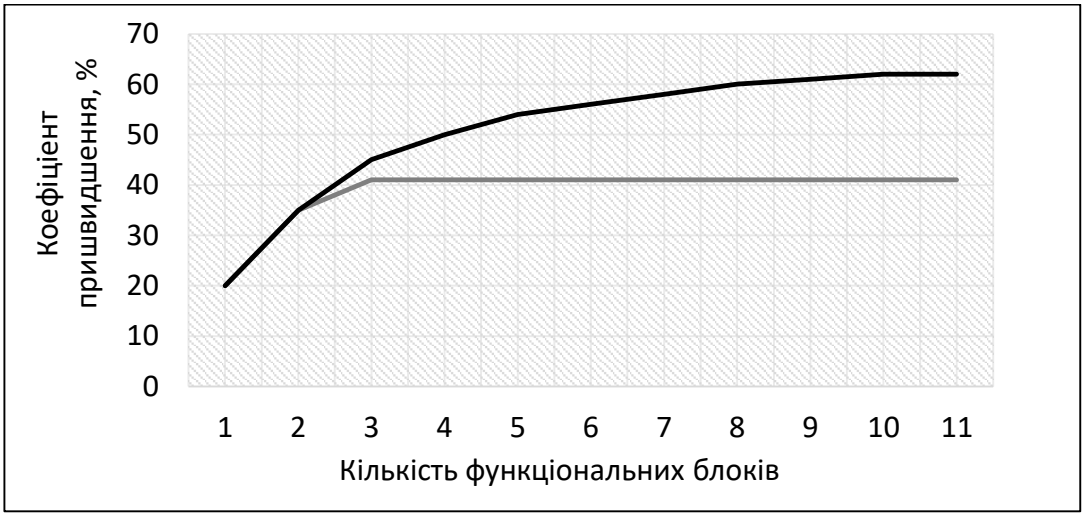


Рис. 5.36. Залежність коефіцієнта відносного пришвидшення обчислень від кількості БФК і кількості ФБ: – два БФК; – чотири БФК

Отримані графіки і розрахунки в цілому збігаються з результатами моделювання методу паралельного формування потоків команд в обчислювальних системах загального призначення [197].

5.4. Спосіб забезпечення відмовостійкості потокових функціональних модулів на ПЛІС

5.4.1. Спосіб автоматичної реконфігурації потокових функціональних модулів на ПЛІС. Для систем керування, що функціонують у режимі реального часу, ефективними є програмно-апаратні методи динамічної реконфігурації систем під час відмови устаткування [51]. Ці методи ґрунтуються на введенні певної апаратної надмірності для забезпечення відмовостійкості систем. У працях [51, 198] описано потокові обчислювальні системи, побудовані на однотипних обчислювальних модулях, та спосіб забезпечення відмовостійкості, заснований на принципі побудови високонадійного ядра на базі послідовного СФК. Під час формування команди не враховується кількість ФБ і, таким чином, створюються передумови продовження обчислень доти, доки не залишиться хоча б один працездатний ФБ.

У працях [192 – 197, 199] описано потокові обчислювальні системи, побудовані на однотипних обчислювальних модулях з різною структурою СФК, у працях [162, 163] – реконфігуровні потокові обчислювальні модулі з гетерогенним реконфігуровним обчислювальним середовищем на ПЛІС. Для всіх потокових обчислювальних систем основною проблемою забезпечення відмовостійкості є відновлення інформації про команду, втрачену через відмову устаткування.

На підставі аналізу різних способів формування команд у СФК, виконаного у праці [200], визначено, що зменшення устаткування і пришвидшення циклів звернення до пам’яті під час формування команд досягається за рахунок адресного читання і запису даних у пам’ять з використанням пам’яті з довільним доступом (ПДД) на відміну від асоціативного читання/запису [51] з використанням асоціативної пам’яті.

У праці [199] описано потоковий обчислювальний пристрій з ПДД, структурну організацію якого наведено на рис. 5.37.

У праці [200] проаналізовано особливості і виявлено недоліки процесу активації команди в цій системі. Команда починає формуватися в комірках ПДД, два розряди яких (*d* і *a*) є ознаками, що показують наявність у ПДД одного операнда D_1 та актора *A*. Запис указаних об'єктів для *i*-ї команди відбувається за адресою I_i. Із надходженням із комунікаційного середовища (КС) 1 другого операнда D_2 для *i*-ї команди він записується безпосередньо у відповідні розряди регістра буферної пам'яті типу *FIFO*, куди одночасно із ПДД надходять *A* і D_1.

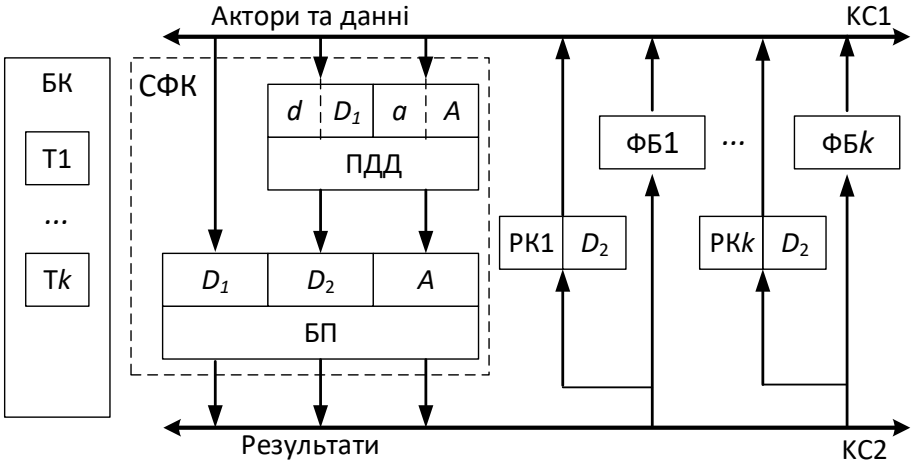


Рис. 5.37. Організація ПФМ (перший спосіб): БП – буферна пам’ять; БК – блок керування; D_1 і D_2 – поля даних; *A* – поле актора; Т – таймер

Таким чином, у буферній пам’яті формується готова для виконання команда, яка через КС2 передається до вільного ФБ. При цьому вміст відповідної комірки ПДД зберігається для забезпечення повторного формування команди в разі відмови ФБ. Одночасно з надходженням команди $\langle A, D_1, D_2 \rangle$ у ФБ*s* її частина D_2, що не зберігається у ПДД, записується в регістр тимчасового зберігання команди РК*s* на вхідній шині відповідного обчислювача. За успішного виконання операції, тобто коли час її виконання не перевищив ліміт часу виконання команди у ФБ*s*, установлений відповідним таймером Т*s* блока керування, результат операції надходить через КС1 у ПДД за адресою N_i, а вміст відповідної комірки ПДД за адресою I_i видаляється.

Якщо у процесі виконання операції відмовив ФБ*s*, тобто ліміт часу виконання операції, заданий відповідним Т*j*, перевищений, ФБ вважається несправним і блокується (відключається від КС1 і КС2), а значення D_2 з РК*s* знову передається у СФК. Оскільки значення A і D_1 не втрачені (відповідну комірку ПДД не змінено), то команда повторно записується у буферну пам’ять, що дає змогу реалізувати наступну успішну спробу виконання команди в дієздатному ФБ.

До недоліків описаного методу варто віднести необхідність зберігання компонентів команди в комірках ПДД до повного завершення виконання цієї команди, у тому числі з урахуванням часу реконфігурації системи у випадку відмови ФБ та повторного виконання команди. Це сповільнює обчислення, якщо відповідну комірку пам'яті необхідно використовувати для інших команд, наприклад, для чергової ітерації або реалізації алгоритму в конвеєрному режимі.

Для усунення цього недоліку у праці [193] пропонується в РК записувати не тільки операнда D_2, а і всю команду із БПК (рис. 5.38). При цьому послідовність формування команди залишається незмінною [199].

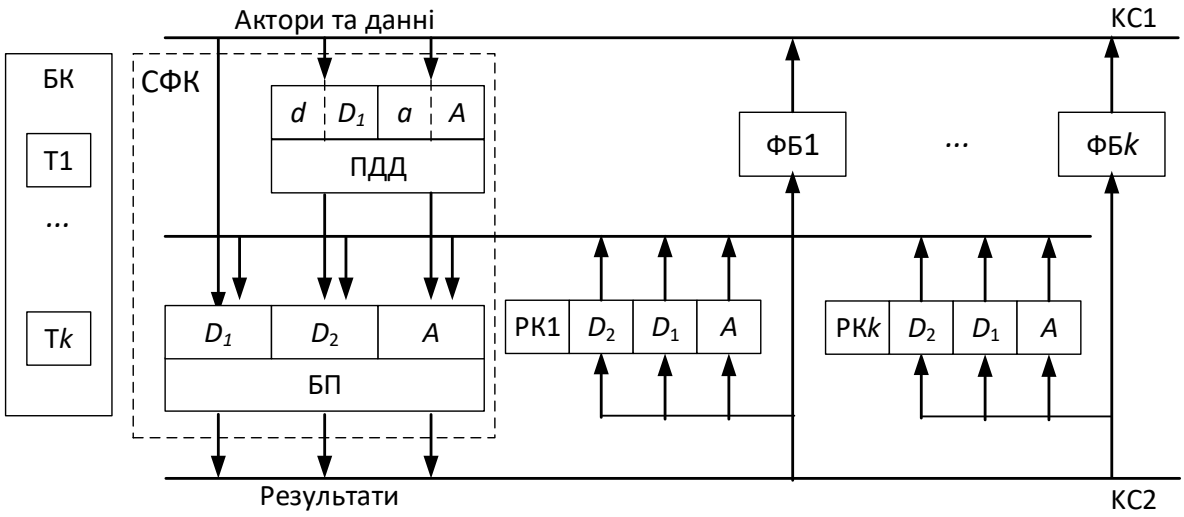


Рис. 5.38. Організація ПФМ (другий спосіб): БП – буферна пам’ять; D_1 і D_2 – поля даних; A – поле актора; Т – таймер

Після запису даних у буферну пам’ять звільняється місце в ПДД для запису наступних даних. За аналогією з попереднім варіантом у разі відмови

ФБ*s* він буде блокований, а дані з РК*s* повернені безпосередньо у БПК, що дасть можливість реалізувати наступну спробу виконання команди.

За такого підходу ресурси системи використовуються більш ефективно. Немає потреби зберігати фрагменти команди у СФК. Комірки ПДД можуть використовуватися для інших алгоритмів або інших ітерацій. Це у свою чергу дозволяє поєднувати обчислювальні процеси на рівні не тільки команд, але й алгоритмів, що забезпечує скорочення витрат часу на перетворення інформації.

Проте з точки зору витрат часу на реконфігурацію системи у випадку відмови ФБ цей підхід є неефективним, оскільки час очікування спрацьовування таймера може бути набагато більшим ніж час, необхідний для виконання команди. Це вносить додаткове затримання часу при реконфігурації системи у разі відмов устаткування.

Вирішення означеної проблеми запропоновано у праці [192]. Формат команди, що передається у ФБ, перетворюється до вигляду $\langle A, D_1, D_2, T \rangle$, де T – час очікування результату виконання конкретної команди (рис. 5.39).

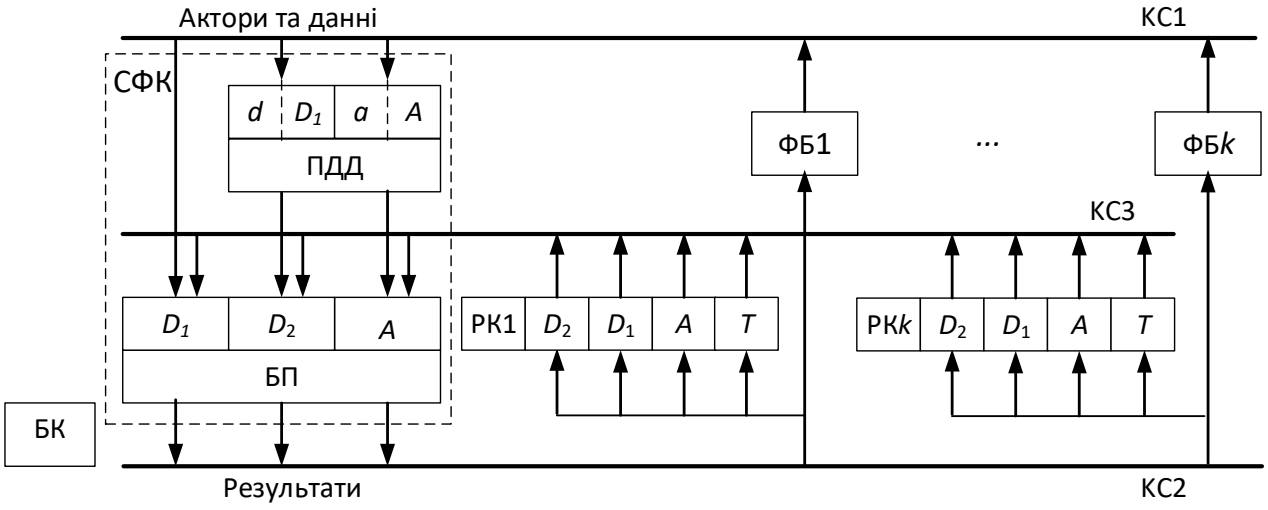


Рис. 5.39. Організація ПФМ (третій спосіб): БП – буферна пам’ять; D_1 і D_2 – поля даних; A – поле актора; T – поле таймера

Частину РК виконано у вигляді таймера, який запускається безпосередньо із записом команди в РК і ФБ (рис. 5.39). Тому немає потреби реалізовувати таймери у БМК.

5.4.2. Моделювання відмовостійкого потокового функціонального модуля на ПЛІС. У праці [135] виконано моделювання і досліджено часові характеристики обчислювального пристрою [192] під час відмов устаткування. Структуру пристрою [192] показано на рис. 5.39.

Для розроблення апаратної моделі потокового обчислювача використовувалась ПЛІС *EP*2*C*35*F*672*C*6*N* сімейства *Cyclone* II фірми *Altera* з такими основними характеристиками: кількість логічних елементів – 33216, кількість виводів мікросхеми – 484, кількість блоків убудованого ОЗП M4K (4 кбіт + 512 бітів парності) – 105. Для проектування, розроблення та налагоджування розроблюваних пристроїв на ПЛІС САПР *Qwartus* II. Проект синтезований з використанням мови опису апаратури *VHDL*.

Під час моделювання роботи потокового обчислювача на апаратурі ПЛІС експериментально визначено період часу, протягом якого всі керувальні сигнали встигають надійти до входів відповідних блоків обчислювальної системи в очікуванні наступного перепаду синхросигналу. Це час тривалості одного такту роботи системи і дорівнює 20 нс. Експериментально визначений час виконання набору основних операції (табл. 5.2), який використовується для контролю працездатності ОМ, таким чином. Отримані значення заносяться у таймер для кожного ОМ перед початком виконання команди. Якщо операція не виконується за визначений час (таймер спрацьовує), вважається, що ОМ відмовив.

Зважаючи на проблему обмеження апаратних ресурсів кристалів ПЛІС [34], зокрема ємності вбудованої пам’яті, внутрішніх каналів передавання даних і виводів мікросхеми, під час реалізації обчислювального середовища оцінено максимальну обчислювальну потужність розроблюваної системи, яка дозволить реалізувати використання одного кристала ПЛІС *EP*2*C*35*F*672*C*6*N*.

Отримано значення використаних ресурсів кристала ПЛІС *EP2C35F672C6N* для різної кількості ОМ (табл. 5.3). З огляду на фактичні ресурси мікросхеми *EP2C35F672C6N* визначено, що максимальна кількість ОМ для розміщення на кристалі ПЛІС *EP2C35F672C6N* дорівнює одинадцять з урахуванням також витрат ресурсів на створення середовища формування команд та пам’ять.

Таблиця 5.2

Час виконання основних операцій в апаратній моделі ПФМ на ПЛІС

Операція	Час виконання, нс	Операція	Час виконання, нс
Додавання	6	Порівняння $x \geq y$	6
Віднімання	6	Порівняння $x \leq y$	6
Множення	7	Вентиль *x if y*	6
Ділення	12	Вентиль *x if not y*	6
Зведення в квадрат	7	2-розмножувач	3
Вилучення кореня	8	*Y*-розмножувач	$Y \cdot 2 + 2$
Порівняння $x = y$	6	Повторювач *x*	6
Порівняння $x > y$	6	Повторювач *y*	6
Порівняння $x < b$	6		

Дослідження часових характеристик і ефективності функціонування ПФМ без відмов устаткування наведено у праці [135]. Досліджені коефіцієнт пришвидшення обчислень $K_{\text{П}}$ та коефіцієнт ефективності використання устаткування $K_{\text{Е}}$ (табл. 5.4) за такими формулами відповідно [51]:

$$K_{\text{П}} = \frac{T_1}{T_k},\ K_{\text{Е}} = \frac{K_{\text{П}}}{k}, \tag{5.4}$$

де T_1 – час розв'язування задачі з одним ФБ; T_k – час розв'язування задачі з k ФБ.

На апаратній моделі ПФМ розв’язано такі обчислювальні задачі.

Задача 1. Обчислення типової дрібнозернистої функції вигляду

$$F = \frac{\left(y^2 + by + 2ab + 4\right) \cdot \left(\left(a^2 + b^2 + c^2 + d^2\right)ab\right)}{a - b - ca + 12\sqrt{b^2 + a^2}}.$$

Таблиця 5.3

Оцінювання використаних ресурсів ПЛІС

Кількість ОМ	Логічні елементи	Ємність пам'яті, біт	Убудовані помножувачі
1	2871	176896	6
2	5468	176896	12
3	8015	176896	18
11	29224	176896	66

Таблиця 5.4

Результати моделювання ПФМ без відмов устаткування

Кількість ПФМ	Час розв'язування, мкс	Коефіцієнт пришвидшення	Коефіцієнт ефективності, %
1	9,45	1	1
2	7,91	1,1947	59
3	7,83	1,2069	40
4	7,83	1,2069	30
5	7,83	1,2069	24
6	7,83	1,2069	20
7	7,83	1,2069	17
8	7,83	1,2069	15
9	7,83	1,2069	13
10	7,83	1,2069	12
11	7,83	1,2069	10

ГА обчислювальної задачі 1, описаний модифікованою графічною мовою Деніса [51], зображено на рис. 5.40.

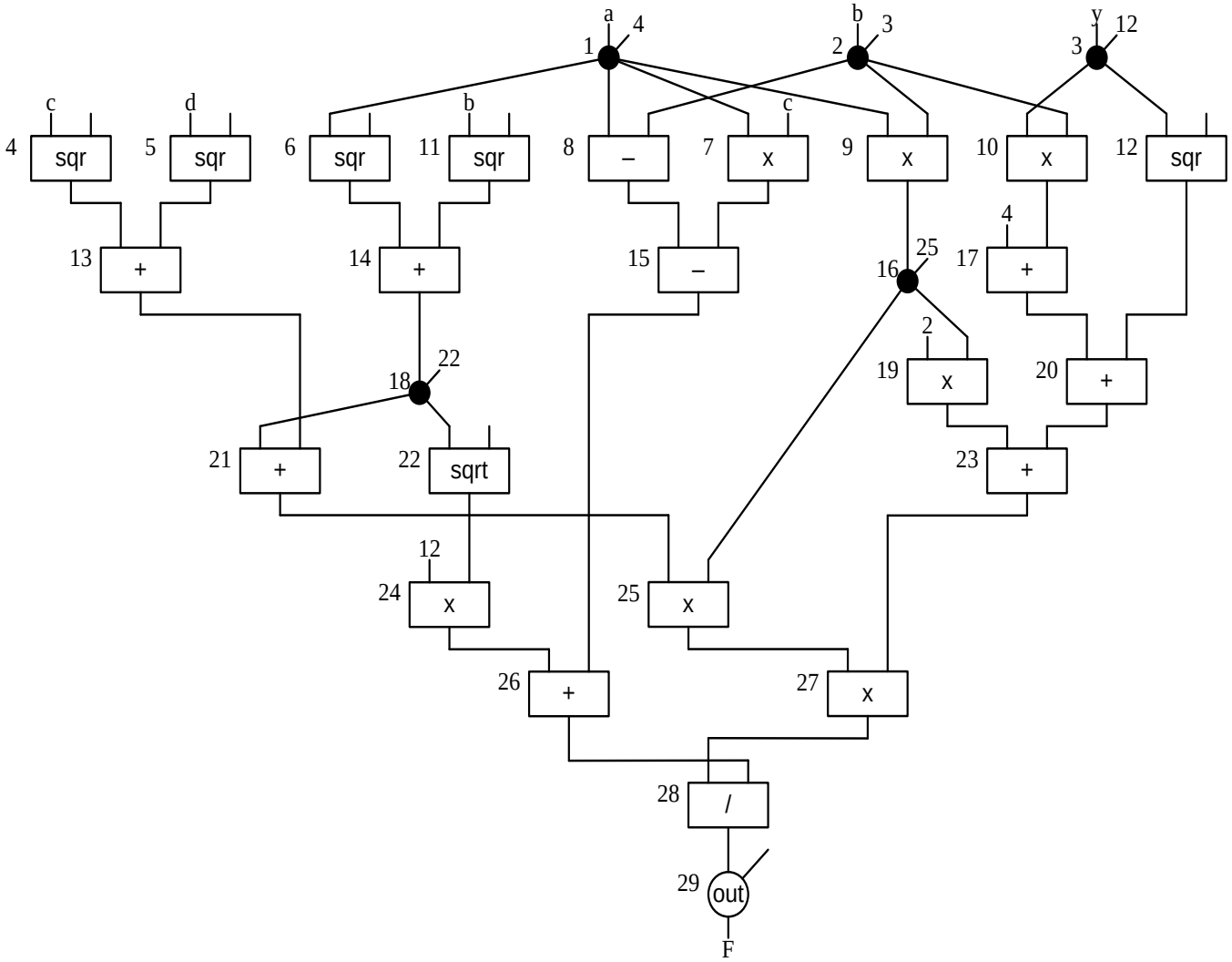


Рис. 5.40. Граф алгоритму задачі 1

Задача 2. Виконання циклічного алгоритму (рис. 5.41), що описується таким мнемонічнім алгоритмом:

X: = *a* + *b*;

F: = 1;

for *i* = 1 **to** *n* **do** *F*:=(*F* + *i*) * (*X* – 2).

З результатів моделювання (табл. 5.4) нормальної роботи ПФМ видно, що для розв’язання типової задачі 1 ефективним є використання не більше ніж три ФБ. Аналогічний результат отримано під час моделювання циклічної задачі 2 (рис. 5.41) [135].

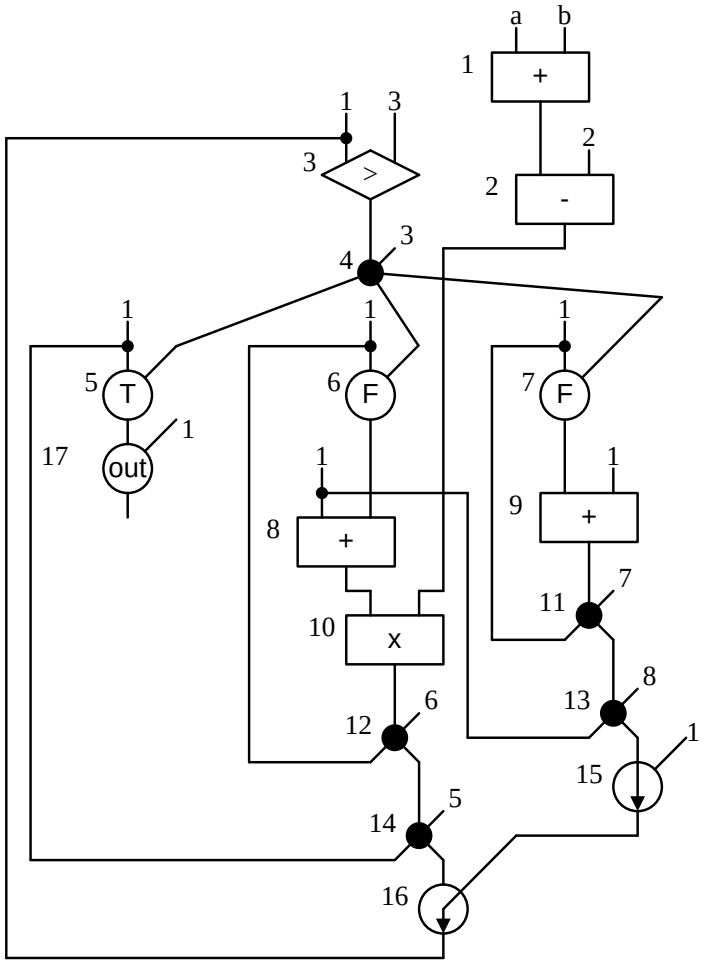


Рис. 5.41. Граф алгоритму задачі 2

Доведемо експериментально теоретичне припущення, що збільшення кількості ФБ не приводить до зменшення часу обчислень, але сприяє збільшенню надійності відмовостійкого ПФМ [135].

Дослідження часових характеристик і ефективності функціонування ПФМ під час відмов устаткування наведено у праці [135]. Виконано моделювання типових обчислювальних задач різної складності: задачі 1 (див. рис. 5.40) і задачі 2, що містить цикли (рис. 5.41).

Змодельовані засоби автоматичної реконфігурації дозволяють продовжувати обчислення, поки є хоча б один працездатний ФБ. Алгоритми керування формуванням і розподілом команди між ФБ обчислювача описано у працях [192, 193]. Алгоритм керування формуванням і розподілом команд у потокових обчислювачах з паралельним СФК подано на рис. 5.35. Загалом у процесі обчислень готові до виконання команди ФБ передають сигнали готовності у БМК. Чергова команда, що сформована в БФК, передається для виконання в один з готових ФБ. Під час виконання обчислень збій в роботі одного або декількох ФБ може виникнути до початку виконання команди: не завершено процес пересилання команди у ФБ (випадок 1), або після того, як ФБ почав виконувати команду (випадок 2).

Виконаємо моделювання запропонованих у працях [192, 193] способів удосконалення автоматичної реконфігурації ПФМ у разі відмов устаткування, що ґрунтується на застосуванні РК у складі кожного ФБ.

Перший спосіб: Регістр команд без таймера команди. У цьому випадку контроль стану системи здійснюється не рівні централізованих засобів керування ПФМ, таймер команди міститься у структурі БМК (рис. 5.38);

Другий спосіб: Регістр команд з таймером команди відповідно структурі ПФМ на рис. 5.39.

Моделювання відновлення ПФМ у разі відмов устаткування виконано за першим і другим способами. Відповідно до цього, якщо ФБ відмовив після початку виконання команди, розглянуто випадок 2.I і випадок 2.II.

Час виконання обчислень з урахуванням часу реконфігурації системи під час відмови ФБ у випадку 1 наведено в табл. 5.5 [135]. Час виконання обчислень з урахуванням часу реконфігурації системи під час відмови ФБ у випадку 2 наведено в табл. 5.6.

На підставі отриманих у праці [135] результатів моделювання роботи апаратної моделі ПФМ на ПЛІС побудовано графіки залежності середнього затримання часу виконання обчислень під час відмови устаткування від кількості працездатних ФБ, показаних на рис. 5.42.

У випадку, якщо команда ще не надійшла до виконання в певний ФБ (крива випадку 1 на рис. 5.42), реконфігурація ПФМ під час відмов устаткування за надмірної кількості обчислювальних ресурсів супроводжується незначними затриманнями.

Таблиця 5.5

Час виконання обчислень зі збоєм роботи ПФМ під час розподілу команд в ФБ (випадок 1)

Загальна кількість ФБ	Час виконання обчислень, мкс	
11	Задача 1	Задача 2
Кількість ФБ, що відмовили		
1	7,83	12,15
2	7,83	12,15
3	7,83	12,15
4	7,83	12,15
5	7,83	12,19
6	7,83	12,33
7	7,83	12,33
8	7,83	12,33
9	7,85	12,45
10	10,21	14,25

Збільшення кількості невиконаних команд в черзі на повторне виконання збільшує час відновлення ПФМ за відмов устаткування. У цьому

випадку в процесі реконфігурації системи перевіряються стани автомата БМК, видаються відповідні керувальні сигнали на виходи БМК, записуються команди із загальної шини повернення команди у БФК для повторного виконання в працездатному ФБ, що зумовлює мінімальний час, який дорівнює часу передавання одного слова даних по шині повернення команди і часу звернення до БПК на локальному рівні БФК. За відмови п’яти і більше ОМ затримання загалом залежить від кількості повернених команд.

Таблиця 5.6

Час виконання обчислень зі збоєм роботи обчислювача в процесі виконання команди (випадок 2)

Загальна кількість ФБ	Час виконання обчислень, мкс	
11	Задача 1	Задача 2
Кількість ФБ, що відмовили		
1	7,95	12,27
2	7,91	12,23
3	7,95	12,49
4	8,07	12,49
5	8,03	12,59
6	8,05	12,57
7	8,11	12,67
8	8,15	12,59
9	8,27	13,01
10	10,37	14,83

Якщо команда вже надійшла на виконання в певний ФБ (крива випадку 2.I на рис. 5.42), відмова устаткування визначається станом таймера команди, проміжок часу спрацювання якого спричинює додаткові затримання

часу реконфігурації ПФМ. Це пояснюється тим, що факт відмови ФБ за станом таймера визначається на рівні централізованого БМК. При цьому час реакції керувального автомата в БМК залежить від його стану в даний момент часу, тому на графіку залежності (рис. 5.42) затримання збільшується досить нерівномірно.

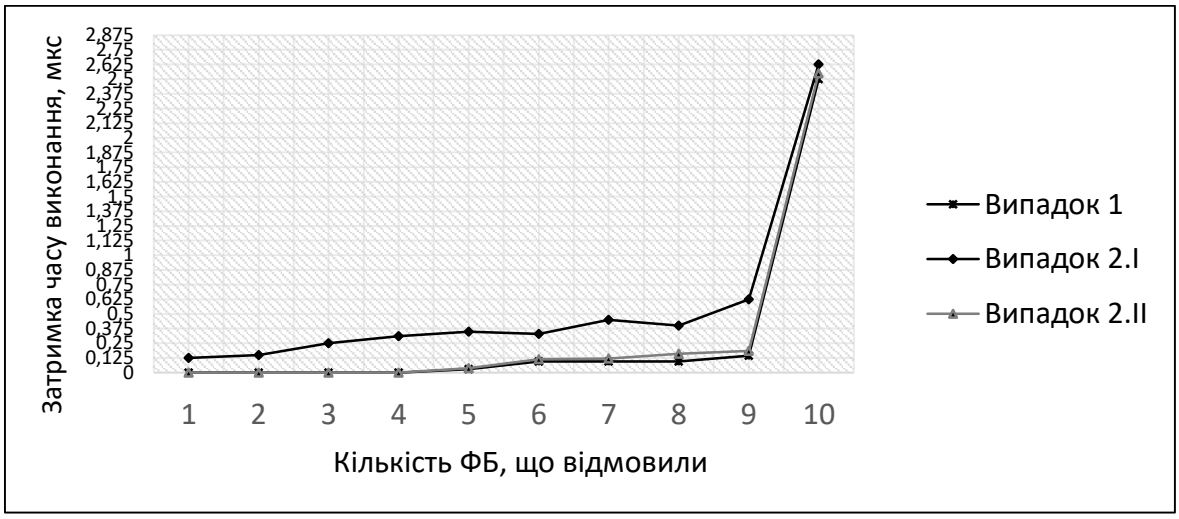


Рис. 5.42. Середнє затримання часу розв’язання задач від кількості ФБ, що відмовили під час обчислень

Реалізація таймера команди в слові команди (крива випадку 2.II на рис.5.42) дозволяє покласти процес реконфігурації у разі відмов устаткування на апаратні засоби інтерфейсу БФК без участі централізованих засобів керування ПФМ. Це значно зменшує час реконфігурації ПФМ за відмов устаткування і підвищує ефективність вдосконалених засобів відмовостійкості керувальних ядер на базі ПСФК.

Зменшення кількості працездатних ФБ порівняно зі зменшенням коефіцієнта ефективності розв’язання задач призводить до значного збільшення затримань через уповільнення обчислювального процесу. Один працездатний ФБ хоча й підтримує мінімальну працездатність системи, що за наявності надмірності устаткування підвищує надійність системи, але реалізує послідовний обчислювальний процес, що характеризується різким збільшенням часу обчислення.

ВИСНОВКИ ДО РОЗДІЛУ 5

1. Реалізація моделі обчислень, керованих потоком даних, сприяє уникненню проблем, пов’язаних із застосуванням традиційних технологій планування обчислень і розподілу обчислювальних ресурсів в РКС, що мають функціональні та апаратні обмеження.

2. Удосконалено метод автоматичного розподілу завдань, що дає змогу автоматизувати функцію керуванням обробкою інформації на апаратному рівні РКС, керованих потоком даних. Розроблені засоби автоматичного розподілу завдань на реконфігуровне обчислювальне середовище на ПЛІС за рахунок суміщення обчислювального процесу і процесу завчасної реконфігурації обчислювального середовища дозволяють скоротити непродуктивні витрати часу та продуктивності у процесі відображення потоку задач на реконфігуровне обчислювальне середовище.

3. Модифіковано спосіб автоматичної синхронізації процесів у РКС, керованих потоком даних, шляхом реалізації дворівневої синхронізації процесів за каналами зв’язку. Це зменшує час звернення до загальних ресурсів, непродуктивні завантаження загального комунікаційного середовища і звернення до загальної пам’яті, обсяг використання вбудованої пам’яті ПЛІС.

4. У цілому вдосконалені засоби розподілу команд та апаратної синхронізації зменшують час звернення до загальних ресурсів, непродуктивне завантаження загального комунікаційного середовища і непродуктивні звернення до пам’яті, накладні витрати на паралельну обробку даних, ємність використовуваної вбудованої пам’яті ПЛІС. Це підвищує ефективність взаємодії між функціональними елементами ПАРС та ОМ РКС, а також розширює функціональні можливості ПАРС з урахуванням передумов для масштабування РКС.

5. Розроблені засоби апаратного автоматичного розподілу завдань та синхронізації процесів для РКС забезпечують автоматичне керування паралельною обробкою даних і розподілом завдань на реконфігуровне обчислювальне середовище прозоро для програмного шару обчислювальної

системи, що звільняє операційну систему для розв’язання неспецифічних задач і дозволяє ефективно використовувати на рівні операційної системи традиційні технології паралельного програмування.

6. Моделювання методу адаптивного відображення завдань на базі розробленої апаратної імітаційної моделі ОМ РКС підтвердило теоретичні та формальні обґрунтування ефективності адаптивного відображення завдань на реконфігуровне обчислювальне середовище, що наведені в другому розділі дисертаційної роботи, і визначило таке:

– порівняно з технологією повторного використання обчислювальних ресурсів ФБ завчасна реконфігурація забезпечує у цілому в два рази пришвидшення обробки інформації в РКС і не залежать від частоти повторень однотипних функцій в алгоритмі;

– комплексне застосування механізмів повторного використання ресурсів ФБ і завчасної реконфігурації дозволяє видалити майже всі непродуктивні витрати під час процесу реконфігурації незалежно від частоти виконання однотипних функцій протягом обчислювального процесу;

– уповільнення приросту ефективності спричинюється неефективним використанням обчислювальних ресурсів або переваженням кількості операцій обміну даними над обсягом обчислень, тобто, якщо зернистість обчислювального середовища не відповідає зернистості розв’язуваної задачі;

– інтенсивність підвищення ефективності паралельної обробки інформації з використанням апаратних засобів автоматичного розподілу завдань та синхронізації в середньому на 2% вища, а в області відповідності зернистості обчислювального середовища зернистості алгоритму розв’язуваної задачі на 5% вища, ніж із застосуванням традиційних технологій паралельних обчислень;

– застосування апаратних засобів автоматичного розподілу завдань і синхронізації процесів в області оптимального співвідношення зернистості обчислювального середовища і зернистості алгоритму обчислювальної задачі

на 65% пришвидшує обчислювальний процес порівняно із застосуванням традиційних технологій паралельних обчислень;

– застосування апаратних засобів автоматичного розподілу завдань і синхронізації забезпечує збільшення ефективності обчислень у середньому на 10,25% порівняно з традиційними технологіями паралельних обчислень.

7. Запропонований новий спосіб побудови паралельного середовища формування команд дозволяє пришвидшити процес формування команд у керувальному ядрі ОМ РКС, керованої потоком даних. Це в цілому дозволяє підвищити ефективність потокових обчислювальних систем під час розв’язання задач великої і змінюваної розмірності, що виконуються в режимі обмеженого часу виконання.

8. Для реалізації модифікованого методу розподілу команд у ПСФК розроблено структуру однотипних інтерфейсів блоків формування команд, протоколів обміну даними та алгоритмів керування формуванням команд і розподілу завдань між ФБ. Розроблені засоби дозволяють реалізовувати масштабовану структуру середовища формування команд та забезпечувати передумови для статичної реконфігурації ПСФК залежно від розмірності вирішуваних обчислювальних алгоритмів, що розширює його функціональні можливості.

9. Розроблені в дисертаційній роботі засоби синхронізації роботи блоків формування команд на базі однотипних інтерфейсів забезпечують синхронізацію паралельних процесів формування команд і доступу до загальних системних ресурсів на рівні апаратних засобів СФК без участі централізованих і децентралізованих засобів керування обчислювальним процесом, що пришвидшує процес формування команд і відмовостійкість ПФМ.

10. На підставі імітаційного моделювання способу побудови паралельного середовища формування команд визначено таке:

– зі збільшенням кількості БФК час виконання алгоритму змінюється за логарифмічним законом і залежить від співвідношення кількості БФК, кількості ФБ і ступеня паралелізму розв’язуваної обчислювальної задачі;

– ефективність паралельного середовища формування команд підвищується пропорційно збільшенню кількості БФК і ширини графу обчислювального алгоритму в межах оптимального співвідношення ступеня розпаралелювання обчислювальної задачі і структури реконфігуровної обчислювальної системи з урахуванням апаратних обмежень елементної бази – кристалів ПЛІС;

– застосування ПФМ з ПСФК у вузлах РКС дозволяє підвищити ефективність розв’язання дрібнозернистих обчислювальних задач великої і несталої розмірності, що розв’язуються в режимі реального часу, за рахунок підвищення швидкодії обчислень більш ніж на 60% порівняно із застосуванням відомих потокових обчислювачів.

11. Удосконалений спосіб реконфігурації ПФМ у випадку відмов устаткування шляхом удосконалення засобів розподілу команд у паралельному середовищі формування команд і нового способу синхронізації роботи паралельних блоків формування команд забезпечує зменшення витрат часу на реконфігурацію системи як під час адаптації її структури до вимог розв’язуваних задач, так і під час відмови устаткування, а також розширює функціональні можливості потокових обчислювачів.

12. Автоматичне відновлення системи за відмови устаткування відбувається на апаратному рівні СФК без додаткових команд і участі централізованих засобів керування. Це потребує мінімального часу, який вимірюється часом пересилання одного слова даних на фізичному рівні кристала ПЛІС. У цілому це підвищує швидкодію ПОМ, достовірність результатів обчислень і забезпечує умови для підвищення надійності керувального ядра ПФМ, що важливо для систем керування в реальному часі.

13. Запропоновані апаратні засоби вдосконалення структури керувального ядра ОМ РКС та підвищення його відмовостійкості можуть бути

використані в РКС із загальним розділеним між відмовостійкими ядрами реконфігуровним обчислювальним середовищем, у тому числі гетерогенним реконфігуровним обчислювальним середовищем на ПЛІС. Це створює передумови для розширення функціональних можливостей РКС для розв’язання широких класів задач керування, зокрема, задач керування великої розмірності та несталої розмірності в режимі реального часу.

РОЗДІЛ 6
МОДЕЛЮВАННЯ МЕТОДІВ ТА ЗАСОБІВ ОРГАНІЗАЦІЇ ОБРОБКИ ІНФОРМАЦІЇ В РЕКОНФІГУРОВНИХ КОМП'ЮТЕРНИХ СИСТЕМАХ НА ПЛІС

6.1. Розроблення імітаційної моделі реконфігуровної комп'ютерної системи

6.1.1. Логічна структура імітаційної моделі обчислювального модуля реконфігуровної комп'ютерної системи. У праці [157] описано логічну структуру імітаційної моделі РКС (рис. 6.1), яку подано центральним процесором, централізованою бібліотекою конфігураційних даних та обчислювальним модулем РКС, які поєднані між собою мережевим комунікаційним середовищем.

Імітаційна модель ОМ РКС (рис. 6.1) складається з модулів локальної пам'яті конфігураційних даних, контролера реконфігурації, багаторівневої кеш-пам'яті (кеш-пам'яті першого рівня, кеш-пам'яті другого рівня), реконфігуровного обчислювального середовища. Спеціалізований контролер реконфігурації, модулі локальної пам'яті конфігураційних даних і кеш-пам'яті складають статичну частину ОМ РКС. Реконфігуровна ділянка є динамічною частиною ОМ РКС, яка має певний розмір і структуру відповідно до виразів (2.10) і (3.24). Реалізовано гнучку 2*D* модель розміщення апаратних задач (див. рис. 1.3, *а*). Умовно вважаємо, що ФБ апаратних задач розміщуються на поверхні ПЛІС відповідно до відомих способів [17, 44, 68] у вільному місці динамічної ділянки кристала ПЛІС. Структура та принцип функціонування багаторівневої кеш-пам'яті детально описано у праці [158].

Узагальнений логічний алгоритм відображення задач на реконфігуровне обчислювальне середовище зображено на рис. 6.2.

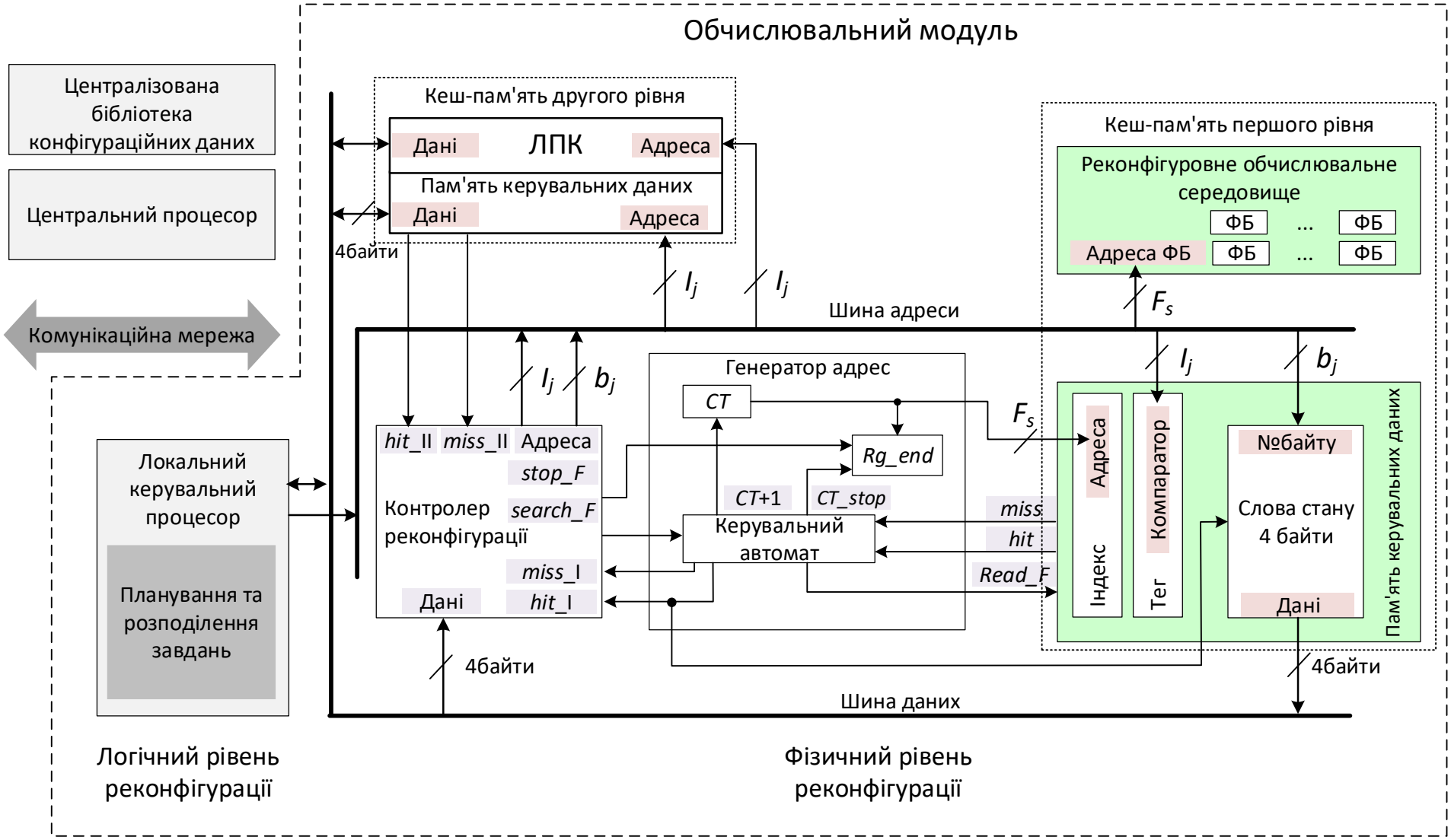


Рис. 6.1. Логічна структура обчислювального модуля РКС

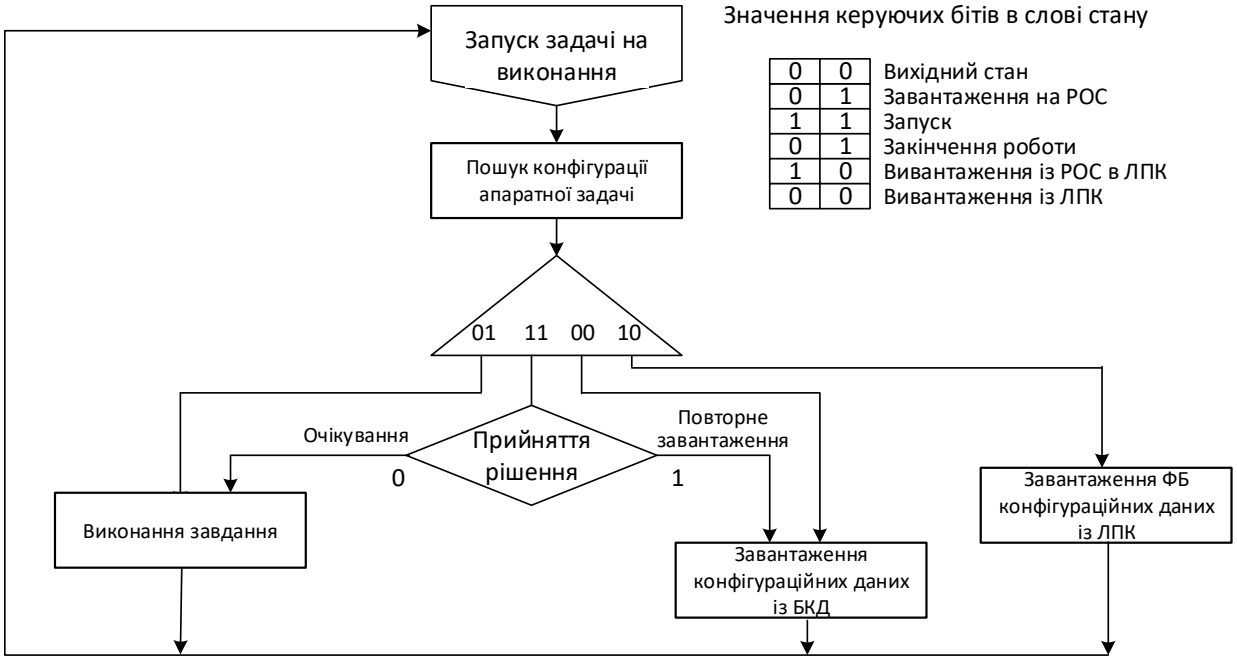


Рис. 6.2. Логічний алгоритм реалізації методу адаптивного відображення задач на реконфігуровне обчислювальне середовище (РОС)

Процес відображення ініціюється готовністю до виконання чергового завдання N_i у ГА обчислювальної задачі G_M і складається з таких етапів:

Етап 1. Виконується процедура пошуку конфігураційних даних для виконання відповідного завдання.

Етап 2. Залежно від місця розташування конфігураційних даних виконуються розгалуження алгоритму і процедура завантаження конфігураційних даних.

Етап 3. Виконується процедура налаштування обчислювального середовища на ПЛІС і запуску апаратної задачі на виконання.

На другому етапі виконання логічного алгоритму виконується розгалуження за такими умовами:

– завантаження конфігураційних даних із БКД та запуск апаратної задачі на виконання здійснюються згідно зі стратегією завантаження I за виразом (2.17);

– конфігураційні дані з локальної пам’яті конфігураційних даних завантажуються шляхом звернення до кеш-пам’яті другого рівня згідно з послідовністю завантаження II за виразом (2.18);

– зверненням до кеш-пам’яті першого рівня виявляється, що апаратна задача вже розміщена на поверхні реконфігуровного обчислювального середовища, що відповідає стратегії завантаження III за виразом (2.19).

Якщо згідно зі стратегією завантаження III ФБ шуканої апаратної задачі знайдено на поверхні реконфігуровного обчислювального середовища, але він використовується для виконання іншого завдання (активний ФБ), приймається рішення щодо подальшої стратегії виконання обчислення: очікування завершення розв’язання активної апаратної задачі або створення дубліката ФБ на поверхні реконфігуровного обчислювального середовища.

Створення дубліката доцільне за таких умов:

– час розв’язання поточної апаратної задачі більший за час завантаження дубліката;

– відповідне завдання має високий показник частоти використання, для визначення якого пускається кількість «бонусів» більша ніж 50% від максимальної кількості «бонусів» [158].

Стратегії обслуговування завдань, що реалізовано в межах вирішення проблеми оптимізації процесу обробки інформації, ураховують такі умови для прийняття рішень щодо створення дублікатів ФБ:

– виконуються умови просторових обмежень реконфігуровного обчислювального середовища відповідно до виразу (3.24);

– стратегія копіювання конфігураційних даних через внутрішню пам’ять ПЛІС або повторне завантаження із ЛПК визначається відповідно до інтегрованого критерію оптимізації, що запропоновано в підрозділі 3.2.1 дисертаційної роботи.

Якщо приймається рішення про недоцільність створення дубліката ФБ апаратної задачі, процес відображення переходить у стан очікування завершення розв’язання поточної апаратної задачі в даному ФБ.

У межах процесу відображення розв’язується задача керування розкладом розміщення та підтримування конфігураційних даних апаратних задач.

Як критерій підтримування ФБ апаратної задачі на поверхні реконфігуровного обчислювального середовища використовується показник частоти використання ФБ, який вимірюється кількістю наданих «бонусів» підтримання. Механізм надання «бонусів» підтримання описано у праці [158], згідно з яким апаратні задачі, що мають більшу частоту повторень, витісняють неактуальні задачі з поверхні реконфігуровного обчислювального середовища. Апаратні задачі, що використали запас «бонусів», видаляються з поверхні реконфігуровного обчислювального середовища, але отримують додатковий час підтримання на рівні обчислювального модуля у швидкодійній ЛПК – кеш-пам’яті ІІ рівня (див. рис. 6.1). Фактори, що зумовлюють додатковий час для зберігання в кеш-пам’яті другого рівня обґрунтовано у праці [158] і узагальнено, що зберігання конфігураційних даних у ЛПК динамічно змінюється протягом обчислювального процесу і залежить від параметрів алгоритму обчислювальної задачі, а саме від частоти повторень однотипних функцій і параметрів реконфігуровного обчислювального середовища. Конфігураційні дані видаляються із ЛПК за відомими технологіями організації кеш-пам’яті, коли із пам’яті видаляються найбільш застарілі записи. Після видалення конфігураційних даних з локального рівня обчислювального модуля за необхідності виконується їх повторне завантаження із центральної БКД.

Пошук конфігураційних даних у багаторівневій кеш-пам’яті ініціалізується контролером реконфігурації, який формує виконавчу адресу I_jF_j на загальну шину адреси обчислювального модуля згідно з протоколом пошуку конфігураційних даних, що описано у праці [158].

Після закінчення процедури пошуку конфігураційних даних контролер реконфігурації аналізує керувальні сигнали вдалого або невдалого пошуку: *hit*_I і *miss*_I або *miss*_II відповідно. На підставі отриманих сигналів контролер реконфігурації визначає подальшу стратегію завантаження конфігураційних даних і виконання завдання.

Логічну схему блока формування керувальних сигналів результату пошуку, яку реалізовано в інтерфейсі контролера реконфігурації, зображено на рис. 6.3.

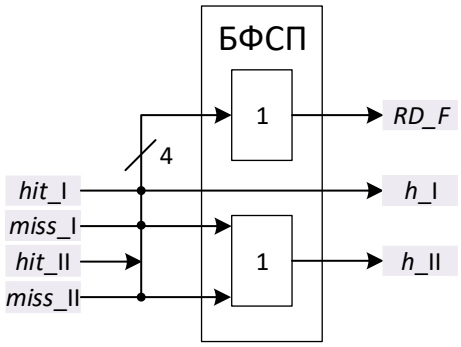


Рис. 6.3. Логічна схема формування сигналів стану пошуку: БФСП – блок формування сигналів стану пошуку

Згідно із функціональною схемою на рис. 6.3 результат пошуку визначається за такими керувальними сигналами:

– *RD_F* – керувальний сигнал готовності результату пошуку встановлюється на підставі сигналів *hit* або *miss* від кеш-пам’яті;

– *h*_I – вдалий пошук на поверхні реконфігуровного обчислювального середовища (у КЕШ I);

– *h*_I – вдалий пошук в ЛПК (КЕШ II).

Значення керувальних сигналів стану пошуку наведено в табл. 6.1.

Алгоритм пошуку та завантаження конфігураційних даних апаратної задачі показано на рис. 6.4. Процедура пошуку конфігураційних даних на різних рівнях кеш-пам’яті здійснюється засобами керувального автомата без участі локального керувального процесора і контролера реконфігурації таким чином.

Таблиця 6.1

Визначення значень сигналів результату пошуку конфігураційних даних

КЕШ I		→	**КЕШ II**		**Результат процедури пошуку**	
					***h*_I *h*_II**	**Запуск процесу**
1	*hit*_I	0			1 0	Розв'язання апаратної задачі на поверхні реконфігуровного обчислювального середовища
0	*miss*_I	1	0	*miss*_II	0 1	Завантаження конфігураційних даних із ЛПК
			1	*hit*_II	0 0	Завантаження конфігураційних даних із БКД – невдалий пошук у локальному адресному просторі

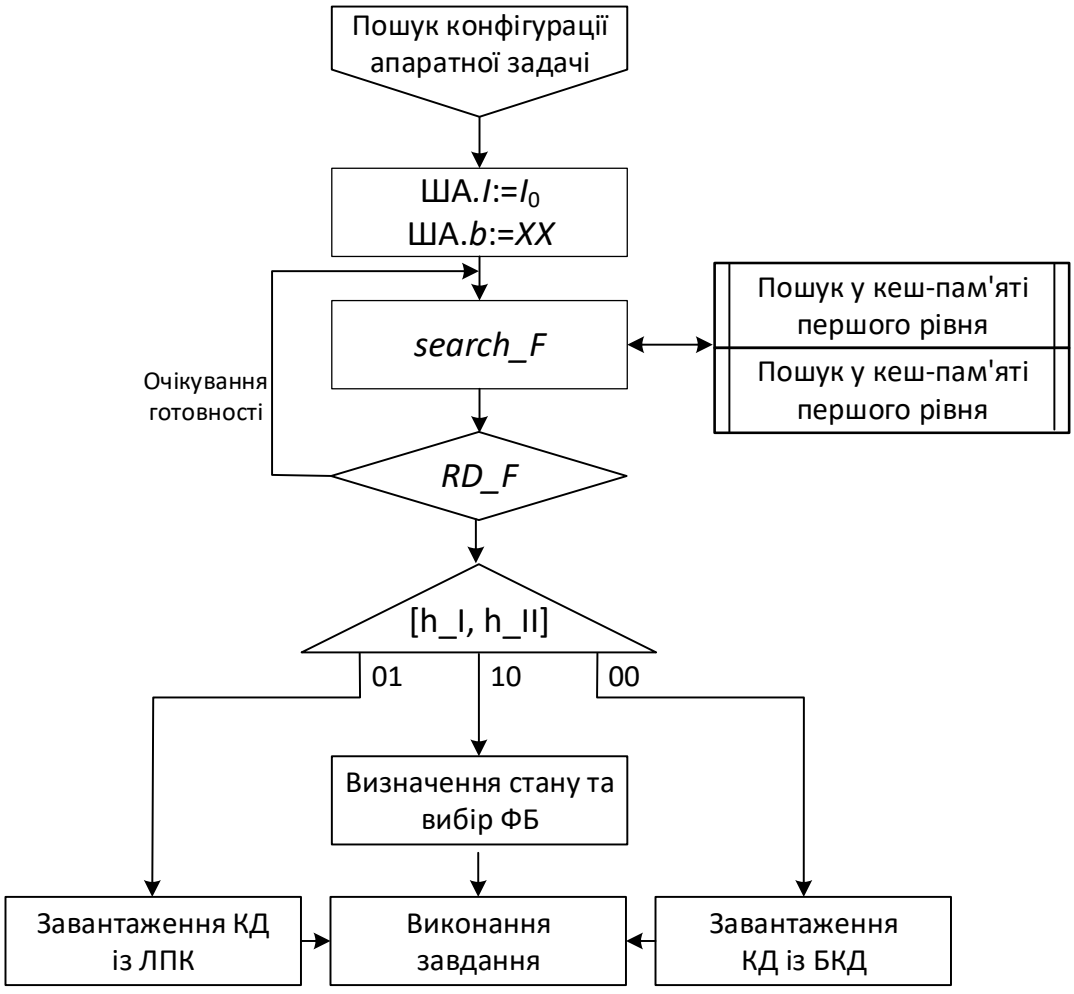


Рис. 6.4. Алгоритм пошуку конфігураційних даних апаратної задачі

Якщо ФБ на поверхні реконфігуровного обчислювального середовища знайдено (*hit_I*), пошук припиняється і керування передається контролеру реконфігурації для збереження й аналізу відповідних бітів слова стану ФБ. На підставі цього контролер реконфігурації або припиняє процедуру пошуку: визначає подальшу стратегію виконання завдань, або продовжує переглядати вміст блоків. Процедура закінчується, якщо жодний ФБ не знайдений на поверхні реконфігуровного обчислювального середовища (*miss_I*).

Після закінчення чергової процедури пошуку в спеціальному регістрі *Rg_end* генератора адрес зберігається останній стан лічильника адрес. Наступна процедура пошуку починається з рядка кеш-пам’яті з адресою (*Rg_end* + 1), з досяганням лічильником значення *Rg_end* пошук знову буде припинено. Такий алгоритм циклічної роботи лічильника адрес забезпечує динамічні пріоритети ФБ на поверхні реконфігуровного обчислювального середовища, що зменшує час пошуку і непродуктивні зберігання ФБ.

Контролер реконфігурації може примусово перервати процедуру пошуку сигналом *stop_F*, при цьому останній стан лічильника завантажиться в регістр *Rg_end*.

Алгоритм функціонування керувального автомата та граф керувального автомата показано на рис. 6.5 і 6.6 відповідно.

Імітаційна модель ОМ РКС емулює функціонування основних функціональних складових РКС, які реалізовано у вигляді таких програмних модулів.

HardwareSystem – головний модуль емулює функціональний процес, що здійснюється контролером реконфігурації і включає підлеглі модулі:

– *FPGA* – реконфігуровна область;

– *memory* – локальна пам’ять, що включає функціонал кеш-пам’яті першого рівня;

– *bonuses* – модуль підтримання «бонусів».

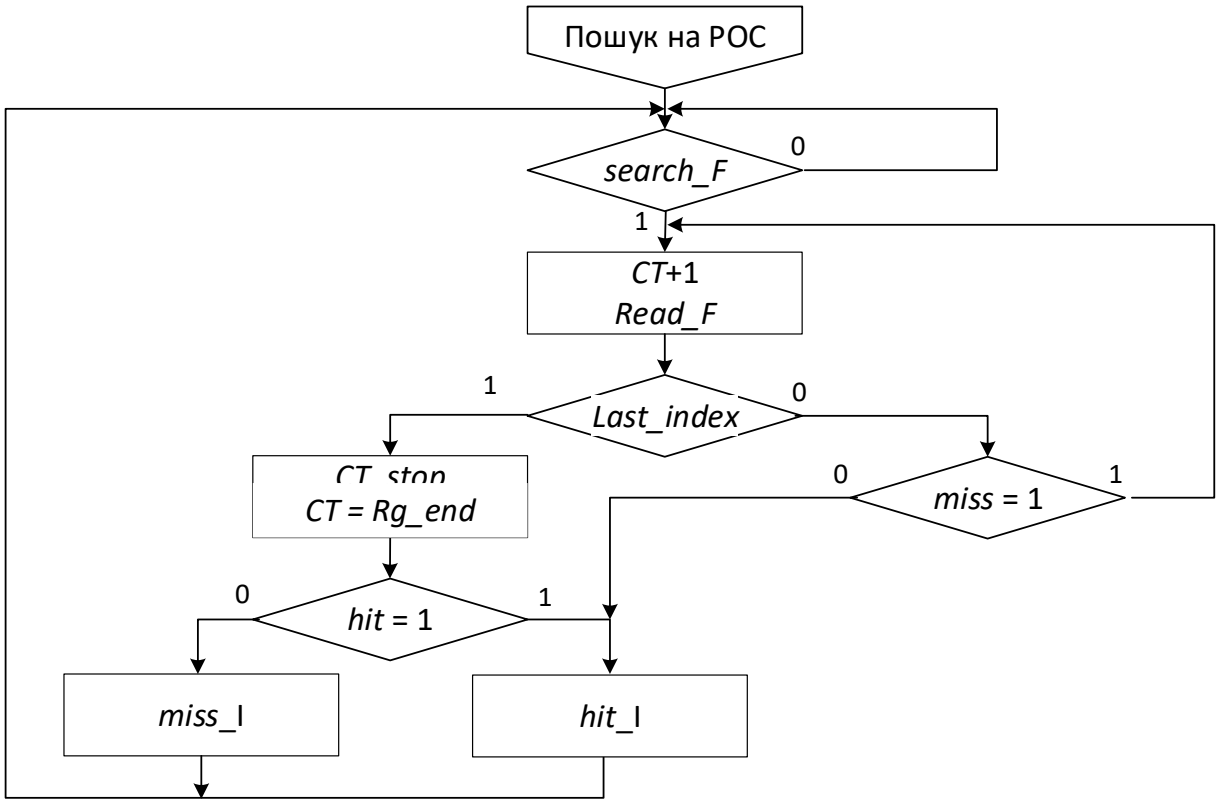


Рис. 6.5. Алгоритм керування пошуком конфігураційних даних

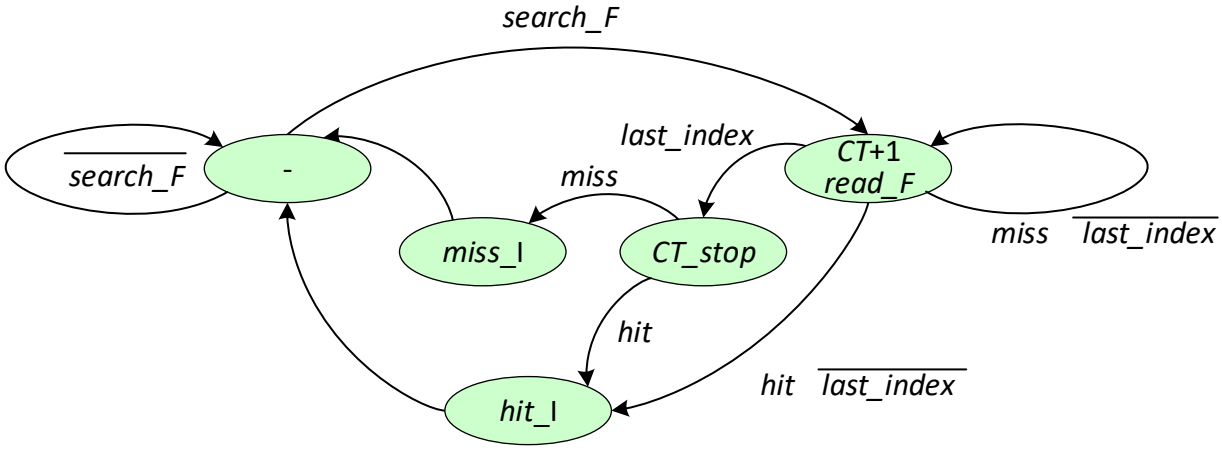


Рис. 6.6. Граф керувального автомата

Модуль *HardwareSystem* виконує такі функції:

– *findConfiguration* – пошук конфігурації апаратної задачі,

– *load* – налаштування конфігурації апаратної задачі на ПЛІС.

Функція *findConfiguration* повертає одне з трьох результатів: *TSK_FPGA*, *TSK_MEM*, *TSK_LIB*, значення яких установлюються на підставі сигналів *hit*

або *miss* (вдалого або невдалого пошуку) від кеш-пам’яті першого або другого рівнів відповідно до табл. 6.2 і табл. 6.1.

Таблиця 6.2

Значення функції *findConfiguration*

КЕШ I		→	**КЕШ II**		**Результат процедури пошуку**	
					***h*_I *h*_II**	**Запуск процесу**
1	*hit*_I	0			1 0	*TSK_FPGA*
0	*miss*_I	1	0	*miss*_II	0 1	*TSK_MEM*
			1	*hit*_II	0 0	*TSK_LIB*

Реконфігуровне обчислювальне середовище подано масивом об’єктів, що відповідають завантаженим у даний момент ФБ апаратних задач. Система обліку та керування реконфігуровними ресурсами забезпечує нарахування «бонусів» та підтримання станів лічильників і таймерів для контролю роботи ФБ апаратних задач на поверхні реконфігуровного обчислювального середовища. ФБ є набором властивостей, що зберігаються в керувальній пам’яті КЕШ I. Локальну пам’ять описано множиною об’єктів, що відповідають апаратним задачам, витісненим із реконфігуровної області. Апаратна задача в локальній пам’яті є набором властивостей, що зберігаються в керувальній пам’яті КЕШ II.

Модуль *lib* центральної БКД являє мовою масиви даних (*data*), що зберігаються у файлі з даними (*LIBRARY_FILE*). Модуль містить набір функцій, що забезпечують доступ до даних. Апаратні задачі, які синтезовані заздалегідь, зберігається в централізованій БКД у вигляді об’єктів конфігураційних файлів разом зі своїми властивостями.

6.1.2. Вихідні дані для моделювання базових стратегій відображення задач на реконфігуровне обчислювальне середовище. На розробленій моделі ОМ РКС проведено імітаційне моделювання функціональних процесів, що

відбуваються у РКС, на підставі математичних моделей, розроблених у дисертаційній роботі. Виконано моделювання запропонованих процесу адаптивного відображення завдань на реконфігуровне обчислювальне середовище (2.24) – (2.27), (2.36) і (3.1), процесу багаторівневого кешування конфігураційних даних, різних стратегій відображення завдань на реконфігуровному обчислювальному середовищі, досліджено ефективність оптимізації процесу обробки інформації на різних рівнях РКС за інтегральним критерієм оптимізації, що запропоновано в підрозділі 3.2.1 дисертаційної роботи.

Програмний модуль (*modeller*) моделює процес відображення завдань на реконфігуровне обчислювальне середовище. Функція моделювання *modell* повертає часові результати моделювання у вигляді діаграми Ганта в текстовому форматі. Програмний модуль відтворює обчислювальні процеси у РКС без пришвидшення і з пришвидшенням реконфігурації обчислювального середовища, а також процеси адаптивного відображення завдань на реконфігуровне обчислювальне середовище згідно з розробленими в дисертаційній роботі математичними моделями (2.24) – (2.27), (2.36), (3.1).

Для цього реалізовано і досліджено різні стратегії завантаження конфігураційних даних згідно з виразами (2.17) – (2.19). Відповідно до них розраховується час обчислення кожного завдання $T_{SUM\,j}$:

$$T^{\mathrm{I}}_{SUM\,j} = (T_{COMM\_NET\ j}) + T_j,\ R_j^{\ \mathrm{I}} = T_{COMM\_NET\,j};$$

$$T^{\mathrm{II}}_{SUM\ j} = T_{COMM\ j} + T_j,\ R_j^{\mathrm{II}} = T_{COMM\ j};$$

$$T^{\mathrm{III}}_{SUM\ j} = T_j,\ R_j^{\mathrm{III}} = 0,$$

де T_j – час розв'язання апаратної задачі I_j, включаючи процеси введення вихідних даних та виведення результатів $T_j = T_{IO\,j} + T_{HW}$ (функція *LOAD_DATUM_TIME*); $T_{COMM\,j}$, $T_{COMM\_NET\,j}$ – час передавання конфігураційних даних (функція *LOAD_LAST_TIME*); $T_{HW\,j}$ – час розв'язання апаратної задачі на апаратурі ПЛІС; *workTime* – очікуваний час виконання –

константа, визначена на підставі функціонального модулювання роботи ФБ у процесі синтезу; *correctTime* – відкоригований час розв'язання апаратної задачі в процесі обчислень;

$T_{COMM\ j}$ – час передавання конфігураційних даних з локальної пам'яті відповідно до способу (2.14) і виразу для визначення часу звернення до модуля пам'яті [51] залежить від часу звернення до модуля пам'яті (константа *MEMORY_ACCESS_TIME*), розміру конфігураційного файлу (*bytestreamWords*) і часу ініціалізації інтерфейсу ПЛІС;

$T_{COMM\_NET\ j}$ – час передавання конфігураційних даних із БКД на рівень обчислювального модуля мережевими засобами зв'язку залежить від часу звернення до модуля пам'яті (константа *MEMORY_ACCESS_TIME*), коефіцієнта затримання мережевого передавання конфігураційних даних, що генерується випадковим чином [158] (*NETWORK_MAX_RANDOM_TIME*), розміру конфігураційного файлу (*bytestreamWords*) і часу ініціалізації інтерфейсу ПЛІС.

Час пошуку конфігураційних даних апаратної задачі для всіх послідовностей завантаження визначається функцією (*TIME_findConfiguration*) з відповідними вихідними параметрами.

Досліджено серію алгоритмів обчислювальних задач, поданих макрографами потоків даних, структуру яких згенеровано випадковим чином. У вузлах ГА розміщуються макрофункції, що реалізовують операції лінійної алгебри та матричні операції різної розмірності.

Під час імітаційного моделювання використано параметри ФБ апаратних задач, синтезованих на ПЛІС із БКД операцій лінійної алгебри з можливістю варіювання зернистістю обчислювального середовища [165 – 167], ФБ для обчислення поліномів [145, 195] та спеціалізованих функціональних ядер [117, 131, 135, 136, 138, 185]. Параметри ФБ, зокрема, розміри прошивань і очікуваний час розв'язання апаратних задач, що використано як вхідні дані для моделювання, наведено в додатку А. Для

моделювання використовуються параметри ФБ, що синтезовано й описано у працях [15, 68, 54, 107].

Відповідно до виразу (2.10) під час моделювання враховано очікуваний час розв'язання апаратної задачі на апаратурі ПЛІС і розмір конфігураційного файлу (у бітах), що визначені під час синтезу та моделювання ФБ апаратних задач у САПР.

Для визначення часу налаштування ФБ апаратної задачі на кристалі ПЛІС використовується відомий вираз (2.14), запропонований у працях [15, 68]. На підставі експериментів [70] визначено пропорційну залежність між часом прошивання кристала ПЛІС і розміром прошивання, отриманим під час трасування проекту:

$$R_{exp} = \left(\frac{B_{exp}}{4} + 3 \right) \cdot 10^{-5}.$$

де R_{exp} – час реконфігурації кристала ПЛІС; B_{exp} – розмір файлу конфігураційних даних в байтах.

Під час моделювання функціональних процесів у РКС також використовуються результати експериментів, виконаних у працях [15, 68, 70], що визначають таке: час реконфігурації цілком залежить від розміру прошивання реконфігуровного модуля, час реконфігурації залежить від швидкодії інтерфейсу, через який виконується реконфігурація, для налаштування інтерфейсу реконфігурації необхідно лише три такти роботи системи, після реконфігурації система починає працювати одразу на наступний такт після відсилання останнього слова з прошивання реконфігуровного модуля.

6.2. Розроблення імітаційних моделей процесів обробки інформації в реконфігуровних комп'ютерних системах

6.2.1. Оцінювання адекватності імітаційної моделі реконфігуровної комп'ютерної системи до поставлених завдань моделювання. Адекватність

розробленої імітаційної моделі відносно реальної РКС доведемо шляхом верифікації [201, 202] і визначимо її валідність [203].

Для верифікації імітаційної моделі РКС засобами розробленого інтерфейсу моделювання виконаємо моделювання довільного графу вихідної обчислювальної задачі (рис. 6.7). Опис інтерфейсу користувача моделювального середовища для введення ГА вихідних обчислювальних задач наведено в додатку Д.

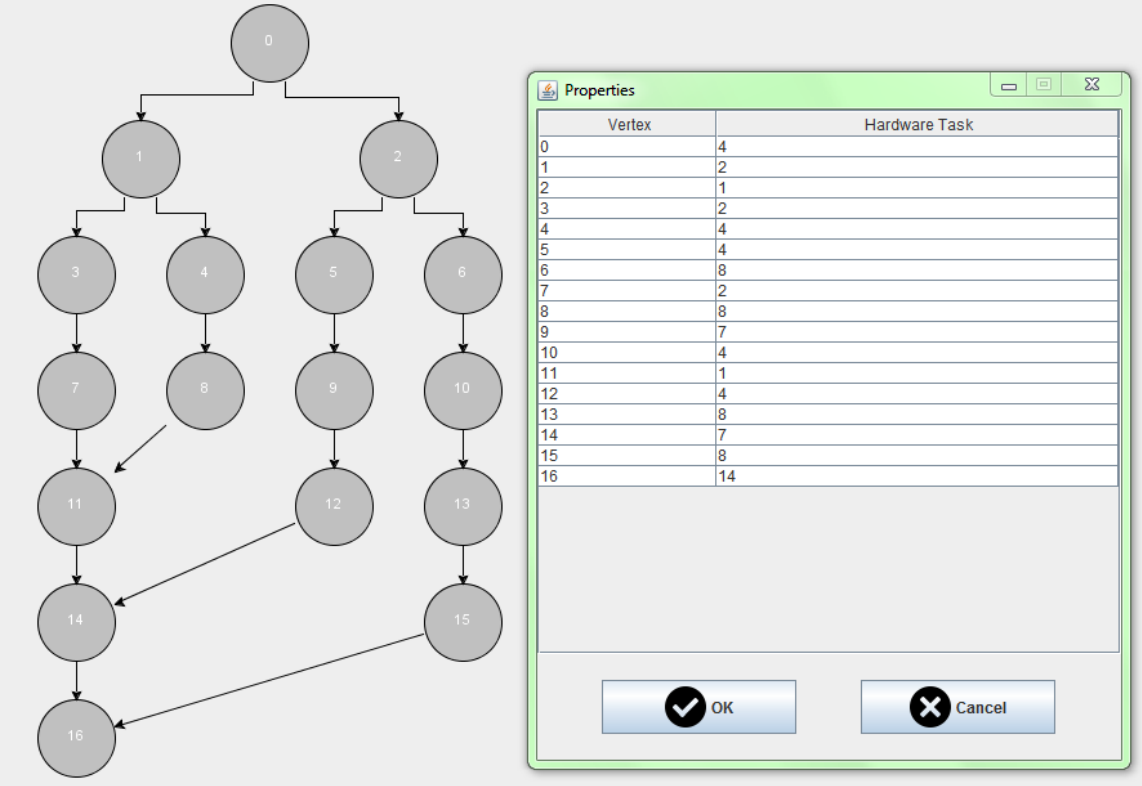


Рис.6.7. Граф алгоритму задачі в середовищі моделювання

Під час моделювання обчислювального процесу із запобіганням повторному передаванню конфігураційних даних (див. вираз (2.18)) отримано результат, візуалізацію якого ілюструє рис. 6.8.

Для верифікації моделювальних засобів виконується перевірка таких положень:

– система не може почати розв’язувати задачу з наступного рівня, поки не закінчила роботу будь-яка задача з попереднього рівня;

– система не може виконувати конфігурацію кількох апаратних задач одночасно, оскільки це послідовний процес, що керується контролером реконфігурації;

– першою на ярусі завжди запускається апаратна задача з найбільшим часом виконання;

– час роботи мережі при конфігурації не може бути більшим ніж значення параметра *networkMaxRandomTime*;

– кількість конфігурацій апаратних задач, що збережені на ПЛІС, не може бути більшою від значення параметра *fpgaSize*.

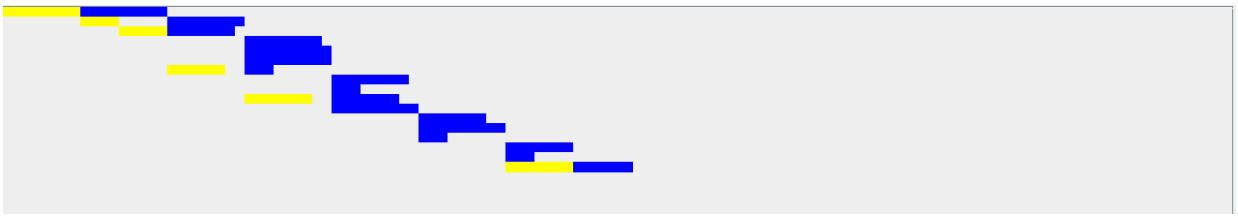


Рис. 6.8. Результати моделювання етапів виконання алгоритму в середовищі моделювання

Для перевірки наведених положень використаємо вбудований в *Java* механізм *assert* [204], який дозволяє перевіряти припущення про значення довільних даних у довільному місці програми, автоматично сигналізувати у разі виявлення некоректних даних, виявляти місце розташування некоректних даних, локалізувати і виправляти помилки, які призвели до некоректних даних. Приклад верифікації моделювальної програми вбудованими засобами *Java*, зображено на рис. 6.9.

Виходячи з аналізу правильності виконання даних положень, вважаємо, що імітаційна модель не має помилок у реалізації і є верифікованою.

Валідність програмної моделі доводить збіг результатів моделювання з теоретичними оцінюванням часу виконання за запропонованими в дисертаційній роботі математичними моделями і теоретичне моделювання певних функціональних процесів на підставі їх формалізацій [159]. Варто

зазначити, що отримані результати є умовно ідентичними, оскільки в процесі імітаційного моделювання враховується випадкова складова затримання мережевих каналів передавання даних, що максимально наближує параметри моделей функціональних процесів до реальних.

```
            switch (hardwareSystem.findConfiguration(t)) {
                case TSK_FPGA:
                    hardwareSystem.load(t);
                    System.out.printf("Load %s from FPGA%n", t);
                    break;
                case TSK_LIB:
                    int libTime = t.getBytestreamWords() * memoryAccessTime;
                    int randTime = new Random().nextInt(networkMaxRandomTime);
                    time.addSearchingAndLoading(id, libTime + randTime);
                    time.addLoadingLastWord(id, loadLastWordTime);
                    hardwareSystem.load(t);
                    System.out.printf("Load %s from library. LibTime = %s, randTime = %s%n", t, libTime, randTime);
                    break;
                case TSK_MEM:
                    int memTime = t.getBytestreamWords() * memoryAccessTime;
                    time.addSearchingAndLoading(id, memTime);
                    time.addLoadingLastWord(id, loadLastWordTime);
                    hardwareSystem.load(t);
                    System.out.printf("Load %s from memory. MemTime = %s%n", t, memTime);
                    break;
                default:
                    assert false : "Task can be found in FPGA or memory or library only!";
            }
```

```
ainFrame
"C:\Program Files\Java\jdk1.8.0_40\bin\java" ...
Start working with time: memoryAccess = 1 loadLastWord = 2 loadDatum = 0
Load hwN = 4 id = 0 from library. LibTime = 6, randTime = 0
java.lang.AssertionError: Task can be found in FPGA or memory or library only!
    at sim.Simulator.simulate(Simulator.java:88)
    at gui.MainFrame$Simulate.actionPerformed(MainFrame.java:284) <4 internal calls>
    at javax.swing.plaf.basic.BasicButtonListener.mouseReleased(BasicButtonListener.java:252) <31 internal calls>

Process finished with exit code 0
```

Рис. 6.9. Приклад верифікації програми моделювання вбудованими засобами *Java*

Викладене вище доводить правильність функціонування імітаційної моделі РКС, її відповідність поставленому завданню дослідження і, відповідно, достовірність та обґрунтованість отриманих результатів моделювання.

6.2.2. Опис середовища моделювання. Програмний код реалізації основних модулів моделювальної програми мовою програмування *Java*

подано в додатку Е. Наведемо основні класи програми моделювання обчислювального процесу:

Клас ***Frame*** – абстрактний підклас *JFrame*, що описує всі вікна в програмі. Він містить кілька методів, необхідних для всіх вікон, наприклад, методи сповіщень користувача, наприклад *showError*(), *showInfo*(), *showQuestion*() та *showWarning*(), зберігає в собі абстрактний клас *Action*, що є батьківським для всіх об'єктів певних дій.

Клас ***MainFrame*** – підклас *Frame*, що виконує функції головного вікна, функцію запуску проекту; серед основних методів – метод *main*() входу в додаток та *makeTaskLevels*(), що виконує перетворення ГА в зручні для моделювання структури даних; він також містить класи-нащадки *Action*, які потрібні для прив'язки до певних елементів керування додатка, наприклад, кнопок панелі інструментів і пунктів меню.

Клас ***LibraryFrame*** – підклас *Frame*, необхідний для роботи з даними центральної БКД.

Клас ***LinkerFrame*** – підклас *Frame*, необхідний для додавання з'єднань між вершинами у ГА.

Клас ***PropertiesFrame*** – підклас *Frame*, необхідний для призначення кожній вершині у ГА певного номера апаратної задачі.

Клас ***GantDiagramPanel*** **–** підклас *JPanel*, **о**б'єкти якого виконують візуалізацію діаграми Ганта.

Клас ***GraphPanel*** **–** підклас *JPanel*, являє собою робоче поле для створення графу обчислювальної задачі; методи *addVertex*() та *addEdge*() використовуються для додавання вершин та дуг графу, які зберігаються в полях *vertexes* та *edges* відповідно; для відображення графу використовуються компоненти *mxGraph* і *mxGraphComponent* із бібліотеки *JGraphX*; методи *getPropertiesData*() та *createTransitions*() формують масив номерів апаратних задач та матрицю переходів відповідно.

Клас ***TimeTracks*** – модельний час програми для кожної вершини алгоритму – діаграма Ганта в текстовому вигляді; об'єктами є набір

функціональних методів, наприклад, запуск завдань на виконання – *addCounting*(), додавання затримок реконфігурації та завантаження даних – *addLoadingData*(), *addSearchingAndLoading*(), *addLoadingLastWord*(), *addWaiting*().

Клас ***SettingsHolder*** – відображення файлу налаштувань у пам'ять; інтерфейс доступу до часових затримок та інших параметрів моделювання за допомогою методів: *getLoadLastWordTime*(), *getFpgaSize*(), *getBonus*(), *getMemoryAccessTime*(), *getLoadDatumTime*(), *getNetworkMaxRandomTime*(), *getMemorySize*(); стандартні значення параметрів, що застосовуються у разі можливих помилок завантаження файлу; файл формату *xml* має можливість редагування в текстовому редакторі; процедура зберігання та відновлення даних використовує клас *XStream* з бібліотеки *xstream*.

Клас ***Library*** – відображення файлу БКД у пам'ять, що містить набір функцій, які забезпечують доступ до даних, зберігає масив даних (*data*) і назву файлу з даними (*LIBRARY_FILE*) у форматі *xml*.

Клас ***Task*** – апаратна задача з параметрами: час роботи (*workTime*), кількість слів у бітовому потоці (*bytestreamWords*) та кількість слів даних – параметрів (*dataCount*), що зберігаються в БКД за номером апаратної задачі (*hwN*), кожна з яких має свій унікальний ідентифікатор (*id*). Метод порівняння *compareTo*() виконує сортування завдань у черзі на підставі порівняння часу їх виконання.

Клас ***HardwareSystem*** – стан системи моделювання; метод *findConfiguration*() виконує пошук конфігураційних даних апаратної задачі: повертає керувальні сигнали *TSK_FPGA*, *TSK_MEM*, *TSK_LIB*; метод *load*() завантажує конфігураційні дані на ПЛІС, нараховує «бонуси», вивантажує апаратні задачі в ЛПК і БКД; локальна пам'ять конфігураційних даних подана масивом *memory*; реконфігуровна область представлена масивом *FPGA*; агрегує об'єкт класу *Library*, що містить центральну БКД та список *bonuses*.

Клас ***Simulator*** – моделювання роботи РКС, що зберігає об'єкт класу *HardwareSystem* та об'єкт класу *SettingsHolder* для доступу до параметрів

моделювання; функція моделювання *simulate()*, що повертає діаграму Ганта в текстовому вигляді.

Клас ***Testbench*** – автоматизоване виконання великої кількості повторень експериментів для збирання статистичних даних моделювання.

Програмна модель РКС дозволяє виконувати експерименти в автоматизованому режимі. Кожний експеримент виконується в три етапи: отримання вихідних даних, здійснення певної кількості повторів експерименту, збереження усереднених результатів.

На етапі отримання вихідних даних моделювальний стенд звертається до переданих йому параметрів та до класу *SettingsHolder* для отримання таких даних: кількості експериментів (*EXPERIMENTS_COUNT*), назви файлів, у яких зберігаються ГА для кожного експерименту (*files*), кількості повторів кожного експерименту (*REPEAT_COUNT*), часових затримок на реконфігурацію (*LoadLastWordTime*), на доступ до пам’яті (*MemoryAccessTime*), на завантаження даних (*LoadDatumTime*), на доступ до мережі (*NetworkMaxRandomTime*) та інші параметри моделювання як, наприклад, розмір реконфігуровної області ПЛІС (*FpgaSize*), розмір ЛПК (*MemorySize*), початкова кількість «бонусів» (*Bonus*).

Наступним етапом є здійснення повторів експерименту, коли кожен запланований експеримент виконується задану кількість разів. Для отримання незалежних результатів щоразу перед початком роботи створюється нове моделювальне середовище. Оскільки для досліджень становить інтерес саме час виконання, зберігаються такі результати прогону, як час роботи алгоритму, час обчислень та час роботи алгоритму без застосування методу пришвидшення.

На заключному етапі на підставі отриманих даних про кожну серію експериментів обчислюються середні значення часу і формується звіт про виконання експериментів.

Для проведення експериментів використовувалась обчислювальна система з такими апаратними характеристиками:

– процесор: *AMD Phenom* II *X*4 960T *Processor*(3000 МГц, 4 ядра, 6 Мбайт кешу третього рівня);

– оперативна пам’ять: *DDR*3 *SDRAM*, 6144 Мбайт.

Програмне забезпечення:

– операційна система: *Microsoft Windows* 7 *x*64 *SP*1 (*Version* 6.01.7601);

– середовище *Java*: *Java SE Runtime Environment* (*build* 1.8.0_65-b17);

– віртуальна машина *Java*: *HotSpot* 64-bit S*erver VM* (*build* 25.65-b01);

– середовище розроблення і компіляції *Java* програм: *Jet Brains IntelliJ IDEA* 15.0.4.

6.3. Моделювання методу адаптивного відображення задач на реконфігуровне обчислювальне середовище

6.3.1. Планування експериментів. Під час імітаційного моделювання перевіримо положення, на яких ґрунтуються теоретичні висновки, зроблені в дисертаційній роботі на підставі математичного моделювання.

Положення 1. За великої кількості однотипних функцій в алгоритмі обчислювальної задачі комунікаційні витрати зменшуються лінійним чином усуненням повторного завантаження конфігураційних даних відповідно до виразів (2.24) – (2.27).

Положення 2. На пришвидшення обчислювального процесу значним чином впливає співвідношення частоти розв’язання функцій нових типів, ступеня розпаралелювання алгоритму (що вимірюється шириною ГА) і просторових параметрів реконфігуровного обчислювального середовища для розміщення ФБ апаратних задач або їх копій. Зважаючи на додаткові затримки під час створення копій ФБ, коли на всій ширині ярусу ГА виконується велика

кількість однотипних завдань, їх виконання буде швидшим ніж виконання різнотипних завдань, що завантажуються із сховищ конфігураційних даних. Додатковий час на копіювання конфігураційних даних ФБ залежить від інтенсивності надходження однотипних завдань в процесі виконання алгоритму обчислювальної задачі, тобто від зміни ситуації від ярусу до ярусу, збігу ширини ярусу і вільного місця на реконфігуровному обчислювальному середовищі, ресурсу внутрішньої пам'яті кристала ПЛІС.

Положення 3. Під час виконання апаратних задач з високим показником апаратного пришвидшення згідно з виразом (2.12) і задач, час розв'язання яких більший від часу завантаження, копіювати конфігураційні дані ФБ і кешувати їх ефективно з застосуванням ЛПК, зберігаючи ресурси внутрішньої пам'яті та реконфігуровної ділянки ПЛІС з врахуванням часових обмежень.

Приклад графу з великою кількістю повторень однотипних завдань зображено на рис. 6.10. Приклади графів з різною інтенсивністю надходження різнотипних завдань на початку (випадок I) і наприкінці (випадок II) графу алгоритму зображено на рис. 6.11 і 6.12 відповідно.

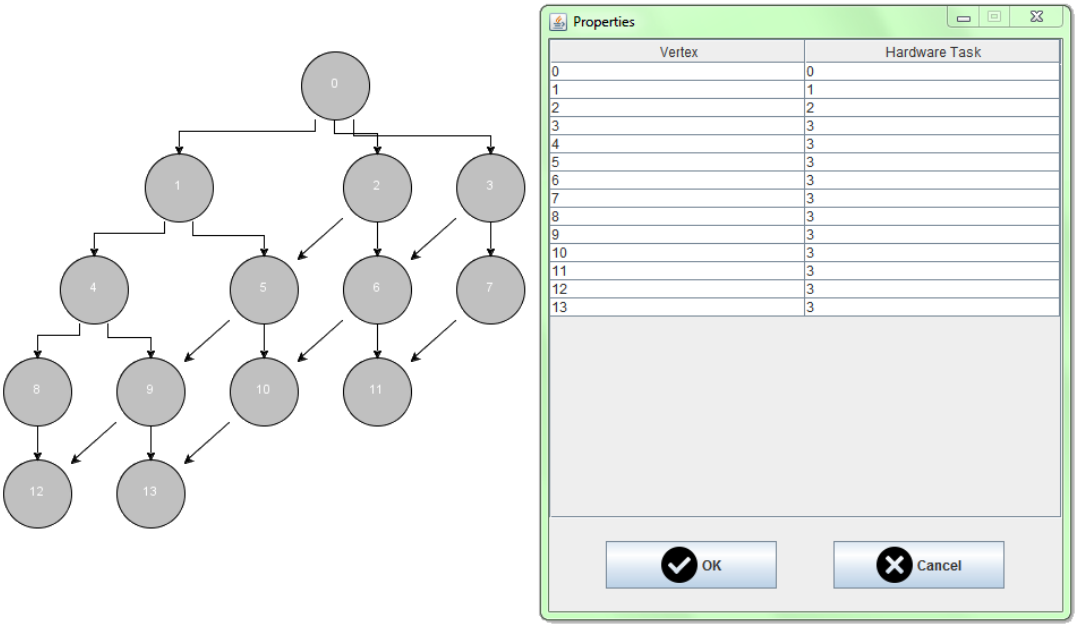


Рис. 6.10. Приклад графу алгоритму з великою кількістю повторів однотипних задач

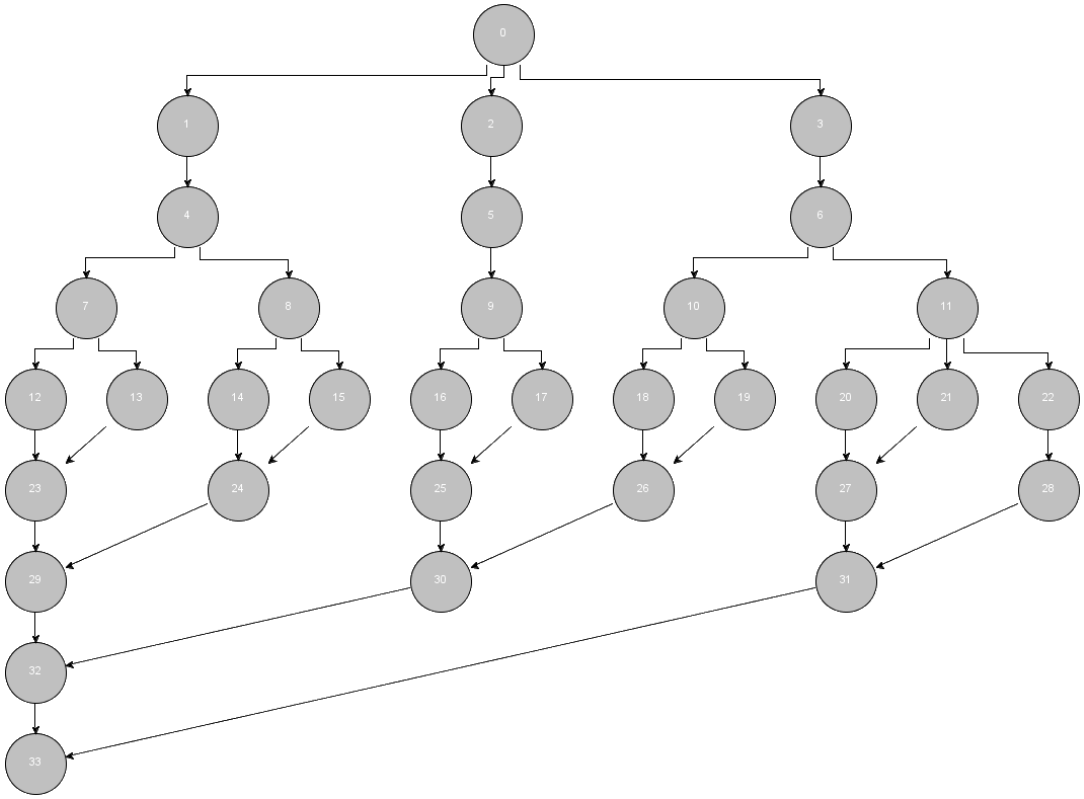


Рис. 6.11. Приклад графу алгоритму з високою інтенсивністю надходження однотипних завдань наприкінці алгоритму (випадок I)

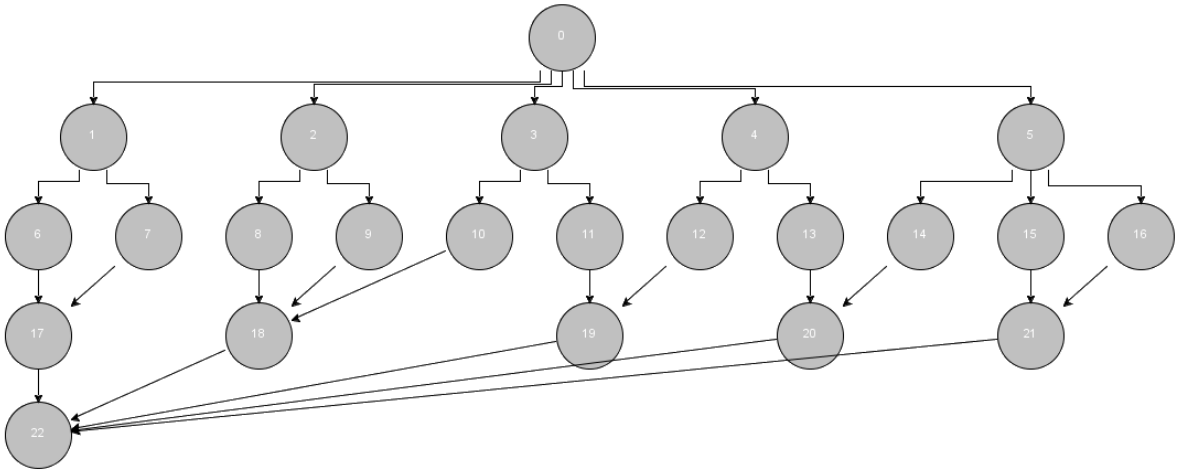


Рис. 6.12. Приклад графу алгоритму з високою інтенсивністю надходженнями однотипних задач на початку алгоритму (випадок II)

6.3.2. Результати моделювання. Для отримання достовірних даних з урахуванням випадкової складової затримань під час мережевого передавання даних моделювання виконання алгоритмів обчислювальних задач здійснюється велику кількість разів (до 100 разів).

Експериментальні часові оцінювання отримано в абстрактних одиницях – машинних тактах. Для оцінювання ефективності пришвидшення реконфігурації використаємо абсолютний коефіцієнт пришвидшення виконання обчислень. Досліджено залежність часу розв'язання обчислювальних задач від кількості однотипних функцій в алгоритмі. На графіках, зображених на рис. 6.13 і 6.14, подано результати моделювання, з яких видно, що характер залежностей збігається з результатами досліджень апаратної реалізації ОМ РКС (див. рис. 5.19, 5.20).

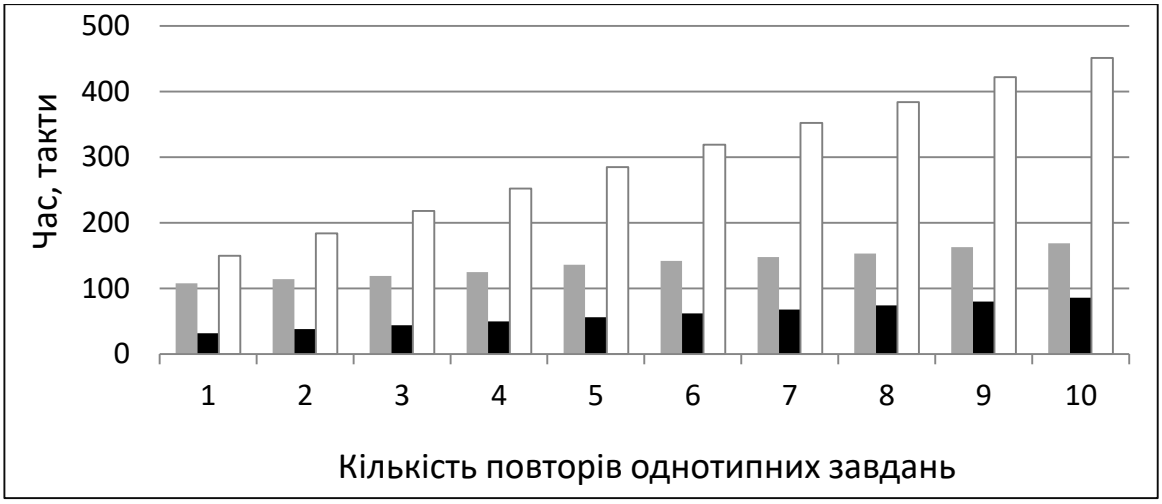


Рис. 6.13. Залежність часу розв'язання від кількості повторів однотипних задач без обмежень реконфігуровного обчислювального середовища: ■ – час обчислення на апаратурі ПЛІС; □ – стандартний процес реконфігурації; ▒ – застосування комплексного підходу до пришвидшення реконфігурації

Видно, що стандартна послідовність реконфігурації супроводжується значними затримками, що значно переважають час апаратних обчислень на реконфігуровному обчислювальному середовищі. Комплексне застосування запобігання повторному завантаженню конфігураційних даних і завчасної реконфігурації інтенсивно зменшує затримки часу реконфігурації. Моделювання методу адаптивного відображення на моделі реконфігуровного обчислювального середовища без просторових обмежень показало, що велика кількість однотипних функцій в алгоритмі також вносить затримки в процес

відображення завдань, які зумовлені копіюванням великої кількості ФБ на поверхні ПЛІС. Ці затримки зменшуються в області збігу параметрів обчислювальної задачі, а саме кількості однотипних задач і ширини ярусу ЯПФ ГА.

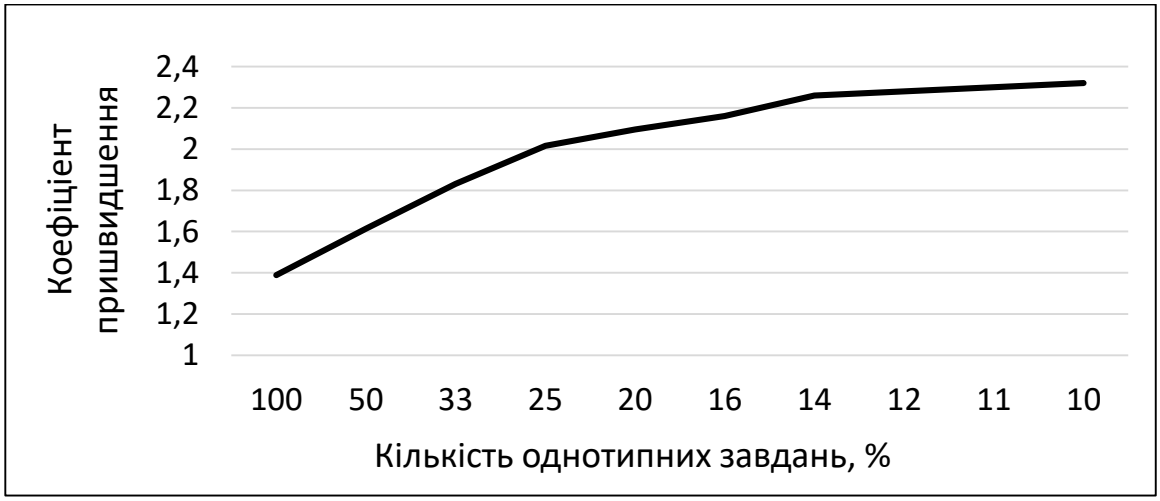


Рис. 6.14. Дослідження коефіцієнта пришвидшення обчислення з застосуванням завчасної реконфігурації

Дослідження методу адаптивного відображення на моделях реконфігуровного обчислювального середовища з обмеженнями розміру реконфігуровного обчислювального середовища (рис. 6.15 і 6.16) показало, що, окрім впливу кількості однакових завдань, на час відображення також впливає і співвідношення ширини ярусу ГА і розміру реконфігуровного обчислювального середовища.

Із графіків залежностей на рис. 6.15 і 6.16 видно, що мінімальний час обчислення досягається за умови збігу ширини ЯПФ графу і просторових параметрів реконфігуровного обчислювального середовища ПЛІС. За кількості однотипних завдань більше ніж 50% збільшення непродуктивного часу зумовлено запровадженням додаткових засобів для копіювання активних ФБ апаратних задач. Для раціонального використання запропонованого методу час виконання завдань вкладається в певні часові обмеження.

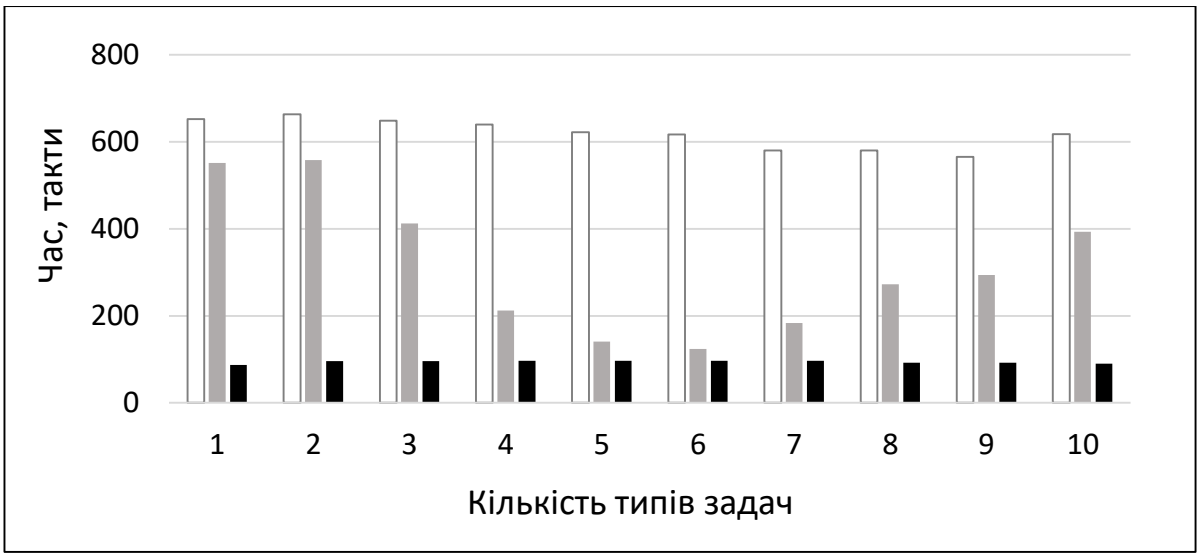


Рис. 6.15. Залежність часу виконання від кількості повторів однотипних задач з врахуванням обмежень реконфігуровного обчислювального середовища: ■ – час обчислення на апаратурі ПЛІС; □ – стандартний процес реконфігурації; ■ – застосування комплексного підходу до пришвидшення реконфігурації

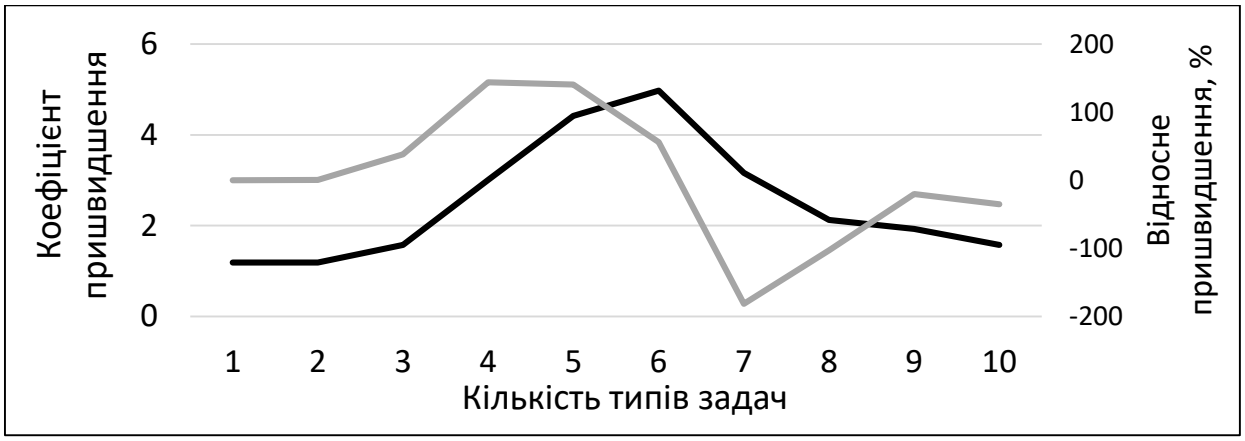


Рис. 6.16. Залежність коефіцієнта пришвидшення від кількості повторів однотипних задач і просторових обмежень реконфігуровного обчислювального середовища: ━ – коефіцієнт пришвидшення; ━ – інтенсивність пришвидшення

Із графіку дослідження показника пришвидшення (рис. 6.16) видно, що механізм повторного використання ресурсів надає інтенсивного пришвидшення в середньому до 63% за умови сумірної з розміром

реконфігуровної області ширини ЯПФ графу та високої кількості однотипних завдань. При цьому в процесі подолання просторових обмежень ПЛІС різко зменшується інтенсивність пришвидшення реконфігурації в середньому до 85%.

Характер кривих на рис. 6.16 обґрунтовує основну тенденцію залежності часу обчислення від співвідношення параметрів задач і параметрів реконфігуровного обчислювального середовища, яка лягла в основу визначення критеріїв оптимізації процесу обробки інформації у третьому розділі дисертаційної роботи (див. рис. 3.1 і 3.2).

Запропоновані методи і засоби підвищення ефективності ґрунтуються на реалізації багаторівневого кешування конфігураційних даних, що реалізовано на рівні імітаційної моделі ОМ РКС. Зроблено припущення, що залежно до співвідношення параметрів задачі та параметрів реконфігуровного обчислювального середовища немає сенсу в копіюванні ФБ на поверхні ПЛІС через внутрішню пам’ять кристалів ПЛІС, що також сприяє підвищенню ефективності обробки інформації шляхом зберігання ресурсів пам’яті. Моделювання (рис. 6.17) показало, що за невідповідності ширини реконфігуровного обчислювального середовища до ширини ЯПФ ГА копіювання через внутрішню пам’ять дає змогу в 1.25 разу пришвидшити процес обчислень. Інакше, в області оптимального співвідношення параметрів обчислювальних задач і реконфігуровного обчислювального середовища, завантаження копій ФБ із зовнішньої ЛПК не впливає на ефективність обчислювального процесу (рис. 6.18).

Графіки залежності часу виконання обчислювального алгоритму і коефіцієнта пришвидшення з використанням різних стратегій копіювання ФБ на поверхні реконфігуровного обчислювального середовища показано на рис. 6.17 – 6.20.

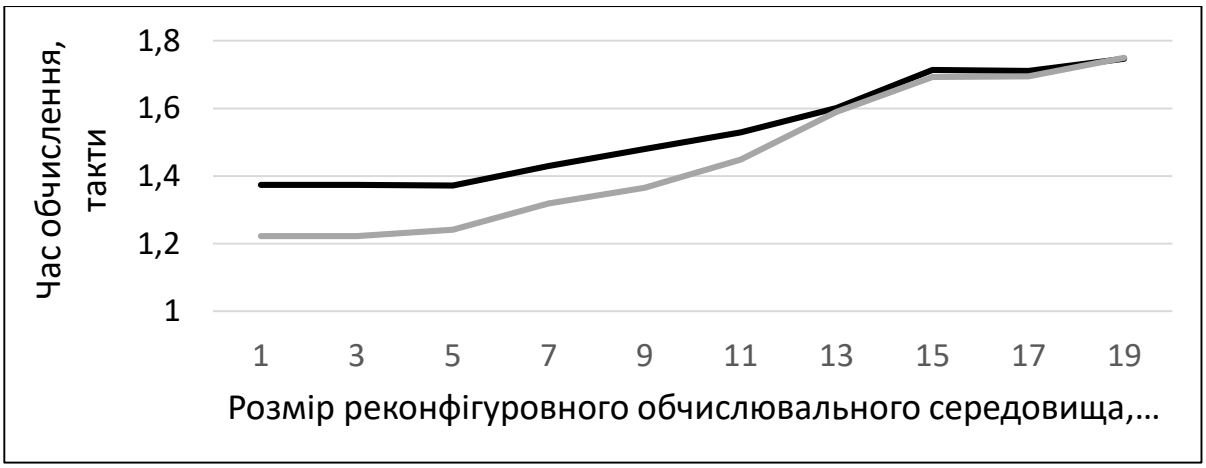


Рис. 6.17. Залежність коефіцієнту пришвидшення від розмірності реконфігуровного обчислювального середовища і частоти надходження завдань різних типів: ━ – адаптивне відображення з копіюванням ФБ через ROM на ПЛІС (випадок І); ━ – адаптивне відображення завдань з завантаженням копій ФБ із ЛПК (випадок І)

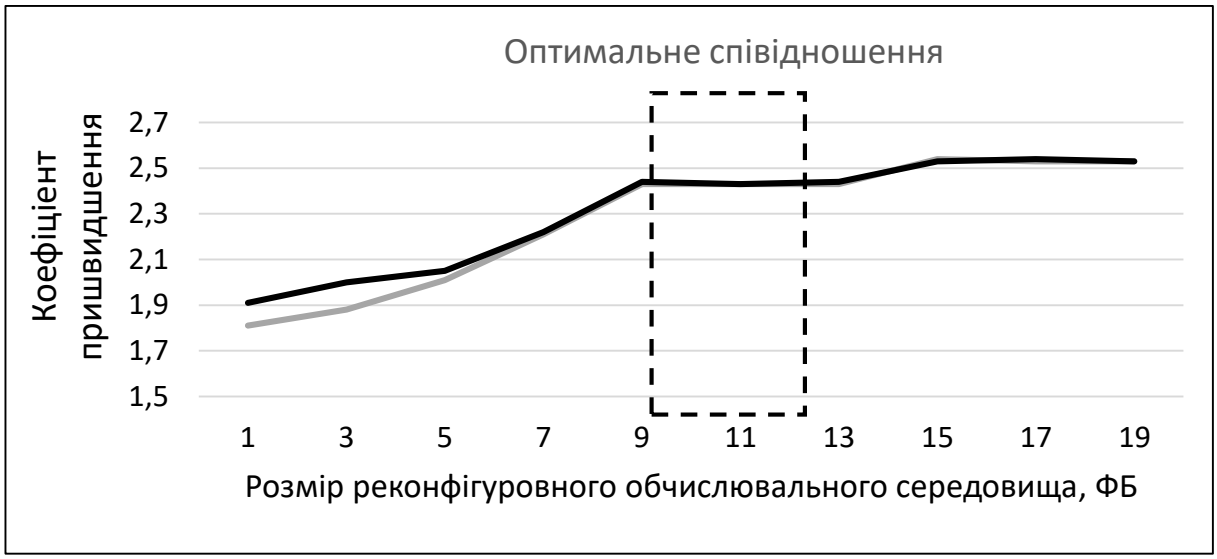


Рис. 6.18. Залежність коефіцієнту пришвидшення від розмірності реконфігуровного обчислювального середовища і частоти надходження завдань різних типів: ━ – адаптивне відображення з копіюванням ФБ через ROM на ПЛІС (випадок ІІ); ━ – адаптивне відображення завдань з завантаженням копій ФБ із ЛПК (випадок ІІ)

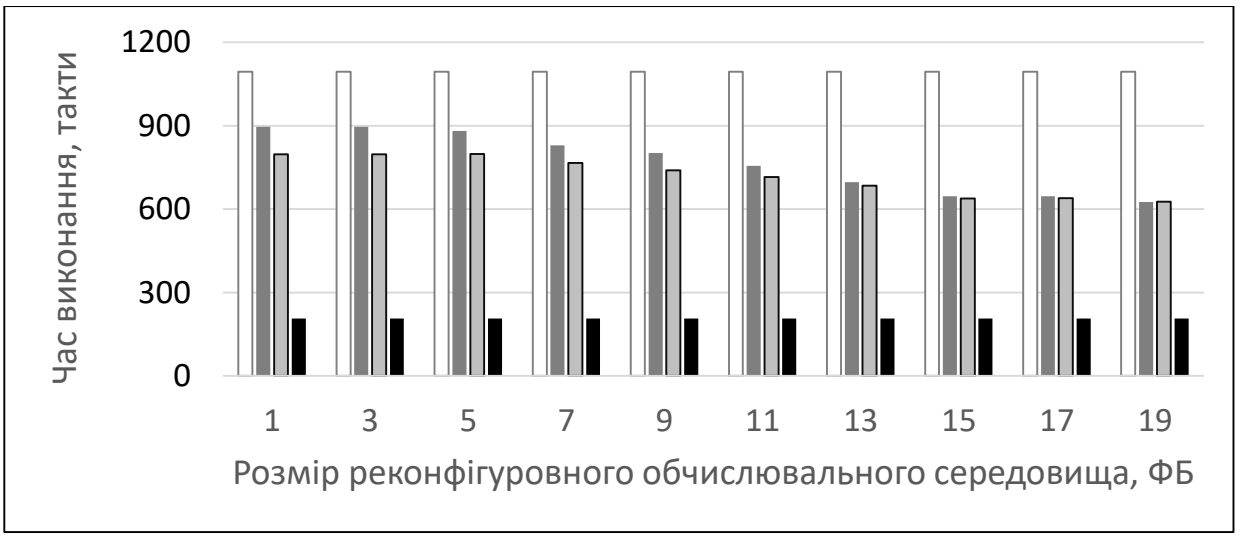


Рис. 6.19. Залежність часу виконання алгоритму від розмірності реконфігуровного обчислювального середовища і частоти надходження завдань різних типів (випадок I): ■ – час обчислення на апаратурі ПЛІС; □ – стандартний процес реконфігурації; ▒ – адаптивне відображення завдань з завантаженням копій ФБ із ЛПК; ▒ – адаптивне відображення з копіюванням ФБ через ROM на ПЛІС

Виконані експерименти на базі розробленої імітаційної моделі РКС довели ефективність запропонованих методів та засобів організації обробки інформації у динамічно РКС. Дослідження підтвердили основні теоретичні положення, що зроблено на підставі розроблених математичних моделей функціональних процесів:

– запропоновані методи та засоби забезпечують лінійне пришвидшення обчислювального процесу за оптимальних умов відображення;

– під час реалізації адаптивного відображення за великої кількості однотипних функцій час відображення завдань інтенсивно зменшується, але збільшується час виконання завдань на апаратурі ПЛІС унаслідок дублювання екземплярів ФБ;

– використання методу адаптивного відображення задач дає змогу в середньому в 2,13 разу пришвидшити процес обробки інформації в РКС за рахунок зменшення комунікаційних витрат в області оптимального

співвідношення параметрів алгоритмів обчислювальних задач і просторових параметрів реконфігуровного обчислювального середовища.

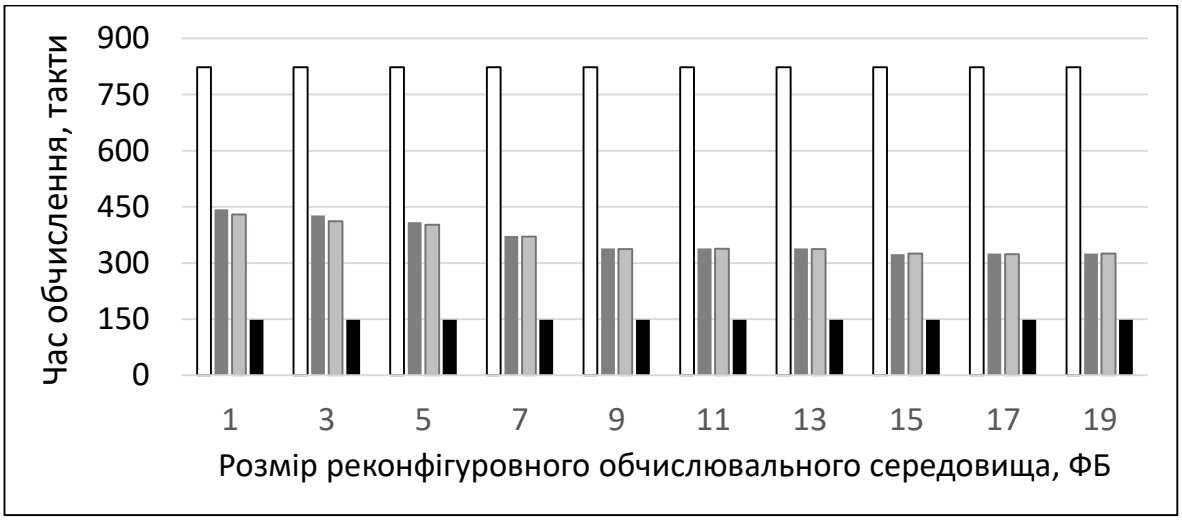


Рис. 6.20. Залежність часу виконання алгоритму від розмірності реконфігуровного обчислювального середовища і частоти надходження завдань різних типів (випадок II): ■ – час обчислення на апаратурі ПЛІС; □ – стандартний процес реконфігурації; ■ – адаптивне відображення завдань з завантаженням копій ФБ із ЛПК; □ – адаптивне відображення з копіюванням ФБ через ROM на ПЛІС

Під час дослідження показників пришвидшення реконфігурації із застосуванням запропонованого методу отримано середній коефіцієнт пришвидшення реконфігурації, що дорівнює 2,5. Зменшення інтенсивності пришвидшення реконфігурації в середньому на 85% спостерігається зі збільшенням кількості типів задач. За сумірної з розміром реконфігуровної області ширини ЯПФ графу та високої кількості однотипних задач досягається збільшення інтенсивності пришвидшення реконфігурації в середньому на 63%. Середній показник збільшення інтенсивності пришвидшення реконфігурації із застосуванням методу пришвидшення реконфігурації порівняно зі стандартною послідовністю реконфігурації досягає 4,3%.

ВИСНОВКИ ДО РОЗДІЛУ 6

1. У цьому розділі проведено експерименти, що полягають у виконанні моделювання методу адаптивного відображення обчислювальних задач на реконфігуровне обчислювальне середовище, поданих ЯПФ графів алгоритмів, та виконано аналіз результатів.

2. За результатами досліджень показано, що запропонований метод забезпечує інтенсивне пришвидшення реконфігурації за рахунок видалення комунікаційних витрат, якщо параметри задачі відповідають структурі реконфігуровного обчислювального середовища, тобто якщо ширина ГА сумірна з розміром динамічної ділянки ПЛІС. У противному випадку, відбувається витіснення функціональних блоків у ЛПК, що потребує часу, порівняно із завантаженням конфігураційних даних із ЛПК.

3. Оскільки повторне використання ресурсів забезпечує пришвидшення обчислень зі збільшенням кількості однотипних завдань в ГА, під час експериментів було доведено, що метод адаптивного відображення обчислювальних задач надає інтенсивного пришвидшення зі збільшенням типів завдань до кількості, сумірної із шириною ЯПФ графу.

4. На час реконфігурації впливає інтенсивність надходження великої кількості завдань різних типів в процесі виконання алгоритму. З’ясовано, що, якщо така критична ділянка розташована умовно наприкінці алгоритму, значно зростає імовірність попереднього кешування потрібних для виконання завдань або на поверхні реконфігуровного обчислювального середовища, або в ЛПК. Отже, запропонований метод адаптивного відображення обчислювальних задач забезпечує пришвидшення виконання обчислень на критичних ділянках ГА.

5. Практична значущість запропонованих засобів адаптивного відображення обчислювальних задач полягає в такому:

– використання поверхні динамічної ділянки ПЛІС для кешування функціональних блоків дозволяє вирішити проблему зберігання ресурсів

внутрішньої пам’яті ПЛІС у широких класах реконфігуровних комп’ютерних систем;

– метод адаптивного відображення обчислювальних задач через реалізацію механізмів повторного використання ресурсів ФБ і завчасної реконфігурації обчислювального середовища дає змогу видалити майже весь непродуктивний час процесу реконфігурації обчислювального середовища, що підвищує ефективність РКС;

– багаторівнева структура пам’яті та апаратні засоби для реалізації пошуку і підтримання даних дають змогу реалізовувати апаратні багатовимірні бази даних і апаратні системи керування базами даними, що сприяє підвищенню ефективності збереження й обробки інформації в спеціалізованих обчислювальних системах, зокрема в розподілених обчислювальних системах, для розв’язання задач керування в обмеженому режимі часу;

– використання імітаційної моделі реконфігуровної комп’ютерної системи дає змогу виділяти реконфігуровні обчислювальні ресурси в наближеному до реального режимі часу, що є зручним інструментом для моделювання і дослідження часових характеристик функціональних процесів обробки інформації в динамічно реконфігуровних комп’ютерних системах.

ВИСНОВКИ

У дисертації наведено теоретичне узагальнення і нове вирішення наукової проблеми, що полягає в розвитку теорії організації обробки інформації в комп'ютерних системах на ПЛІС з урахуванням їх функціональних та апаратурних обмежень.

Запропоновані методи та засоби включають в себе взаємозв'язані вирішення завдань оптимізації процесу обробки інформації з урахуванням співвідношення параметрів розв'язуваних задач і структури обчислювального середовища шляхом визначення оптимальної зернистості обчислень і зменшення накладних витрат процесу відображення задач на реконфігуровне обчислювальне середовище, що в цілому забезпечує підвищення ефективності обробки інформації в реконфігуровних комп'ютерних системах на ПЛІС.

Запропоновано нову стратегію взаємної адаптації розв'язуваних задач і обчислювального середовища на ПЛІС, що ґрунтується на варіюванні зернистістю обчислень під час розв'язання задач великої розмірності, та вдосконалено концепцію реалізації локальних розподілених засобів керування відображенням задач на реконфігуровне обчислювальне середовище, що забезпечує підвищення ефективності врахування фізичних параметрів кристалів ПЛІС на всіх рівнях реалізації комп'ютерної системи.

Основні наукові та практичні результати роботи полягають в наступному.

1. Виконано аналіз структурної і функціональної організації динамічно реконфігуровних комп'ютерних систем, на підставі якого розроблено критерії ефективності їх функціонування, які на відміну від відомих ґрунтуються на вдосконаленні показника пришвидшення обчислень шляхом урахування часової складності процесу відображення обчислювальних задач на реконфігуровне обчислювальне середовище.

2. Розроблено математичні моделі функціональних процесів, що на відміну від відомих, ураховують багаторівневу структуру динамічно реконфігуровних комп'ютерних систем і непродуктивні витрати на всіх рівнях

їх функціонування. Це дозволяє виконати аналіз і оцінити можливість оптимізації процесу обробки інформації за розробленим інтегральним критерієм, який ґрунтується на співвідношенні параметрів розв’язуваних задач, структури обчислювального середовища та непродуктивних витрат процесу його реконфігурації з урахуванням апаратурних обмежень ПЛІС.

3. Запропоновано новий спосіб організації віртуальної пам’яті в реконфігуровних комп’ютерних системах, який відрізняється від відомих багаторівневою багатофункціональною структурою для зберігання, пошуку і керування конфігураційними даними. Багаторівневе кешування конфігураційних даних дає змогу реалізації різних стратегій обслуговування завдань, які різняться частотою їх виконання, непродуктивним часом та обсягом апаратурних витрат. Реалізація концепції асоціативного пошуку в багаторівневій пам’яті керувальних даних дозволяє скоротити ємність використання внутрішньої пам’яті ПЛІС у 64 рази порівняно з реалізацією пам’яті з адресним доступом.

4. Запропоновано новий спосіб скорочення критичного часу виконання обчислювальних задач, який на відміну від відомих інтегрує статичний і динамічний підходи до трансформації графів обчислювальних задач на базі модифікованого методу гілок і границь. Використання способу забезпечує інтенсивне скорочення непродуктивних витрат процесу відображення задач на реконфігуровне обчислювальне середовище таким чином, що непродуктивний час вимірюється лише часом реконфігурації завдань першого ярусу ЯПФ графу.

5. Запропоновано й обґрунтовано новий метод адаптивного відображення задач на реконфігуровне обчислювальне середовище, який відрізняється від відомих скороченням критичного часу виконання обчислювальних задач на базі багаторівневого кешування конфігураційних даних. Це дає змогу реалізувати ефективні стратегії організації процесу відображення задач за обсягом непродуктивного часу і апаратурних витрат з урахуванням несталих умов відображення. Реалізація різних стратегій

відображення дозволяє від 2,5% до 4,3% скоротити непродуктивний час процесу обробки інформації.

6. Запропоновано й обґрунтовано новий метод оптимізації процесу відображення задач на реконфігуровне обчислювальне середовище, який відрізняється від відомих визначенням стратегії обслуговування завдань на підставі аналізу показника їх апаратного пришвидшення. Оптимізація непродуктивного часу відображення за розробленим інтегральним критерієм забезпечує потрібний час виконання обчислювальних задач, скорочує загальну кількість відхилень виконання завдань і в цілому підвищує на 10% користувацьку ефективність реконфігуровних комп'ютерних систем шляхом зменшення впливу апаратурних обмежень ПЛІС на непродуктивний час відображення.

7. Уперше визначено й обґрунтовано критерій швидкодії реконфігуровного обчислювального середовища на ПЛІС, який на відміну від відомих ґрунтується на визначенні співвідношення часу виконання завдань на апаратурі ПЛІС і обсягу корисних даних для обчислення з урахуванням затримок поширення сигналів на фізичному рівні функціональних блоків і апаратурних обмежень ПЛІС. Запропонований критерій дозволяє оцінити ефективність процесу обробки даних на фізичному рівні реконфігуровних комп'ютерних систем під час розв'язання задач великої розмірності.

8. Запропоновано нову стратегію організації процесу обробки інформації, яка відрізняється від відомих взаємною адаптацією задач і обчислювального середовища на підставі визначення оптимальної зернистості обчислень за розробленим критерієм максимальної швидкодії реконфігуровного обчислювального середовища. Застосування нової стратегії забезпечує підвищення користувацької ефективності реконфігуровних комп'ютерних систем у середньому до 12%, залежно від технічних показників кристалів ПЛІС, порівняно з реалізацією великозернистих обчислень.

9. Удосконалено структурну організацію бібліотеки функціональних ядер на ПЛІС, яка відрізняється від відомих формуванням набору

функціональних блоків з оптимальними характеристиками продуктивності на базі розробленого критерію швидкодії реконфігуровного обчислювального середовища. Це дозволяє варіювати зернистістю обчислень з метою підвищення ефективності виконання обчислювальних задач великої розмірності.

10. Запропоновано й обґрунтовано нові рівні абстракції розгляду архітектури реконфігуровних обчислювальних систем, які на відміну від відомих локалізують процес реконфігурації обчислювального середовища на структурному рівні обчислювальних модулів. Це забезпечує підвищення ефективності процесу обробки інформації за рахунок застосування на кожному рівні ефективних методів та засобів організації обчислень, у тому числі методів та засобів, запропонованих у дисертаційній роботі.

11. Модифіковано метод автоматичного розподілу завдань і синхронізації процесів на базі моделі обчислень, керованих потоком даних, який відрізняється від відомих механізмом завчасної реконфігурації обчислювального середовища, дворівневою організацією синхронізації процесів і вдосконаленим способом підвищення відмовостійкості керувального ядра. Застосування модифікованого методу дозволило підвищити ефективність процесу обробки інформації на локальному рівні обчислювального модуля на 10,25% порівняно з традиційними технологіями паралельної обробки інформації на рівні операційної системи, а також підвищити надійність реконфігуровних комп’ютерних систем.

12. Розроблені математичні моделі, методи і засоби доведені до практичної реалізації у вигляді програмного продукту і промислових зразків.

СПИСОК ЛИТЕРАТУРИ

1. TOP 10 Sites for november 2016 [Електронний ресурс]. – Copyright 1993-2016 TOP500.org, 2017. – Режим доступу : https://www.top500.org/lists/2016/11/

2. Hilbert M. The world’s technological capacity to store, communicate, and compute information / M. Hilbert, P. López // Science. – 2011. – Vol. 332 (6025). – P. 60 – 65.

3. The most used interconnect on the TOP500 supercomputing list [Електронний ресурс]. – Mellanox Technologies, 2017. – Режим доступу : http://www.mellanox.com/related-docs/applications/TOP500_Nov_2015.pdf

4. Луцкий Г. М. Повышение эффективности кластеров на основе Infiniband / Г. М. Луцкий, И. С. Райзин // Вісник університету «Україна». Інформатика, обчислювальна техніка та кібернетика. – 2011. – № 8. – С. 133.

5. Абрамов С. М. Суперкомпьютеры «СКИФ» / С. М. Абрамов, В. Ф. Заднепровский, Е. П. Лилитко // Информационные технологии и вычислительные системы. – 2012. – № 1. – С. 3 – 16.

6. Каляев И. А. Архитектура семейства реконфигурируемых вычислительных систем на основе ПЛИС / И. А. Каляев, И. И. Левин, Е. А. Семерников // Штучний інтелект. – 2008. – № 3. – С. 663 – 673.

7. Воеводин В. В. Параллельные вычисления / В. В. Воеводин, Вл. В. Воеводин – С.- Петербург : «БХВ-Петербург», 2004. – 599 с.

8. Мельник А.О. Архітектура комп'ютера / А. О. Мельник. – Луцьк : Волинська обласна друкарня, 2008. – 470 с.

9. Fuechsle M. A single-atom transistor / M. Fuechsle, J. A. Miwa, S. Mahapatra, H. Ryu [et al.] // Nat Nanotechnol. Nature. – 2012. – No. 7 – P. 242 – 246.

10. Kumar S. Fundamental limits to Moore's law [Електронний ресурс] / S. Kumar // arXiv preprint arXiv:1511.05956. – 2015. – Режим доступу : https://www.researchgate.net/profile/Suhas_Kumar5/publication/284219009_Fundamental_Limits_to_Moore's_Law/links/5663fd9408ae192bbf901e85.pdf

11. Rajasekhar Y. Architecture and applications for an all-FPGA parallel computer / Y. Rajasekhar, R. Sass // Proceeding of 41st International Conference on Parallel Processing Workshops (ICPPW), (US, PA, Pittsburgh, 10 – 13 September 2012). – 2012. – P. 157 – 164.

12. Kogge P. Exascale computing study: Technology challenges in achieving exascale systems [Електронний ресурс] / P. Kogge, K. Bergman S. Borkar D. Campbell [et al.] // DARPA Information Processing Techniques Office (IPTO) sponsored study (Tech. Rep. TR-2008-13). – 2008. – Режим доступу : http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf.

13. Палагин А. В. Реконфигурируемые вычислительные системы: Основы и приложения / А. В. Палагин, В. Н. Опанасенко. – К. : Просвіта, 2006. – 280 с.

14. Баркалов А. А. Классификация реконфигурируемых вычислительных систем / А. А. Баркалов, Р. В. Мальчева, А. А. Гриценко // III Научно-практическая конференция «Донбасс 2020: Наука и Техника – Производству», (3 – 4 февраля 2006, Донецк, Украина). – Донецк : ДонНТУ Министерства образования и науки, 2006. – С. 473 – 478.

15. Дунець Р. Б. Дослідження часткової реконфігурації ПЛІС / Р. Б. Дунець, Д. Я. Тиханський // Радіоелектронні і комп’ютерні системи, 2009. – № 6 (40). – С. 240 - 244.

16. Melnyk V. Self-configurable fpga-based computer systems: basics and proof of concept / V. Melnyk // Advances in cyber-physical systems. – Vol. 1, No. 1. – 2016. – P. 39 – 50.

17. Koch D. Partial reconfiguration on FPGAs. Architectures, tools and applications / D. Koch. – N.Y. Springer-Verlag, 2013. – 296 p.

18. Каляев И. А. Семейство вычислительных систем с высокой реальной производительностью на основе ПЛИС / И. А. Каляев, И. И. Левин, Е. А. Семерников // Вестник УГАТУ. Энергетика, электрификация и энергетическое машиностроение. – Уфа : УГАТУ, 2010. – Т. 14, № 5 (40). – С. 91 – 101.

19. Baxter R. Maxwell – a 64 FPGA supercomputer / R. Baxter, S. Booth, M. Bull, A. Trew [et al.] // Adaptive Hardware and Systems, 2007 (AHS 2007), (Edinburgh, August 5 – 8, 2007). – IEEE Computer Society, 2007. – P. 287 – 294.

20. Клименко І. А. Класифікація реконфігурованих обчислювальних систем / І. А. Клименко, М. В. Рудницький // Вісник Вінницького політехнічного інституту. – Вінниця : ВНТУ, 2014. – № 5 (116). – С. 120 – 128.

21. Smith M. C. Optimization of shared high-performance reconfigurable computing resources / M. C. Smith, G. D. Peterson // ACM Transactions on Embedded Computing Systems. – Vol. 11, No. 2. – 2012. – P. 36.1 – 36.22.

22. George A. Novo-G: At the forefront of scalable reconfigurable supercomputing / A. George, H. Lam, and G. Stitt // 2011 Computing in Science & Engineering. – IEEE, 2011. – Vol. 13 (1). – P. 82 – 86.

23. Wolf W. Computers as components. Principles of embedded computing system design. – USA : Elsevier Inc., 2012. – 500 p.

24. Сергієнко А. М. Архітектура комп'ютерів: Конспект лекцій / А. М. Сергієнко. – К. : ІСЗЗІ НТУУ «КПІ», 2013. – 198 с.

25. Tessier R. Reconfigurable computing architectures / R. Tessier, K. Pocek, A. DeHon // Proceedings of the IEEE. – Vol. 103, No. 3. – 2015. – P. 332 – 354.

26. Putnam A. A reconfigurable fabric for accelerating large-scale datacenter services / A. Putnam, A. M. Caulfield E. S. Chung, D. Chiou [et al.] // Proceeding of the 41st annual international symposium on Computer architecuture (ISCA), (14 – 18 June 2014, Minneapolis, Minnesota, USA). – USA, NJ, IEEE Press Piscataway, 2014. – P. 13 – 24.

27. Клименко І. А. Класифікація та архітектурні особливості програмованих мультипроцесорних систем-на-кристалі / І. А. Клименко // Проблеми інформатизації та управління : зб. наук. праць. – К. : НАУ, 2012. – Вип. 1 (37). – С. 55 – 72.

28. Dorta T. Reconfigurable multiprocessor systems: A Review / T. Dorta, J. Jiménez, J. L. Martín, U.Bidarte, A.Astarloa // International Journal of Reconfigurable Computing: Special issue on selected papers from ReconFig

International conference on reconfigurable computing and FPGAs (ReconFig 20010) – 2010. – Vol. 2010. – 11 p.

29. Варшавский В. И. Однородные структуры: Анализ. Синтез. Поведение. – М. : Энергия, 1973. – 152 с.

31. Erstin G. Parallel processing in a reconfigurable computer / G. Erstin, R. Turn // IEEE Transaction on Electronic Computers. – EC-12, No. 6. – 1963. – P. 747 – 755.

30. Евреинов Э. В., Хорошевский В.Г. Однородные вычислительные системы. – Новосибирск : Наука, 1978. – 319 с.

32. Kasap S., Redif S. Novel field-programmable gate array architecture for computing the eigenvalue decomposition of para-hermitian polynomial matrices // IEEE transactions on very large scale integration (VLSI) systems. – IEEE, 2014. – Vol.22, No. 3. – P. 522 – 536.

33. Huang M. Reconfiguration and communication-aware task scheduling for high-performance reconfigurable computing / M. Huang, V.K. Narayana, H. Simmler, O. Serres, T. El-Ghazawi // Transactions on Reconfigurable Technology and Systems (TRETS). – US, NY, New York, ACM, 2010. – Vol. 3, No 4. – P. 20.1 – 20.25.

34. Клименко І. А. Тенденції застосування сучасної елементної бази для побудови високопродуктивних обчислювальних систем / І. А. Клименко // Проблеми інформатизації та управління : зб. наук. праць.– К. : НАУ, 2010. – Вип. 1 (29). – С. 90 – 96.

35. Partial reconfiguration in the ISE design suite [Електронний ресурс]. – Xilinx, 2014. – Режим доступу : http://www.xilinx.com/tools/partial-reconfiguration.htm.

36. Paiz Gatica C. V. Dynamically reconfigurable hardware for embedded control systems: Doctor of Science in Electronics Thesis : 21.12.2012 / C. V. Paiz Gatica. – Bielefeld, 2012. – 220 p.

37. Berthold O. Self-reconfiguring system-on-chip using Linux on a Virtex-5 FPGA: Master of Science Thesis : 26.04.2012 / O. Berthold; Universität zu Berlin. – Berlin, 2012. – 101 p.

38. Wildermann S. Systematic design of self-adaptive embedded systems with applications in image processing: Master of Science Thesis : 16.04.2012 / S. Wildermann. – Nurnberg, 2012. – 215 p.

39. Paya-Vaya G. Dynamic data-path self-reconfiguration of a VLIW-SIMD Soft-processor architecture / G. Paya-Vaya, R. Burg, H. Blume. // Workshop on Self-Awareness in Reconfigurable Computing Systems (SRCS), (1 September 2012, Oslo, Norway) – Oslo, 2012. – P. 26 – 29.

40. Azarian A. Reconfigurable computing architecture survey and introduction / A. Azarian, M. Ahmadi // Proceeding of 2nd IEEE International Conference Computer Science and Information Technology (ICCSIT 2009), (8 – 11 August, 2009, Beijing, China). – IEEE Computer Society, 2009. – P. 269 – 274.

41. Fons Lluís F. Embedded electronic systems driven by run-time reconfigurable hardware: Master of Science Thesis : 21.03.2012 / F. Fons Lluís. – Tarragona, 2012. – 242 p.

42. Abidine El Z. Self-partial and dynamic reconfiguration implementation for AES using FPGA / Z. El Abidine, A. Ismaili, A. Moussa // IJCSI International Journal of Computer Science Issues – 2009. – Vol. 2. – P. 33 - 40.

43. Danne K. Periodic real-time scheduling for FPGA computers / K. Danne, M. Platzner // Intelligent Solutions in Embedded Systems. – IEEE Computer Society, 2005. – P. 117 - 127.

44. Bassiri M. M. Mitigating reconfiguration overhead in on-line task scheduling for reconfigurable computing systems / M. M. Bassiri, S. H. Shahriar // Proceeding of 2nd International Conference on Computer Engineering and Technology (ICCET), (16 – 18 April 2010, Chengdu, China). – IEEE, 2010. – Vol. 4. – P. 397 – 402.

45. Wang X. Migration between software and hardware task on preemptive multitasking CPU/FPGA Hybrid Architecture / X. Wang, D. Feng, C. Tian-zhou, Hu Tong-sen // High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), (25 – 27 June 2012, Liverpool, England). – IEEE Computer Society, 2012. – P. 1329 – 1336.

46. Al-Wattar A. Efficient on-line hardware/software task scheduling for dynamic run-time reconfigurable systems / A. Al-Wattar, S. Areibi, F. Saffih // Proceeding of 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), (21-25 May 2012, Shanghai, China). – IEEE, 2012. – P. 401 – 406.

47. El-Araby E. Exploiting partial runtime reconfiguration for high-performance reconfigurable computing / E. El-Araby E., I. Gonzalez, T. El-Ghazawi // ACM Transactions on Reconfigurable Technology and Systems (TRETS). – Vol. 1 (4). – ACM New York, NY, USA, 2009. – P. 36.1 – 36.23.

48. Belletti F. Ianus: an adaptive FPGA computer / F. Belletti, I. Campos, A. Maiorano, S. P. Gavir [et al.] // Computing in Science & Engineering. – Vol. 8 (1). – 2006. – P. 41 – 49.

49. Палагин А. В. Реконфигурируемые структуры на базе FPGA: Синтез проблемно-ориентированных структур / А. В. Палагин, В. Н. Опанасенко, С. Л. Крывый // Verlag: LAP Lambert Academic Publishing, 2014. – 54 с.

50. Court T. V. Families of FPGA-based accelerators for approximate string matching / T. V Court., M. C. Herbordt // ACM Microprocessors and Microsystems – 2007. – Vol. 31, No. 2. – P. 135 – 145.

51. Жабин В. И. Архитектура вычислительных систем реального времени / В. И. Жабин. – К. : Век +, 2003. – 176 с.

52. Abirami B. Performance Analysis of OS Scheduling for a Reconfigurable Computing Environment / B. Abirami, K. Sridhar, V. Vaidhyanathan // Indian Journal of Science & Technology. – India, Indian Society of Education and Environment, 2015. – Vol. 8. – P. 22:1 – 22:5.

53. Platzner M. Dynamically reconfigurable systems: architectures, design methods and applications / M. Platzner, J. Teich, N. Wehn – Springer, 2010 – 300 p.

54. Ahmed W. Adaptive resource management for simultaneous multitasking in mixed-grained reconfigurable multicore processors / W. Ahmed, M. Shafique, L. Bauer, J. Henkel // Proceedings of the 9th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), (9 – 14 October 2011, Taipei, Taiwan). – IEEE, 2011. – P. 365 – 374.

55. Каляев И. А. Высокопроизводительные реконфигурируемые вычислительные системы на основе плис Virtex-6 и Virtex-7 / И. А. Каляев, А. И. Дордопуло, И. И. Левин, Е. А. Семерников // Параллельные вычислительные технологии (ПаВТ’2012), (26 – 30 марта 2012 г., Новосибирск, Россия). – Челябинск : ЮУрГУ, 2012. – С. 449 – 458.

56. Schmidt A. G. Investigation into scaling I/O bound streaming applications productively with an all-FPGA cluster / A. G. Schmidt, S. Datta, A. A. Mendon, R. Sass // Journal Parallel Computing. – Vol. 38 (8). – 2012. – P. 344 – 364.

57. So H. K.-H. Improving usability of fpga-based reconfigurable computers through operating system support / H. K.-H. So and R. W. Brodersen // Proceeding of International Conference on Publication Field Programmable Logic and Applications (FPL '06), (28 – 30 August 2006, Madrid, Spain). – 2006. – P. 349 – 354.

58. Fazlali M. Efficient task scheduling for runtime reconfigurable systems / M. Fazlali, M. Sabeghi, A. Zakerolhosseini, K. Bertels // Journal of Systems Architecture. – JSA, Euromicro journal, 2010. – Vol. 56 (11). – P. 623 – 632.

59. Schmidt A. G. Improving FPGA design and evaluation productivity with a hardware performance monitoring infrastructure / A. G. Schmidt, R. Sass // Proceeding of International Conference on Reconfigurable Computing and FPGAs (ReConFig 2011), (30 December – 2 November, 2011, Cancun, Mexico). – IEEE Computer Society, 2011. – P. 422 – 427.

60. Rajasekhar Y. Architecture and applications for an All-FPGA parallel computer / Y. Rajasekhar, R. Sass // Cluster Computing. – 2014. – Vol. 17, No. 2. – P. 315 – 325.

61. Schmidt A. G. An evaluation of an integrated on-chip/off-chip network for high-performance reconfigurable computing / A. G. Schmidt, W. V. Kritikos, S. Gao, R. Sass // International Journal of Reconfigurable Computing. – 2012. – Vol. 2012. – P. 564704.1 – 564770.15.

62. Schmidt A. G. Investigation into scaling i/o bound streaming applications productively with an all-FPGA cluster / A. G. Schmidt, S. Datta, A. A. Mendon, R. Sass // International Journal on Parallel Computing. – 2012. – Vol. 38 (8). – P. 344 – 364.

63. Chang C. BEE2: A highend reconfigurable computing system / C. Chang, J. Wawrzynek, R. Brodersen // IEEE Des. Test. Comput. – 2005. – Vol. 22, No. 2. – P. 114 – 125.

64. Starke C. Optimizing investment strategies with the reconfigurable hardware platform RIVYERA. Research article / C. Starke, V. Grossmann, L. Wienbrandt, S. Koschnicke, J. Carstens, M. Schimmler // International Journal of Reconfigurable Computing. – Hindawi Publishing Corporation, 2012. – Vol. 2012. – P. 646984.1 – 646984.10.

65. Jing C. Ant-colony optimization based algorithm for energy-efficient Scheduling on Dynamically Reconfigurable Systems / C. Jing // Proceeding of Ninth International Conference on Frontier of Computer Science and Technology. – 2015. – P. 127 – 134.

66. Redif S. Novel reconfigurable hardware architecture for polynomial matrix multiplications / S Redif, S Kasap // IEEE Transactions on Very Large Scale Integration (VLSI) Systems. – 2014. – Vol. 23 (3). – P. 454 – 465.

67. Kasap S. Parallel processor design and implementation for molecular dynamics simulations on a FPGA-based supercomputer / S. Kasap, K. Benkrid // Journal of Computers. – 2012. – Vol. 7, No. 6. – P.1312 – 1328.

68. Дунець Р. Б. Проблеми побудови частково реконфігурованих систем на ПЛІС / Р. Б. Дунець, Д. Я. Тиханський // Радіоелектронні і комп'ютерні системи, 2010. – № 7 (48). – С. 200 – 204.

69. Guan N. Schedulability analysis of preemptive and nonpreemptive EDF on partial runtime-reconfigurable FPGAs / N. Guan, Q. Deng, Z. Gu, W. Xu [et al.] // ACM Transactions on Design Automation of Electronic Systems. – New York : ACM, 2008 – Vol. 13, No. 4. – P. 56:1 – 56:43.

70. Тиханський Д. Я. Метод підвищення ефективності частково реконфігурованої системи / Д. Я. Тиханський, Р. Б. Дунець, Р. В. Грица // Комп'ютерні системи та мережі: зб. наук. праць. – Львів : Львівська політехніка, 2011. – С. 192 – 198.

71. R. Salvador Evolvable Hardware in FPGAs: Embedded tutorial / R. Salvador // Proceeding of 11-th International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS), (12 – 14 April 2016, Istanbul, Turkey) – 2016. – P. 13:1 – 13:6.

72. Lie W. Dynamic partial reconfiguration in FPGAs / W. Lie, Wu Feng-Yan. // Third International Symposium on Intelligent Information Technology Application (IITA 09), (21 – 22 November 2009, NanChang, China). – IEEE Computer Society Washington, 2009. – Vol. 2. – P. 445 – 448.

73. Hussain H. M. Novel Dynamic Partial Reconfiguration Implementation of K-Means Clustering on FPGAs: Comparative Results with GPPs and GPUs / H. M. Hussain, K. Benkrid, A. Ebrahim, A. T. Erdogan [et al.] // International Journal of Reconfigurable Computing. – Hindawi Publishing Corporation, 2012. – Vol. 2012. – P. 135926:1 – 135926:15.

74. Saldana M. Using Partial Reconfiguration and Message Passing to Enable FPGA-Based Generic Computing Platforms / M. Saldana, A. Patel, H. J. Liu, P. Chow // International Journal of Reconfigurable Computing. – Hindawi Publishing Corporation, 2012. – Vol. 2012. – 2012. – P. 3:1 – 3:10.

75. Berthold O. Self-reconfiguring System-on-Chip using Linux on a Virtex-5 FPGA: Master of Science Thesis : 26.04.2012 / O. Berthold. – 101 p.

76. Paya-Vaya G. Dynamic Data-Path Self-Reconfiguration of a VLIW-SIMD Soft-Processor Architecture / G. Paya-Vaya, R. Burg, H. Blume // Workshop on Self-Awareness in Reconfigurable Computing Systems (SRCS) – Oslo, 2012. – P. 26 – 29.

77. Sverre H. Framework for self reconfigurable system on a Xilinx FPGA: Master of Science in Electronics Thesis : 11.06.09 / H. Sverre. – Trondheim, 2009. – 59 p.

78. Gonzalez I. Dynamically Reconfigurable Coprocessors in FPGA-based Embedded Systems : Doctor of Science in Electronics Thesis : 21.03.2006 / I. Gonzalez. – Madrid, 2006. – 56 p.

79. Schleupen K. Dynamic Partial FPGA Reconfiguration in a Prototype Microprocessor System / K. Schleupen, S. Lelaich, R. Mannion, G. Zhi [et al.] // Field Programmable Logic and Applications (FPL 2007), (27 – 29 August 2007, Amsterdam, Netherlands). – IEEE Computer Society, 2007. – P. 533 – 536.

80. What's New in Xilinx ISE Design Suite 12 [Електронний ресурс]. – Xilinx, 2014. – Режим доступу : http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/whatsnew.htm.

81. Yanase R. Formal verification of dynamically reconfigurable systems / R. Yanase; T. Sakai, M. Sakai, S. Yamane // Proceeding of 4th Global Conference on Consumer Electronics (GCCE). – IEEE, 2015. – P. 71 – 75.

82. Stratix V Partial Reconfiguration [Електронний ресурс]. – Altera wiki, 2014. – Режим доступу : http://www.alterawiki.com/wiki/Stratix_V_Partial_Reconfiguration.

83. About Partial Reconfiguration [Електронний ресурс]. – Altera Quartus II Help, 2014. – Режим доступу : http://quartushelp.altera.com/13.0/master.htm#mergedProjects/comp/comp/comp_about_part_reconfig.htm?GSA_pos=1&WT.oss_r=1&WT.oss=partial reconfiguration

84. Birla M. Partial run-time reconfiguration of FPGA for computer vision applications / M. Birla, K. N. Vikram. // Parallel and Distributed Processing, 2008

(IPDPS 2008), (Miami, April 14 - 18, 2008). – IEEE Computer Society, 2008. – P. 1 – 6.

85. Ming L. Run-time Partial Reconfiguration speed investigation and architectural design space exploration / L. Ming, W. Kuehn, L. Zhonghai, A. Jantsch. // Field Programmable Logic and Applications, 2009. (FPL 2009), (31 August –2 September 2009, Prague, Czech Republic). – IEEE Computer Society, 2009. – P. 498 – 502.

86. Kwok-Hay So H. Improving usability of fpga-based reconfigurable computers through operating system support / H. Kwok-Hay So, R.W. Brodersen // Proceeding of International Conference Field Programmable Logic and Applications (FPL '06), (28 – 30 August 2006 Madrid, Spain). – 2006. – P. 1 – 6.

87. Sridharan R. FPGA-based Reconfigurable Computing for Pricing Multi-asset Barrier Options / R. Sridharan, G. Cooke, K. Hill, H. Lam, A. George // Proceedings of Symposium on Application Accelerators in High Performance Computing (SAAHPC), (10-11 July 2012, Chicago, IL, US) – IEEE, 2012. – P. 34 – 43.

88. Kalte H. Context Saving and Restoring for Multitasking in Reconfigurable Systems / H. Kalte, M. Porrmann // Proceeding of International Conference on Field Programmable Logic and Applications, (24 – 26 August 2005 Tampere, Finland). – IEEE, 2005. – P. 223 – 228.

89. Liu S. Achieving Energy Efficiency through Runtime Partial Reconfiguration on Reconfigurable Systems / S. Liu, R.N. Pittman, A. Forin, J.- L. Gaudiot // Transactions on Embedded Computing Systems (TECS). – US, NY, New York, ACM, 2013. – Vol. 12, No. 3. – P. 72.1 – 72.21.

90. Клименко І. А. Аналіз ефективності управління ресурсами та додатками в реконфігурованих обчислювальних системах / І. А. Клименко // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка : зб. наук. праць. – К. : Век+, 2015. – № 62. – С. 11 – 21.

91. Taher M. Virtual Configuration Management: A Technique for Partial Runtime Reconfiguration / M. Taher, T. El-Ghazawi // IEEE Transactions on Computers. – IEEE, 2009. – Vol. 58, No. 10. – P. 1398 – 1410.

92. Koch D. Fine-Grained Partial Runtime Reconfiguration on Virtex-5 FPGAs / D. Koch, C. Beckhoff, J. Torrison // Proceeding of 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM) (2 – 4 May 2010, Charlotte, NC, USA). – IEEE, 2010. – P. 69 – 72.

93. Liu S. Minimizing the runtime partial reconfiguration overheads in reconfigurable systems / S. Liu, R.N. Pittman, A. Forin, J.-L. Gaudiot // The Journal of Supercomputing, 2012. – Vol. 61, No. 3. – P. 894 – 911.

94. Jara-Berrocal A. VAPRES: A Virtual Architecture for Partially Reconfigurable Embedded Systems / A. Jara-Berrocal, A. Gordon-Ross // Proceeding of Design, Automation & Test in Europe Conference & Exhibition (DATE), (8 – 12 March 2010, Dresden, Germany) – IEEE, 2010. – P. 837 – 842.

95. Zhao G. Research on reconfiguration technology based on SOPC for PXI instrument / G. Zhao, Y. Hou, S. Wang // Proceeding of IEEE AUTOTESTCON 2015, (USA, Virginia, November 2 – 5, 2015) – 2015. – P. 280 – 283.

96. Wu A. A flexible FPGA-to-FPGA interconnect interface design and implementation / A. Wu, X. Jin, S. Z. Guo, X. L. Du, // Proceeding of International Conference Computers Communications and Systems (ICCCS), (6 November 2015, Gyongsan, Korea). – 2015. – P. 52 – 56.

97. Jacoby A. Proteus: An open source dynamically reconfigurable system-on-chip with applications to digital signal processing / A. Jacoby, D. Llamocca, R. Jordan, G. A. Vera, // Proceeding of International Caribbean Conference of Devices Circuits and Systems (ICCDCS), (2 – 4 April 2014, Mexico City, Mexico). – 2014. – P. 1 – 6.

98. Reorda M. S. An Error-Detection and Self-Repairing Method for Dynamically and Partially Reconfigurable Systems / M. S. Reorda; L. Sterpone; A. Ullah //IEEE Transactions on Computers. – 2016. – Vol. PP. – P. 99.1 – 99.4.

99. Massively Parallel Dynamically Reconfigurable Multi-FPGA Computing System / V. Viswanathan; R. B. Atitallah; J.-L. Dekeyser // Proceeding of 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, (2 – 6 May 2015 Vancouver, Canada). – IEEE, 2015. – P. 165 – 167.

100. Buell D. High-Performance Reconfigurable Computing // D. Buell, T. El-Ghazawi, K. Gaj, V. Kindratenko // Journal Computer. – US, CA, IEEE Computer Society Press Los Alamitos, 2007. – Vol. 40 (3) – P. 23 – 27.

101. Panella A. A Design Workflow for Dynamically Reconfigurable Multi-FPGA Systems / A. Panella, F. Redaelli, F. Cancare, D. Sciuto // Proceeding of 18th IEEE/IFIP VLSI System on Chip Conference (VLSI-SoC), (27 – 29 Sept. 2010, Madrid, Spain). – IEEE, 2010. – P. 114 – 119.

102. Liu S. Achieving Energy Efficiency through Runtime Partial Reconfiguration on Reconfigurable Systems / S. Liu, R. N. Pittman, A. Forin, J.- L. Gaudiot // Transactions on Embedded Computing Systems (TECS). – USA, NY, New York, ACM, 2013. – Vol. 12 (3). – P. 72.1 – 72.21.

103. Wang X. Migration between software and hardware task on preemptive multitasking CPU/FPGA hybrid architecture / X. Wang, D. Feng, C. Tian-zhou, H. Tong // Proceeding of 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), (25 – 27 June 2012, Liverpool, United Kingdom). – IEEE, 2012. – P. 1329 – 1336.

104. Levchenko R. I. A System of Automatic Dynamic Paralleling of Computations for Multiprocessor Computer Systems with Weak Connection (DDCI) / R. I. Levchenko, O. O. Sudakov, S. D. Pogorelij, Y. V. Bojko // USiM, 2008. – No. 3. – P. 66 –72.

105. Левченко Р. И. Проблемы эффективности автоматического динамического распараллеливания вычислений для многопроцессорных компьютерных систем со слабой связью / Р. И. Левченко, А. А. Судаков, С. Д. Погорелый, Ю. В. Бойко // Проблеми програмування. Спеціальний випуск – № 2 – 3. – К. : ИПС НАНУ, 2010. – С. 178 – 184

106. Луцкий. Г. М., Стиренко С. Г., Зиненко А. И. Представление задач в системах распараллеливания с изменяемой зернистостью. // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка. – К. : 2012. – Вип. 55. – С. 30 – 37.

107. Ahmed W. mRTS:Run-Time System for Reconfigurable Processors with Multi-Grained Instruction-Set Extensions / W. Ahmed, M. Shafique, L. Bauer, J. Henkel // Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), (14 – 18 March 2011, Grenoble, France). – IEEE, 2011. – P. 1 – 6.

108. Koenig R. KAHRISMA: A Novel Hypermorphic Reconfigurable-Instruction-Set Multi-grained-Array Architecture / R. Koenig, L. Bauer, T. Stripf, M. Shafique, W. Ahmed, J. Becker, J. Henkel // Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), (8 – 12 March 2010 Dresden, Germany). – IEEE, 2010. – P. 819 – 824.

109. Dümmler J. Scalable computing with parallel tasks / J. Dümmler, T. Rauber, G. Rünger // Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS '09) (14 – 20 November 2009, Portland, Oregon, US). – ACM New York, NY, USA, 2009. – № 9 – P. 1 – 10.

110. Дронов В. В. Проблемы имитационного моделирования в режиме реального времени / В. В. Дронов // Вестник НовГУ. Серия: Естеств. и техн. науки. – № 19. – 2001. – С. 99 – 102.

111. Минаев Ю. Н. Нечеткая математика на основе тензорных моделей неопределенности. Часть 1. Тензор-переменная в системе нечетких множеств / Ю. Н. Минаев, О. Ю. Филимонова // Электронное моделирование. – К. : ИПМЭ НАН Украины, 2008. – № 1, Т. 30 – С. 43 – 59.

112. Минаев Ю. Н. Нечеткая математика на основе тензорных моделей неопределенности. Часть 2 – нечеткая математика в тензорном базисе / Ю. Н. Минаев, О.Ю. Филимонова // Электронное моделирование. – К. : ИПМЭ НАН Украины, 2008. – № 2, Т. 30 – С. 4 – 21.

113. Opanasenko V. Method synthesis of the configurable logical blocks on basis of universal logical elements / V. Opanasenko, S. Kryvyi // Радіоелектронні і комп'ютерні системи: наук.-техн. журн. – Харків : ХАІ, 2016. – № 5, Вип. 79. – С. 93–97.

114. Опанасенко В. Прямая задача синтеза адаптивных логических сетей / В. Опанасенко, С. Крывый // International Journal «Information Technologies & Knowledge». – Bulgaria, Sofia, Publisher ITHEA, 2014 – Vol. 8, No 1. – P. 3– 12.

115. Додонов А. Г. Организация структуры системы обработки информации и управления / А. Г. Додонов, В. Г. Путятин, А. Н. Буточнов, Н. С. Козлов, В. В. Юзефович // Математичні машини і системи. – 2014. – № 4. – С. 18–34.

116. Сергієнко А. М. Реконфігурована багатопроцесорна обчислювальна система на ПЛІС / А. М. Сергієнко, І. А. Клименко, П. А. Сергієнко // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка : зб. наук. пр. – К. : Век+, 2016. – № 64. – С. 47 – 50.

117. Мартынова О. П. Оценка эффективности системоаналоговых вычислительных структур / О. П. Мартынова, Г. В. Данилина, И. А. Клименко // Проблеми інформатизації та управління : зб. наук.праць. – К. : НАУ, 2006. – Вип. 1 (16). – С. 106 – 109.

118. Клименко І. А. Особливості реалізації паралельних обчислювальних систем на програмованій елементній базі / І. А. Клименко // Матеріали 2-ї Міжнар. конф. «Високопродуктивні обчислення (HPC_UA'2012)», (8 – 10 жовтня, 2012 р., Київ, Україна). – К. : НАН України, 2012. – С. 210–214.

119. Минаев Ю. Н. Мягкие вычисления на основании моделей кронекеровой (тензорной) алгебры / Ю. Н. Минаев, І. А. Клименко // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка : зб. наук. праць. – К. : ВЄК +, 2011. – № 54. – С. 18–25.

120. Zade L. A. Fuzzy logic, neural networks and soft computing / L. A. Zade // Communications of the ACM. – 1994. – Vol. 37, № 3. – P. 77 – 84.

121. Заде Л. А. Понятие лингвистической переменной и его применение к принятию приближенных решений / Л. А. Заде. – М. : Мир, 1976. – 165 с.

122. Поспелов Д. А. Нечеткие множества в моделях управления и искусственного интеллекта / Д. А. Поспелов. – М. : Наука, 1986. – 312 с.

123. Brannon R. M. Elementary and Intermediate Vector and Tensor Analysis presented in a framework generalizable to higher-order applications in material modeling // UNM Report, University of New Mexico, Albuquerque, 2002 . – 222 p.

124. Hay G. E. Vector and Tensor Analysis / G. E. Hay. – Courier Corporation, 2012 – 193 p.

125. Springer C. E. Tensor and Vector Analysis: With Applications to Differential Geometry / C. E. Springer. – Courier Corporation, 2012. – 242 p.

126. Тыртышников Е. Е. Матричные методы и технологии для задач со сверхбольшим числом неизвестных / Е. Е. Тыртышников. – М. : ИВМ РАН, 2005. – 23 с.

127. Крон Г. Тензорный анализ сетей / Г. Крон. – М. : Сов. радио, 1978. – 720 с.

128. Мінаєв Ю. М. Розв’язок нечітких систем лінійних алгебричних рівнянь / Ю. М. Мінаєв, О. Ю. Філімонова, Ю. І. Мінаєва, Є. О. Гончарова // Наукоємні технології: Науковий журнал. – К. : Видавництво НАУ, 2009. – С. 45 – 67.

129. Graham A. Kronecker product and matrix calculus with applications / A. Graham. – USA, NY : Wiley, 1981. – 130 p.

130. Голуб Д. Матричные вычисления: Пер. з англ. / Д. Голуб, Ч. Ван Лоун – М. : Мир, 1999. – 548 с.

131. Жуков І. А. Обчислювальна система для розв’язку нечітких СЛАР / І. А. Жуков, І. А. Клименко // Проблеми інформатизації та управління : зб. наук. праць. – К. : НАУ, 2011. – Вип. 3 (35). – С. 34 – 43.

132. Клименко І. А. Дослідження ефективності апаратних прискорювачів на ПЛІС для рішення задач великої розмірності / І. А. Клименко, О. С. Головко, М. О. Гілляка, Я. І. Мицьо // Тези доповідей IX Міжнар. наук.-техніч. конференції «Комп'ютерні системи і мережні технології (CSNT 2016)», (21 – 23 квітня, 2016 р., Київ, Україна). – К. : НАУ, 2016. – С. 43 – 44.

133. Жабин В. И., Макаров В. В. Таблично-алгоритмический метод вычисления многочленов // Вісник НТУУ «КПІ», «Інформатика, управління та обчислювальна техніка». – № 46. – 2007. – С. 206 – 210.

134. Жабин В. И. Построение быстродействующих специализированных вычислителей для реализации многоместных выражений / В. И. Жабин, В. И. Корнейчук, В. П. Тарасенко // Автоматика и вычислительная техника. – 1981. – № 6. – С. 18-22.

135. Клименко І. А. Моделювання відмовостійкої потокової обчислювальної системи на ПЛІС / І. А. Клименко, В. В. Жабіна, В. В. Зволинський // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка : зб. наук. праць. – К. : ВЄК +, 2011. – № 54. – С. 175 – 180.

136. Клименко І. А. Співпроцесор для виконання арифметичних операцій з плаваючою комою на ПЛІС / І. А. Клименко, В. В. Ткаченко, Є. В. Красовська // Наукові праці Донецького національного технічного університету «Інформатика, кібернетика та обчислювальна техніка». – Донецьк : ДВНЗ, 2012. – № 16 (204). – С. 58 – 67.

137. Клименко І. А. Спосіб управління зернистістю задач в реконфігурованих обчислювальних системах / І.А Клименко // Тези доп. Всеукраїнської наук.-практич. конф. «Перспективні напрямки сучасної електроніки, інформаційних та комп'ютерних систем (MEICS-2015)», (25 – 27 листопада 2015 р., Дніпропетровськ, Україна). – Дн. : ДНУ ім. О. Гончара, 2015. – С. 117 – 118.

138. Жабин В. И. Исследование методов построения вычислительных устройств на основе FPGA фирмы XILINX / В. И. Жабин, Н. А. Ковалев // Технология и конструирование в электронной аппаратуре. – 2002. – № 1. – С. 35 – 39.

139. Клименко І. А. Мультипроцесорна обчислювальна система на ПЛІС / І. А. Клименко // Тези доп. Міжнар. наук.-практич. конф. «Інформаційні технології в освіті, науці й техніці, (ІТОНТ-2012)», (25 – 27 квітня 2012 р., Черкаси, Україна). – Черкаси : ЧДТУ, 2012. – Т. 1. – С. 114.

140. Клименко І. А. Співпроцесор для розв'язання систем лінійних алгебраїчних рівнянь на ПЛІС / І. А. Клименко // Матеріали 3-ї Міжнар. конф. «Високопродуктивні обчислення (HPC_UA'2013)», (7 – 11 жовтня, 2013 р., Київ, Україна). – К. : НАН України, 2013. – С. 386 – 389.

141. Жабин В. И., Макаров В. В. Использование таблиц функций для быстрого вычисления многочленов // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка. – № 49. – 2008. – С. 109-112.

142. Самофалов К. Г. Основы теории многоуровневых конвейерных вычислительных систем / К. Г. Самофалов, Г. М. Луцкий. – М. : Радио и связь, 1989. – 272 с.

143. Информационные системы: Табличная обработка информации / Под ред. Е. П. Балашова и В. Б. Смолова. – Л. : Энергоатомиздат, 1985. – 184 с.

144. Кнут Д. Искусство программирования для ЭВМ: В 3-х т., т. 2. – М. : Мир, 1977. – 723 с.

145. Жабин В. И. Некоторые машинные методы вычисления рациональных функций многих аргументов / В. И. Жабин, В. И. Корнейчук, В. П. Тарасенко // Автоматика и телемеханика. – 1977. – № 12. – С. 145-154.

146. Клименко І. А. Мультипроцесорна система на базі програмованих процесорних ядер Nios II Altera / І. А. Клименко, В. В. Ткаченко,

О. М. Сторожук // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка : зб. наук. праць. – К. : Век+, 2012. – № 56. – С. 78 – 87.

147. Жабин В. И. Реализация вычислений под управлением потока дескрипторов данных в мультипроцессорных системах / В. И. Жабин // Электронное моделирование. – К. : ИПМЕ, 2003. – Т. 25, № 1. – С. 35 – 47.

148. Жабин В. И. Повышение эффективности параллельных вычислений в потоковых системах [Електронний ресурс] / В. И. Жабин, В. В. Жабина // Вісник Національного технічного університету України «КПІ». Інформатика, управління та обчислювальна техніка. – 2013. - Вип. 59. – С. 122-128.

149. Dennis J. B., Missunas D. P. A preliminary architecture for basic data flow processor // Proc. 2nd Annual Symp. Comput. Stockholm, May 1975. N. Y. IEEE. – 1975. – P. 126 – 132.

150. Johnson D. Data flow machines threaten the program counter // Electronic Design. – 1980. – No. 22. – P. 255 – 258.

151. Silva J. G. D., Wood J. V. Design of processing subsystems for Manchester data flow computer // IEEE Proc. N.Y. – 1981. – Vol. 128, N 5. – P. 218 – 224.

152. Watson R., Guard J. A practical data flow computer // Computer. – 1982. – Vol. 15, No. 2. – P. 51 – 57.

153. Жабін В. І. Методи і засоби підвищення ефективності паралельних обчислювальних систем реального часу : автореф. дис. на здобуття наук. ступеня докт. техн. наук : спец. 05.13.13 «Обчислювальні машини, системи та мережі» / В. І. Жабін. – К. : 2006. – 36 с.

154. Дронов В. В. Распараллеливание вычислительного алгоритма методом приоритетного спуска / В. В. Дронов, Ю. В. Иванов // Вестник НовГУ. Серия: Естеств. и техн. науки. – № 22. – 2001. – С. 46 – 49.

155. Кулаков Ю. О. Метод оптимізації ярусно-паралельної форми подання задачі для реконфігурованих обчислювальних систем / Ю. О.

Кулаков, І. А. Клименко // Електроніка та зв'язок. – К. : НТТУ «КПІ», 2014. – Т. 19, № 4 (81). – С. 90 – 96.

156. Поспелов Д. А. Введение в теорию вычислительных систем / Д. А. Поспелов. – М. : Советское радио, 1972. – 280 с.

157. Луцький Г. М. Зменшення накладних видатків реконфігурації в реконфігурованих обчислювальних системах / Г. М. Луцький, І. А. Клименко // Штучний інтелект. – К. : Наука і освіта, 2015. – № 1 – 2 (67 – 68). – С. 146 – 156.

158. Кулаков Ю. О. Розробка методу прискорення реконфігурації в динамічно реконфігурованих обчислювальних системах / Ю. О. Кулаков, І. А. Клименко, М. В. Рудницький // Східно-Європейський журнал передових технологій. – Харків : УДА залізничного транспорту, 2015. – № 4/4 (76). – С. 25 – 30.

159. Кулаков Ю. О. Організація багаторівневої пам'яті в реконфігурованих обчислювальних системах / Ю. О. Кулаков, І. А. Клименко // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка : зб. наук. праць. – К. : ВЄК+, 2014. – № 61. – С. 18 – 26.

160. Кулаков Ю. О. Формалізація методу повторного використання апаратних ресурсів функціональних блоків реконфігурованих обчислювальних систем / Ю. О. Кулаков, І. А. Клименко // Моделювання та інформаційні технології. – К. : ІПМЕ ім. Г. Є. Пухова НАН України, 2015. – Вип. 75. – С. 60 – 68.

161. Клименко І.А. Формалізація адаптивного відображення задач у реконфігуровних обчислювальних системах на ПЛІС / І. А. Клименко // Штучний інтелект. – К. : Наука і освіта, 2016. – № 1 (71). – С. 57 – 67.

162. Рудницький М. В. Метод адаптивного планування в динамічно реконфігурованих обчислювальних системах / М. В. Рудницький, І. А. Клименко // Тези доповідей наук. конф. студентів, магістрантів та аспірантів «Інформатика та обчислювальна техніка, ІОТ-2016», (25 – 27 квітня 2016 р., Київ, Україна). – К. :. НТУУ «КПІ», 2016. – С. 107 – 112.

163. Клименко І. А. Засоби адаптивного відображення задач на реконфігуровну обчислювальну структуру в паралельних обчислювальних системах, що керуються потоком даних / І. А. Клименко, В. В. Ткаченко, О. М. Сторожук // Електроніка та зв'язок. – К. : НТТУ «КПІ», 2016. – Том 21, № 2 (91). – С. 71 – 77.

164. Клименко І. А. Алгоритм завчасної реконфігурації для реконфігуровних обчислювальних систем на ПЛІС / І. А. Клименко, О. В. Опанасюк // Проблеми інформатизації та управління : зб. наук. праць.– К. : НАУ, 2016. – Вип. 2 (54). – С. 29 – 35.

165. Klymenko I. The method for providing quality of service time requirements in reconfigurable computing systems / I. Klymenko, Y. Kulakov, V. Tkachenko, O. Storozhuk // Східно-Європейський журнал передових технологій. – Харків : УДА залізничного транспорту, 2016. – № 5 / 9 (83). – С. 4 – 12.

166. Клименко І. А. Оптимізація реконфігурації в динамічно реконфігурованих обчислювальних системах / І. А. Клименко // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка : зб. наук. праць. – К. : Век+, 2015. – № 63. – С. 93 – 100.

167. Klymenko I. A. The development of means of definition of the optimum ratio of computational algorithm and the reconfigurable structure / I. A. Klymenko, O. S. Holovko, M. O. Hilliaka, Y. I. Mytsio // Східно-Європейський журнал передових технологій. – Харків : УДА залізничного транспорту, 2016. – № 3 / 2 (81). – С. 4 – 8.

168. Додонов А. Г. Комп'ютерне моделювання систем організаційного управління / А. Г. Додонов // Математичні машини і системи. – 2016. – № 3. – С. 85–92.

169. Колмогоров А.Н. Теория вероятностей и математическая статистика / А. Н. Колмогоров. – 1986. – 400 с.

170. Стіренко С. Г. Організація паралельних обчислювальних процесів в кластерних системах. – К. : Век+, 2014. – 196 с.

171. Клименко І. А. Модифікований спосіб забезпечення часових вимог додатків в динамічно реконфігурованих обчислювальних системах / І. А. Клименко // Матеріали І Міжнар. конф. «Infocom Advanced Solution 2015», (24 – 25 листопада 2015 р., Київ, Україна). – К. : НТУУ «КПІ», 2015. – С. 36 – 37.

172. Клименко І. А. Метод динамічного аналізу графічної інформації для вводу даних в ПЕОМ / І. А. Клименко, Я. М. Паламар // Проблеми інформатизації та управління : зб. наук. праць. – К. : НАУ, 2010. – Вип. 2 (30). – С. 84 – 89.

173. Клименко І. А. Пристрій для автоматичного вводу графічної інформації в ПЕОМ / І. А. Клименко, Я. М. Паламар // Тези доп. 13-ї Всеукраїнської (8-ї Міжнародної) студентської наук. конф. з прикладної математики та інформатики (СНКПМІ-2010), (22 – 23 квітня 2010 р., Львів, Україна). – Львів : ЛНУ, 2010. – С. 179 – 181.

174. Клименко И. А. Метод отображения задач на реконфигурируемую архитектуру вычислительной системы / И. А. Клименко // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка : зб. наук. праць. – К. : Век+, 2013. – № 59. – С. 43 – 49.

175. Колесник И. Н. Анализ применения технологии FPGA в составе облачной инфраструктуры / И. Н. Колесник, В. А. Куланов, А. Е. Перепелицын // Радіоелектронні і комп’ютерні системи. – № 6 (80) – 2016. – С. 130 – 135.

176. Byma S. FPGAs in the cloud: Booting virtualized hardware accelerators with OpenStack / S. Byma, J. G. Steffan, H. Bannazadeh, A. Leon-Garcia [et al.] // Proceeding of 22nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), (11 - 13 May 2014, Boston, Massachusetts, USA). – IEEE, 2014. – P. 109 – 116.

177. Кулаков Ю. А. Особенности реализации динамической GRID среды / Ю. О. Кулаков, И. А. Клименко // Проблеми інформатизації та управління : зб. наук. праць. – К. : НАУ, 2009. – Вип. 3 (27). – С. 87 – 93.

178. Кулаков Ю. А. Способ формирования структуры виртуальной GRID системы / Ю. О. Кулаков, И. А. Клименко // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка : зб. наук. праць. – К. : ВЄК +, 2009. – № 50. – С. 97 – 100.

179. Пат. на вин. 74712 Україна, МПК G06F 15/16 (2006.01). Багатопроцесорна система / І. А. Жуков, В. І. Жабін, І. А. Клименко, Р. Л. Антонов. – № 20040403215; заявл. 28.04.2004; опубл. 16.01.2006, Бюл. № 1. – 18 с.

180. Пат. на кор. мод. 10444 Україна, МПК G06F 15/16 (2006.01). Обчислювальна система / І. А. Жуков, В. І. Жабін, І. А. Клименко, В. В. Ткаченко. – № u 2005 04117; заявл. 29.04.2005; опубл. 15.11.2005, Бюл. № 11. – 8 с.

181. Пат. на кор. мод. 25009 Україна, МПК G06F 15/16 (2006). Обчислювальна система / В. І. Жабін, І. А. Жуков, І. А. Клименко, В. В. Ткаченко. – № u 2007 02005; заявл. 26.02.2007; опубл. 25.07.2007, Бюл. № 11. – 8 с.

182. Пат. на кор. мод. 60398 Україна, МПК G06F 15/16 (2006.01). Багатопроцесорна обчислювальна система / І. А. Жуков, І. А. Клименко, С. М. Біляєв. – № u 2010 07467; заявл. 15.06.2010; опубл. 25.06.2011, Бюл. № 12. – 16 с.

183. Клименко І. А. Обчислювальна система на ПЛІС / І. А. Клименко, С. В. Біляєв // Тези доп. ІІІ Міжнар. наук.-техніч. конф. «Комп’ютерні системи та мережні технології», (15 – 17 червня 2010 р., Київ, Україна). — К. : НАУ, 2010. – С. 45.

184. Клименко І. А. Процесорне ядро для обчислювальної системи на ПЛІС / І. А. Клименко, С. М. Біляєв, Д. С. Пономарчук // Тези доп. Другої наук. конф. «Прикладна математика та комп’ютинг ПМК 2010», (12–16 квітня 2010 р., Київ, Україна). – К. : Просвіта, 2010. – С. 197 – 201.

185. Жуков І. А. Модуль оперативної пам’яті для RISC на ПЛІС / І. А. Жуков, І. А. Клименко // Проблеми інформатизації та управління : зб. наук.праць. – К. : НАУ, 2009. – Вип. 4 (28). – С. 50 – 54.

186. Клименко І. А. Реалізація схеми управління станами та зсувами на ПЛІС / І. А. Клименко, С. В. Біляєв // Тези доп. Міжнар. наук.-техніч. конф. «Інтелектуальні технології лінгвістичного аналізу», (21 – 22 жовтня 2009 р., Київ, Україна). – К. : НАУ, 2009. – С. 21.

187. Клименко І. А. Спосіб автоматичного розпаралелювання задач із завчасною реконфігурацією на ПЛІС / І. А. Клименко, О. М. Сторожук // Тези доп. VII Міжнар. наук. конф. «Сучасні проблеми математичного моделювання, прогнозування та оптимізації (ОПТІМА 2016)», (21 – 22 квітня, 2016 р., Кам’янець-Подільський, Україна). – Кам’янець-Подільський : Кам’янець-Подільський національний університет імені Івана Огієнка, 2016. – С. 93 – 94.

188. Клименко І. А. Апаратні засоби прискорення реконфігурації в реконфігурованих обчислювальних системах / І. А. Клименко // Тези доп. XXII Міжнар. конф. з автоматичного управління «Автоматика – 2015», (10 – 11 вересня 2015 р., Одеса, Україна). – Одеса : ТЕС, 2015. – С. 104 – 105.

189. Жабін В. І., Жуков І. А., Ткаченко В. В., Клименко І. А. Мікропроцесорні системи : навч. посібник. – К. : Вид-во «СПД Гуральник О. Ю.», 2009. – с. 423-439.

190. Klymenko I. Multiprocessor system on chip based on programmable processor cores Nios II Altera / I. Klymenko, A. Storozhuk // Матеріали VI Міжнародної конференції молодих вчених «Комп'ютерні науки та інженерія (CSE-2013)», (21 – 23 листопада 2013 р., Львів, Україна). – Львів : Видавництво Львівської політехніки, 2013. – С. 32 – 33.

191. Клименко І. А. Вдосконалення засобів синхронізації процесів в мультипроцесорних системах-на-кристалі / І. А. Клименко // Електроніка та зв’язок. – К. : НТТУ «КПІ», 2015. – Т. 20, № 3. – С. 107 – 113.

192. Пат. на кор. мод. 59112 Україна, МПК G06F 15/16 (2011.01). Обчислювальний пристрій / І. А. Клименко, В. В. Жабіна. – № u 2010 09793; заявл. 06.08.2010; опубл. 10.05.2011; Бюл. № 9. – 18 с.

193. Пат. на кор. мод. 7727 Україна, МПК G06F 15/16 (2006.01). Обчислювальний пристрій / І. А. Жуков, В. І. Жабін, І. А. Клименко, В. В. Ткаченко. – №20040907712; заявл. 22.09.2004; опубл. 15.07.2005, Бюл. № 7. – 12 с.

194. Клименко І. А. Підвищення відмовостійкості систем, що управляються потоком даних / І. А. Клименко, В. В. Ткаченко // Тези доп. Ювілейної міжнар. наук.-практич. конф. «Розподілені комп'ютерні системи», (6 – 8 квітня 2010 р., Київ, Україна). – К. : НТУУ «КПІ», 2010. – С. 144 – 146.

195. Жабин В. И., Жабина В. В., Безгинский М. А., Жабин В. И., Жабина В. В., Безгинский М. А. Эффективность реализации потоковых вычислений в системах с непосредственными связями, реализованных на ПЛИС // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка : зб. наук. праць. – К. : Век+, 2012. – №55. – С. 149-156.

196. Жабина В. В. Параллельное формирование команд в потоковых системах / В. В. Жабина // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка: Збірка наукових праць. – К. : Век+, 2009. – № 50. – С. 113 – 117.

197. Жабіна В. В. Дослідження моделі потокової обчислювальної системи з паралельним формуванням команд / В. В. Жабіна, А. Ю. Кеда // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка : зб. наук. праць. – К. : Век+, 2012. – № 57. – С. 164-169.

198. Жабин В. И. Повышение эффективности параллельной обработки данных и обеспечение отказоустойчивости мультипроцессорных вычислительных систем реального времени / В. И. Жабин // Штучний інтелект. – 2015. – № 1- 2. – С. 87 – 97.

199. Пат. №2030785 СССЗ, МКВ G 06 F 15/16. Вычислительное устройство / В. І. Жабін, Г. В.Гончаренко, В. В.Макаров, В. В. Ткаченко. – №4867678: Заявлено 21.09.1990; Опубл. 10.03.1995. – 23 с.

200. Клименко И. А. Обеспечение отказоустойчивости потоковых систем на однотипных вычислительных модулях / И. А. Клименко, В. В. Жабина // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка : зб. наук. праць. – К. : ВЄК +, 2009. – № 51. – С. 166 – 171.

201. Томашевський В. М. Моделювання систем. – К. : Видавнича група BHV, 2005. – 352 с.

202. Кельтон В., Лоу А. Имитационное моделирование. Классика CS. 3- е вид. – СПб. : Питер; К. : Видавнича група BHV, 2004. – 847 с.

203. Глушко А.Н. Основные этапы подбора и адаптации психологических методик в целях профессионального психологического отбора – М. : ЦВМУ, 1991. – 119 с.

204. Programming With Assertions [Електронний ресурс]. – Oracle, 2016. – Режим доступу : http://docs.oracle.com/javase/8/docs/technotes/guides/ language/assert .html.@

ДОДАТОК А

А.1. Часові характеристики функціональних блоків різної розмірності для виконання операції підсумовування матриць

Сімейство мікросхем	**Розмірність даних, біт**	**Частота роботи ФБ, МГц**	**Час обчислення, нс**	**Розмірність матриць,** $n \times n$
EP4CGX150DF31C8	12	128.78	7.77	2
EP4CGX110DF31C7	10	140.65	7.1	2
EP4CGX110DF27C7	8	140.45	7.12	2
EP4CGX30CF23C6	8	140.55	7.11	2
EP4CGX150DF31C8	16	128.45	7.79	4
EP4CGX150DF31C8	12	128.37	7.789	4
EP4CGX110DF31C7	10	140.65	7.109	4
EP4CGX110DF27C7	8	140.39	7.123	4
EP4CGX30CF23C6	8	127.86	7.82	4
EP4CGX150DF31C8	64	86.93	11.503	64
EP4CGX110DF31C7	64	96.73	10.338	64
EP4CGX110DF27C7	32	130.86	7.641	64
EP4CGX30CF23C6	32	127.28	7.856	64
EP4CGX150DF31C8	64	79.49	12.58	128
EP4CGX110DF31C7	64	91.52	10.926	128
EP4CGX110DF27C7	32	87.8	11.389	128
EP4CGX30CF23C6	32	127.15	7.864	128
EP4CGX150DF31C8	64	64.99	15.386	256
EP4CGX110DF31C7	64	73.24	13.653	256
EP4CGX110DF27C7	32	86.71	11.532	256
EP4CGX30CF23C6	32	96.02	10.414	246

А.2. Результати дослідження часових характеристик функціональних блоків різної розмірності для виконання операції множення матриць на константу

Сімейство мікросхем	**Розмірність даних, біт**	**Частота роботи ФБ, МГц**	**Час обчислення, нс**	**Розмірність матриць,** $n \times n$
EP4CGX150DF31C8	8	64.11	15.598	4x4, 4x4
EP4CGX110DF31C7	8	73.21	13.659	4x4, 4x4
EP4CGX110DF27C7	7	71.6	13.966	4x4, 4x4
EP4CGX30CF23C6	4	92.33	11.130	4x4, 4x4
EP4CGX150DF31C8	11	62.58	15.979	4x3, 3x4
EP4CGX110DF31C7	10	73.78	13.553	4x3, 3x4
EP4CGX110DF27C7	8	71.93	13.902	4x3, 3x4
EP4CGX30CF23C6	4	100.64	9.936	4x3, 3x4
EP4CGX150DF31C8	12	79.03	12.653	2x2, 2x2
EP4CGX110DF31C7	10	86.11	11.613	2x2, 2x2
EP4CGX110DF27C7	8	89.84	11.130	2x2, 2x2
EP4CGX30CF23C6	4	117.16	8.535	2x2, 2x2
EP4CGX150DF31C8	5	118.85	8.413	8x1, 1x8
EP4CGX150DF31C8	4	75.32	13.276	8x2, 2x8

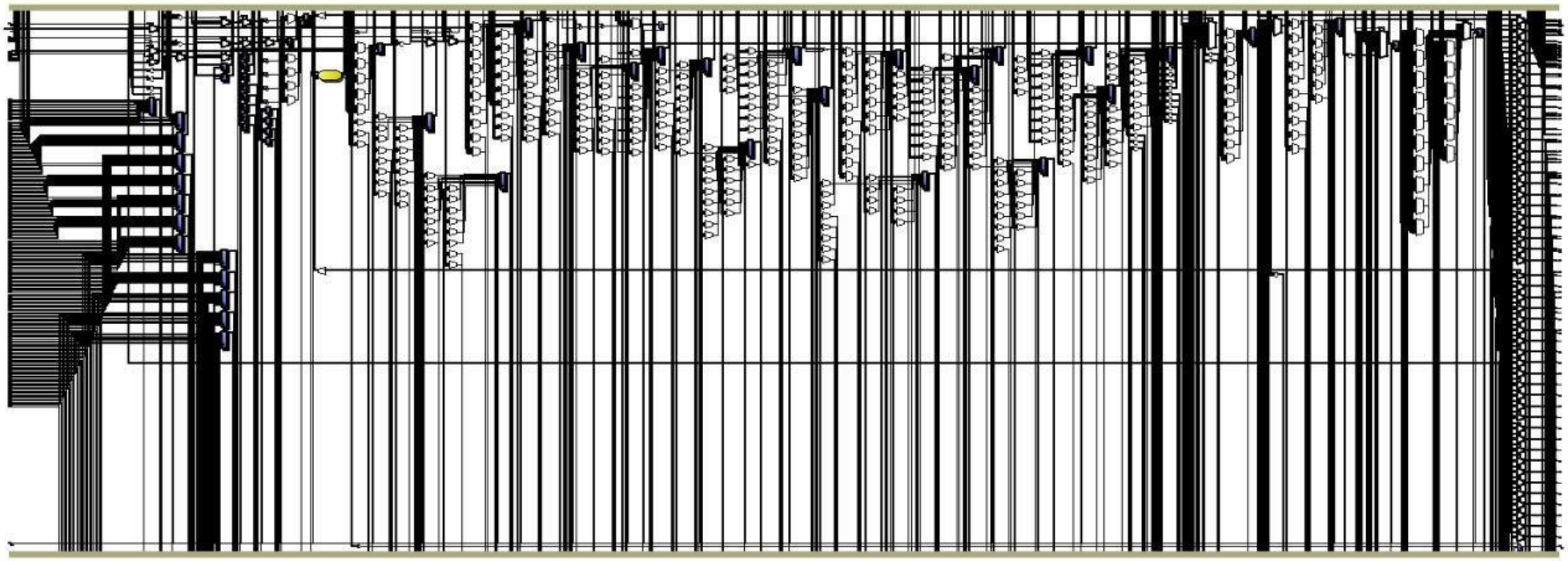
А.3. Часові характеристики функціональних блоків для реалізації часткової динамічної реконфігурації обчислювального середовища на ПЛІС [15, 68, 70]

Розмір прошивки в САПР, байт	**Реальний розмір прошивки, байт**	**Час конфігурування, такт**	**Час конфігурування, мс**
5904	6632	1661	0,01661
11808	12540	3138	0,03128
17712	18444	4614	0,04614
23616	24348	6090	0,06090
47232	48148	12040	0,12040
70848	71948	17990	0,17990
94464	95748	23940	0,23940
118080	119548	29890	0,29890
141696	143348	35840	0,35840
165312	167148	41790	0,41790

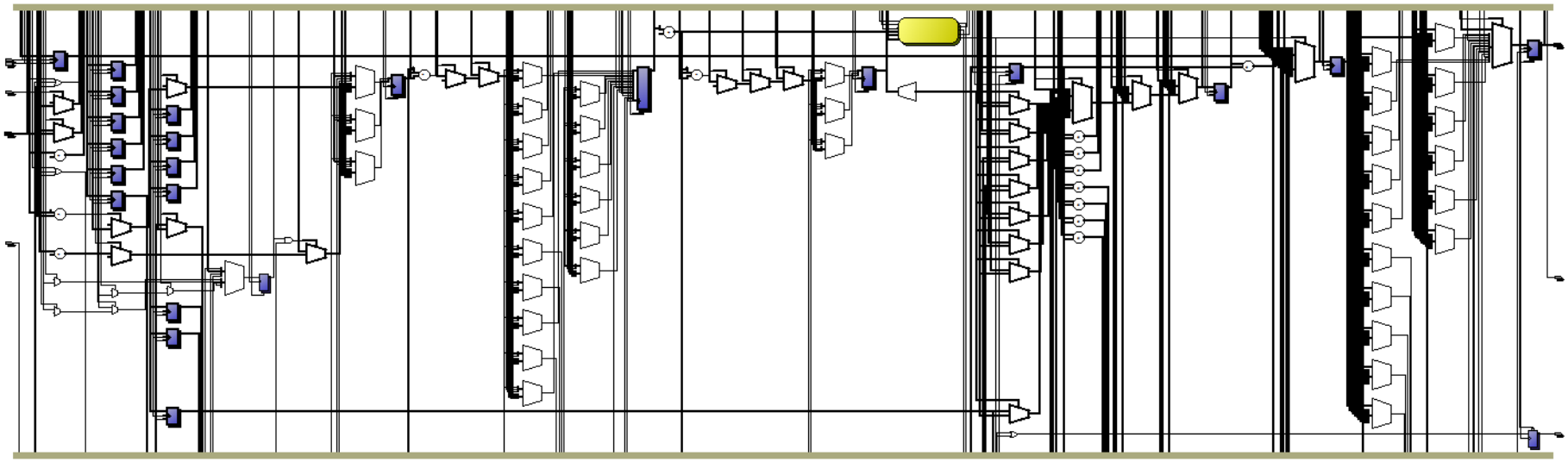
ДОДАТОК Б

Б.1. Функціональна схема в САПР функціонального блоку виконання операції підсумовування матриць розмірність 8 × 8

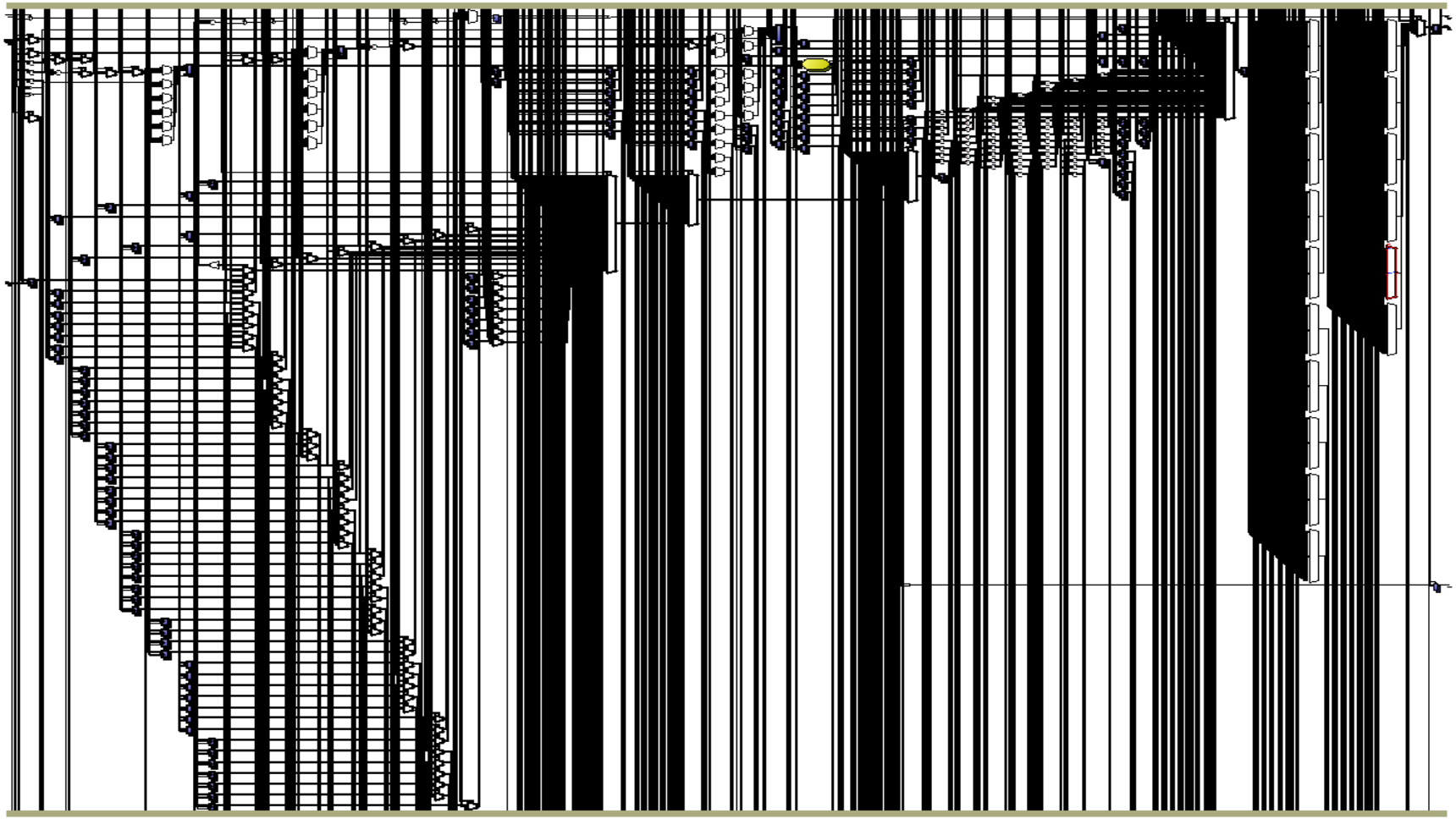
Б.2. Функціональна схема функціонального блоку в САПР виконання операції підсумовування матриць розмірністю 16 × 16



Б.3. Функціональна схема функціонального блоку в САПР виконання операції множення матриць розмірністю 16 × 16 на константу



Б.4. Функціональна схема в САПР функціонального блоку виконання операції множення матриць розмірністю 64 × 64 на константу



ДОДАТОК В

В.1. Код програми реалізації функціональних блоків різної розмірності для виконання операції підсумовування матриць

```
`timescale 1ns / 1ps

// Matrix:
// m = 4
// n = 3
// + +-----n-----+
// | |A11 A12 A13|
// m |A21 A22 A23|
// | |A31 A32 A33|
// | |A41 A42 A43|

// Input data:
// |-matrices_count-| 0
// |---matrix_a_0---| 1
// |---matrix_a_1---| 2
// |---.........---| ...
// |---matrix_a_N---| matrices_count - 1

// |---matrix_b_0---| matrices_count
// |---matrix_b_1---| matrices_count + 1
// |---.........---| ...
// |---matrix_b_N---| 2 * matrices_count - 1
//

module matrices_sum_int(i_clk, i_rst_n, i_push, i_data, i_pop, o_data, o_ready,
o_res_avail);
      function integer clogb2;
            input [31:0] value;
            integer i;
            begin
                  for(i = 0; 2**i < value; i = i + 1)
                        clogb2 = i + 1;
            end
      endfunction

      parameter DATA_WIDTH = 16;
      parameter BUFFER_SIZE = 256; // buffer for operands, number must be power
of 2!
      parameter REGS_WIDTH = 16;

      input i_clk;
      input i_rst_n;
      input i_push;
      input [DATA_WIDTH - 1 : 0] i_data;
      input i_pop;

      output [DATA_WIDTH - 1 : 0] o_data;
      output o_ready;
      output o_res_avail;

      reg ready; // ready register, 1 - result is ready and can be popped, 0 -
device is busy

      assign o_ready = ready;
```

```
reg signed [DATA_WIDTH - 1 : 0] buffer [0 : BUFFER_SIZE - 1];
reg signed [DATA_WIDTH - 1 : 0] res_buffer [0 : BUFFER_SIZE / 2 - 1];
reg signed [DATA_WIDTH - 1 : 0] res_out_buffer;
reg [REGS_WIDTH - 1 : 0] matrices_count;
reg [clogb2(BUFFER_SIZE) - 1 : 0] buffer_pointer;
reg [clogb2(BUFFER_SIZE) - 1 : 0] m_counter;
reg [2 : 0] state;
reg res_avail;

assign o_data = res_out_buffer;
assign o_res_avail = res_avail;

localparam  IDLE = 0,
                  LOAD_SIZE = 1,
                  LOAD_OPERANDS_1 = 2,
                  LOAD_OPERANDS_2 = 3,
                  PERFORM_CALC = 4;

integer i;
always @(posedge i_clk, negedge i_rst_n) begin
      if (!i_rst_n) begin // async reset
            m_counter <= 0;
            ready <= 1'b1;
            state <= IDLE;
            buffer_pointer <= 0;
            matrices_count <= 0;
            res_out_buffer <= 0;
            res_avail <= 0;
      end else begin
            case(state)
                  IDLE : begin
                        m_counter <= 0;
                        ready <= 1'b1;
                        if (i_pop && res_avail) begin
                        res_out_buffer <= res_buffer[buffer_pointer];
                              buffer_pointer <= buffer_pointer + 1'b1;
                        if (buffer_pointer == matrices_count - 1) begin
                                    buffer_pointer <= 0;
                                    matrices_count <= 0;
                                    res_avail <= 0;
                              end
                        end
                        if (i_push) begin
                              state <= LOAD_SIZE;
                        end
                  end

                  LOAD_SIZE : begin
                        ready <= 0'b0;
                        if (i_push) begin
                              state <= LOAD_OPERANDS_1;
                              matrices_count <= i_data;
                        end else begin
                              state <= IDLE;
                              res_avail <= 0;
                              buffer_pointer <= 0;
                        end
                  end

                  LOAD_OPERANDS_1 : begin
                        ready <= 0'b0;
                        if (i_push) begin
                              if (m_counter < matrices_count) begin
```

```
                                    buffer[m_counter] <= i_data;
                                if (m_counter == matrices_count - 1) begin
                                        m_counter <= 0;
                                        state <= LOAD_OPERANDS_2;
                                    end else begin
                                        m_counter <= m_counter + 1'b1;
                                    end
                                end else begin
                                    state <= IDLE;
                                    res_avail <= 0;
                                    buffer_pointer <= 0;
                                end
                            end else begin
                                state <= IDLE;
                                res_avail <= 0;
                                buffer_pointer <= 0;
                            end
                        end

                        LOAD_OPERANDS_2 : begin
                            ready <= 0'b0;
                            if (i_push) begin
                                if (m_counter < matrices_count) begin
                            buffer[m_counter | (BUFFER_SIZE / 2)] <= i_data;
                                if (m_counter == matrices_count - 1) begin
                                        state <= PERFORM_CALC;
                                    end else begin
                                        m_counter <= m_counter + 1'b1;
                                    end
                                end else begin
                                    state <= IDLE;
                                    res_avail <= 0;
                                    buffer_pointer <= 0;
                                end
                            end else begin
                                state <= IDLE;
                                res_avail <= 0;
                                buffer_pointer <= 0;
                            end
                        end

                        PERFORM_CALC : begin
                            state <= IDLE;
                            ready <= 1'b1;
                            res_avail <= 1'b1;
                            for (i = 0; i < BUFFER_SIZE; i = i + 1) begin
                    res_buffer[i] <= buffer[i] + buffer[i + BUFFER_SIZE / 2];
                            end
                        end
                        default : begin
                            state <= IDLE;
                            res_avail <= 0;
                            buffer_pointer <= 0;
                        end
                    endcase

                end
            end

        endmodule
```

В.2. Код програми реалізації модифікованого способу автоматичної синхронізації для пристрою автоматичного розподілу завдань та синхронізації

```
#include <stdio.h>
#include "alt_types.h"
#include "io.h"
#include "altera_avalon_mailbox.h"
const int BASE = 0xf0000;
const int SIZE = 6;
const int byte_amount = 4;
const int sub_SHIFT = 1000;
const int proc_amount = 2;
const int proc_number = 0;
void read_row(int row_number, float* row, int start_index);
void make_step_ahead(float* divisor_row, float* curr_row,
    int start_point, int end_point, int column);
void make_step_back(float* x_row, float* curr_row,
    int start_point, int end_point, int row_number);

int main()
{
    printf(" core started\n");
    int i = 0;
    int length = SIZE / proc_amount;
    int start_point = proc_number * length; //always inclusive
    if (0 == start_point)
    {
        start_point = 1;
    }
    int end_point = start_point + length; // always exclusive

    float* divisor_row = (float* ) malloc((SIZE + 1) * byte_amount);
    float* curr_row = (float* ) malloc((SIZE + 1) * byte_amount);
    for (i = start_point; i < end_point; i++)
    {
      make_step_ahead(divisor_row, curr_row, start_point, end_point, i);
      printf(" --- made step ahead");
      printf("step = %d -- after pend\n", i);
    }
        for (i = end_point-1; i >=start_point; i--)
    {
        make_step_back(divisor_row, curr_row, start_point, end_point, i);
    }
    printf("delta time = %d ms \n", time);
    print_matrix(0);
    printf("x = ");
    int offset1 = SIZE * (SIZE + 1) * byte_amount;
    print_row(offset1, SIZE);
    printf(" --- finished ---\n");
    return 0;
}
void read_row(int row_number, float* row, int start_index)
{
    int i = start_index;
    for (i = start_index; i < (SIZE + 1); i++)
    {
        int int_value = IORD_32DIRECT(BASE,
                                      (row_number * (SIZE + 1) + i) *
byte_amount);
        int *pvalue = &int_value;
```

```
        float value = *((float*)pvalue);
        row[i] = value;
    }
}
void make_step_ahead(float* divisor_row, float* curr_row,
    int start_point, int end_point, int column)
{
    if (column >= end_point)
    {
        return;
    }
    read_row(column, divisor_row, column);
    float divisor = divisor_row[column];

    int next_column = column + 1;
    int i = start_point > next_column
        ? start_point
        : next_column;

    for (; i < end_point; i++)
    {
        read_row(i, curr_row, column);
        int j = column;
        for (j = column; j < (SIZE + 1); j++)
        {
            float result = curr_row[j] - curr_row[j] * divisor_row[j] /
divisor;
            float* fpointer = &result;
            int pointer_k_int = *((int*)fpointer);
            int offset = (i* (SIZE + 1) + j) * byte_amount;
            IOWR_32DIRECT(BASE, offset, pointer_k_int);
        }
    }
}
float get_value_from_sram(int offset)
{
    int int_value = IORD_32DIRECT(BASE, offset);
    int *pvalue = &int_value;
    float value = *((float*)pvalue);
    return value;
}
void make_step_back(float* x_row, float* curr_row,
    int start_point, int end_point, int row_number)
{
    if (! ((row_number >= start_point)
            && (row_number < end_point)))
    {
        return;
    }
    read_row(SIZE, x_row, 0);
    read_row(row_number, curr_row, 0);
    int i = SIZE - 1;
    float result = curr_row[SIZE];
    for (; i > row_number; i--)
    {
        result = result - x_row[i] * curr_row[i];
    }
    result = result / curr_row[row_number];
    float* fpointer = &result;
    int pointer_k_int = *((int*)fpointer);
    int offset = (SIZE * (SIZE + 1) + row_number)* byte_amount;
    IOWR_32DIRECT(BASE, offset, pointer_k_int);
}
```

ДОДАТОК Д

Опис інтерфейсу користувача моделюючого середовища

Головне вікно програми, зображене на рис. Д.1. Панель інструментів зображена на рис. Д.2. Зліва направо знаходяться кнопки для відкриття файлу опису алгоритму, збереження файлу алгоритму, додавання вершини, додавання зв'язків, редагування властивостей вершин і виконання моделювання.

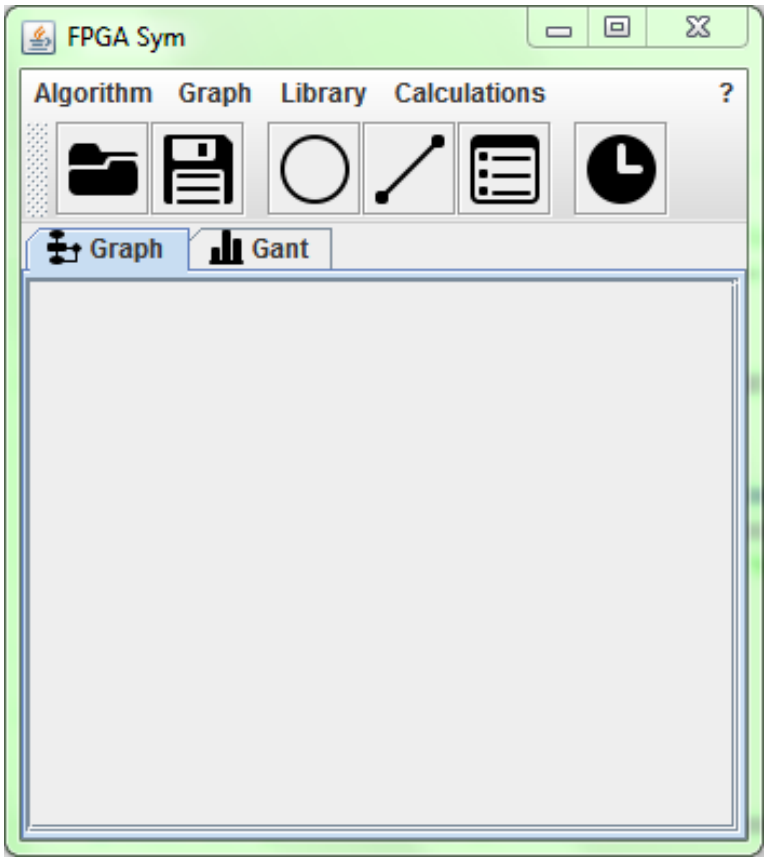


Рис. Д.1. Головне вікно програми

Панель інструментів має здатність змінювати місце розташування, тому може бути переміщена користувачем в будь-яку частину головного вікна програми.

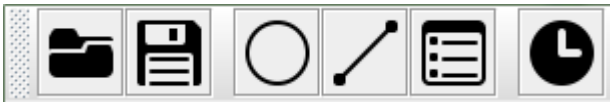


Рисунок Д.2 – Панель інструментів

Почати роботу користувач може із завантаження графу із попередньо збереженого файлу або створити його наново.

В першому випадку варто скористатись пунктом меню ***Algorithm – Open*** чи кнопкою на панелі інструментів. Після цього необхідно вибрати файл алгоритму з розширенням *xml* у діалоговому вікні (рис. Д.3).

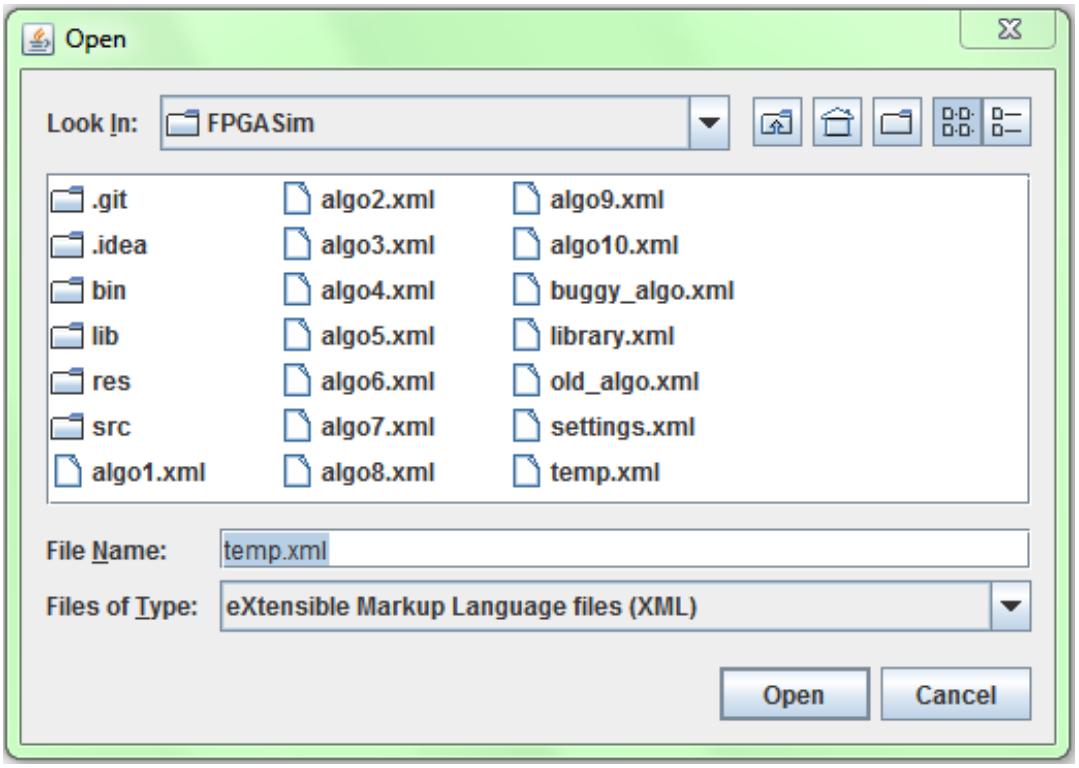


Рис. Д.3. Діалогове вікно відкриття файлу

В іншому випадку варто скористатись пунктами меню ***Graph – Add vertex*** та ***Graph – Add edges*** або кнопками на панелі інструментів із зображенням вершини та дуги, відповідно. Варто зазначити, що нова вершина з’явиться у верхньому лівому куті набірного поля, а її ідентифікатор буде більшим на одиницю, як це зображено на рис. Д.4.

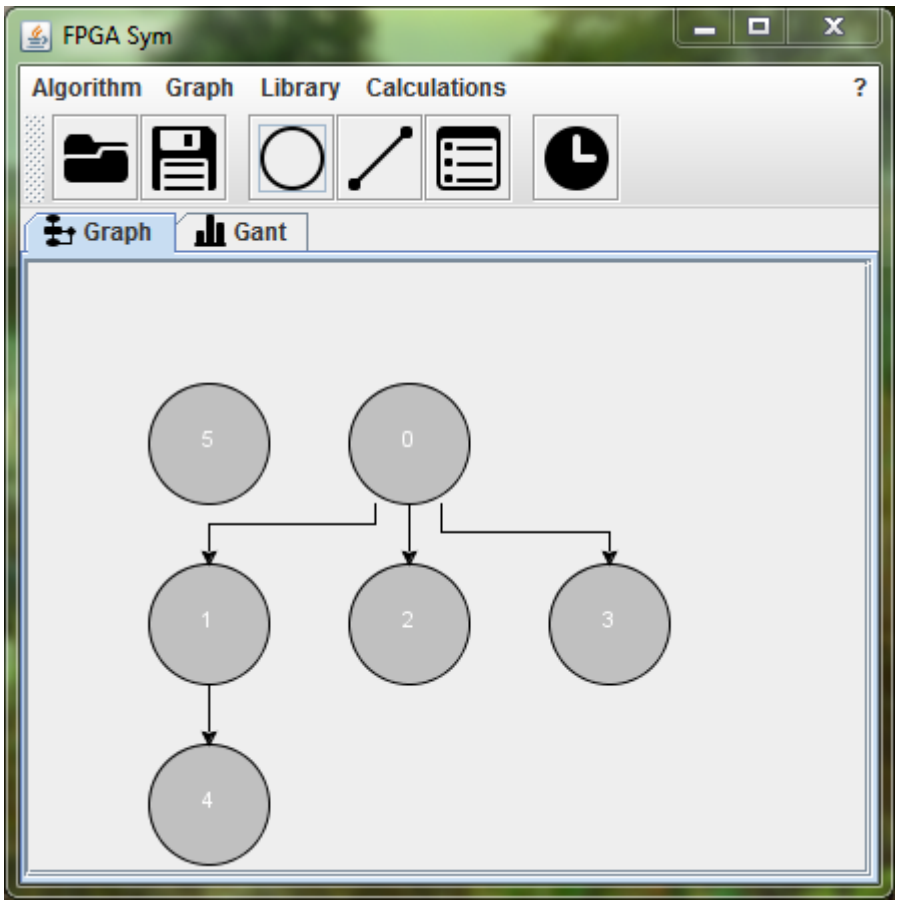


Рис. Д.4. Додавання нової вершини ГА

Додавання з’єднань між вершинами виконується у діалоговому вікні, що зображено на рис. Д.5. Вікно зображує таблицю з двома колонками: ***Від*** та ***До***. В них вказуються ідентифікатори вершин – початку та кінця з’єднання. Під таблицею знаходяться кнопки управління: ***Add***, ***Remove***, ***OK*** та ***Cancel***. Перші дві використовуються для додавання з’єднання або його видалення. Після цього, в таблиці необхідно обрати ідентифікатори початку та кінця з’єднання. Для зручності в кожному полі реалізовано випадаючий список всіх доступних вершин. Коли всі з’єднання вже додані, для закінчення редагування треба натиснути ***OK*** або ***Cancel*** для підтвердження або скасування виконаних змін.

Останнім кроком в створенні алгоритму є призначення кожній вершині певного номера апаратної задачі із списку доступної бібліотеки функціональних блоків. Це виконується в діалоговому вікні, зображеному на

рис. Д.6. Для його виклику необхідно вибрати пункт меню ***Graph – Properties*** або відповідну кнопку на панелі інструментів.

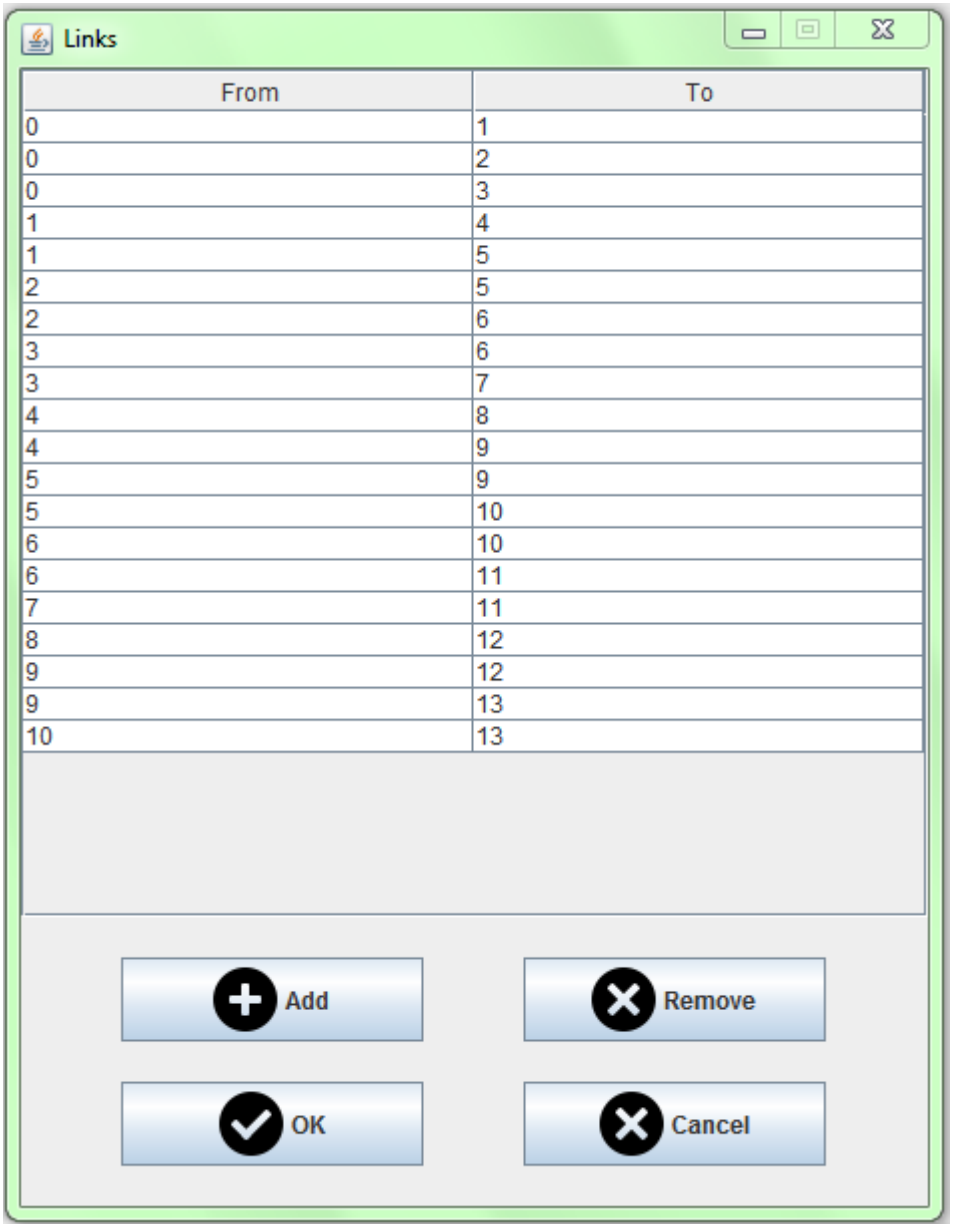


Рис. Д.5. Додавання дуг ГА

Вікно (рис. Д.6) реалізовано аналогічно до попереднього: в цій таблиці встановлюється відповідність між унікальним ідентифікатором вершини та апаратним номером задачі. Для зручності в полі реалізовано випадаючий список всіх доступних ідентифікаторів апаратних задач, що отримується з бібліотеки конфігураційних даних (рис. Д.6). Коли всі властивості

встановлені, для закінчення редагування треба натиснути ***OK*** або ***Cancel*** для прийняття або скасування виконаних змін.

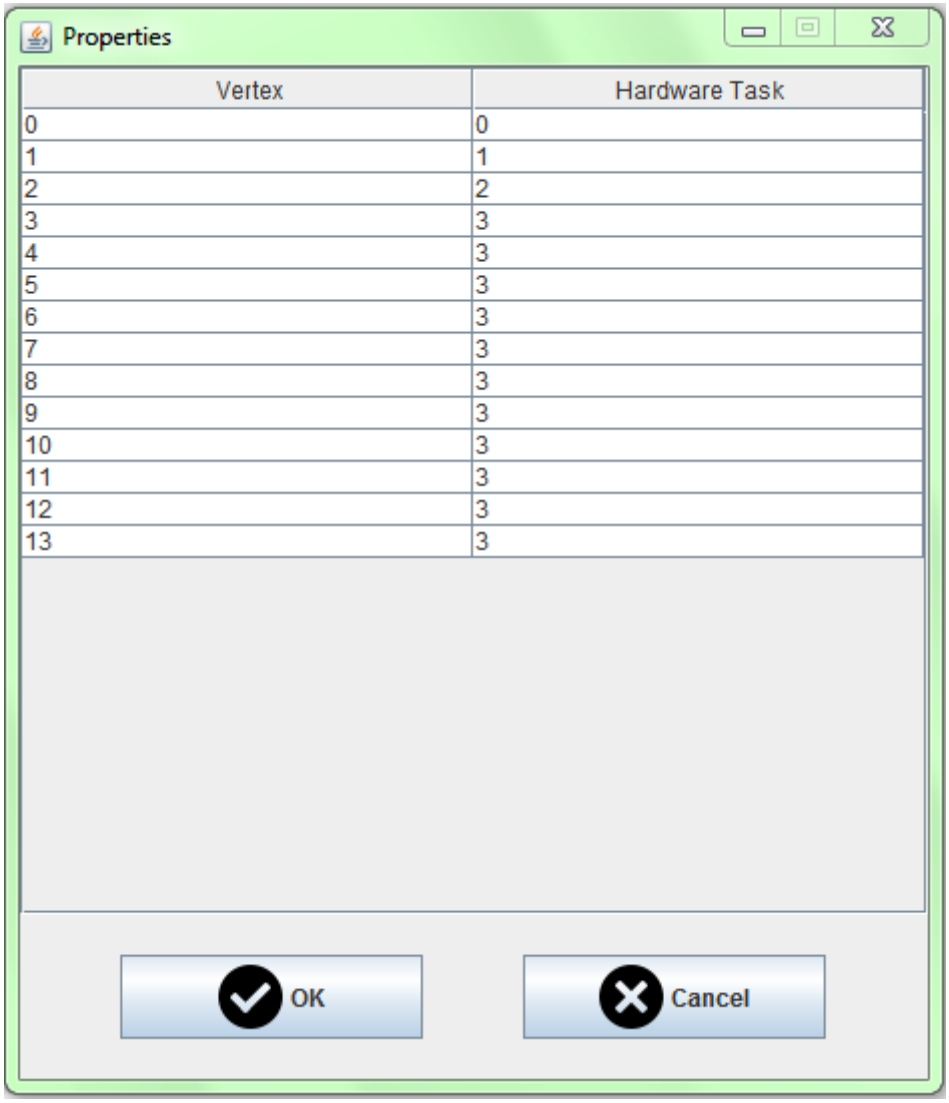


Рис. Д.6. Встановлення властивостей вершин ГА

Ідентифікатори апаратних задач разом з їх властивостями задач зберігаються в бібліотеці конфігураційних даних, що доступна в меню ***Library – Edit*** (рис. Д.7).

У діалоговому вікні (рис. Д.7) встановлюється відповідність між номером апаратної задачі та її властивостями: кількість слів в конфігураційному бітовому потоці (колонка ***BYTESTREAM_WORDS***), час роботи (колонка ***WORK_TIME***) та кількість слів даних (колонка

DATA_WORDS). Механізм зміни кількості апаратних задач не передбачено в графічному інтерфейсі. Проте, є можливість редагування файлу *library.xml*, що зберігає бібліотеку між сеансами роботи програми.

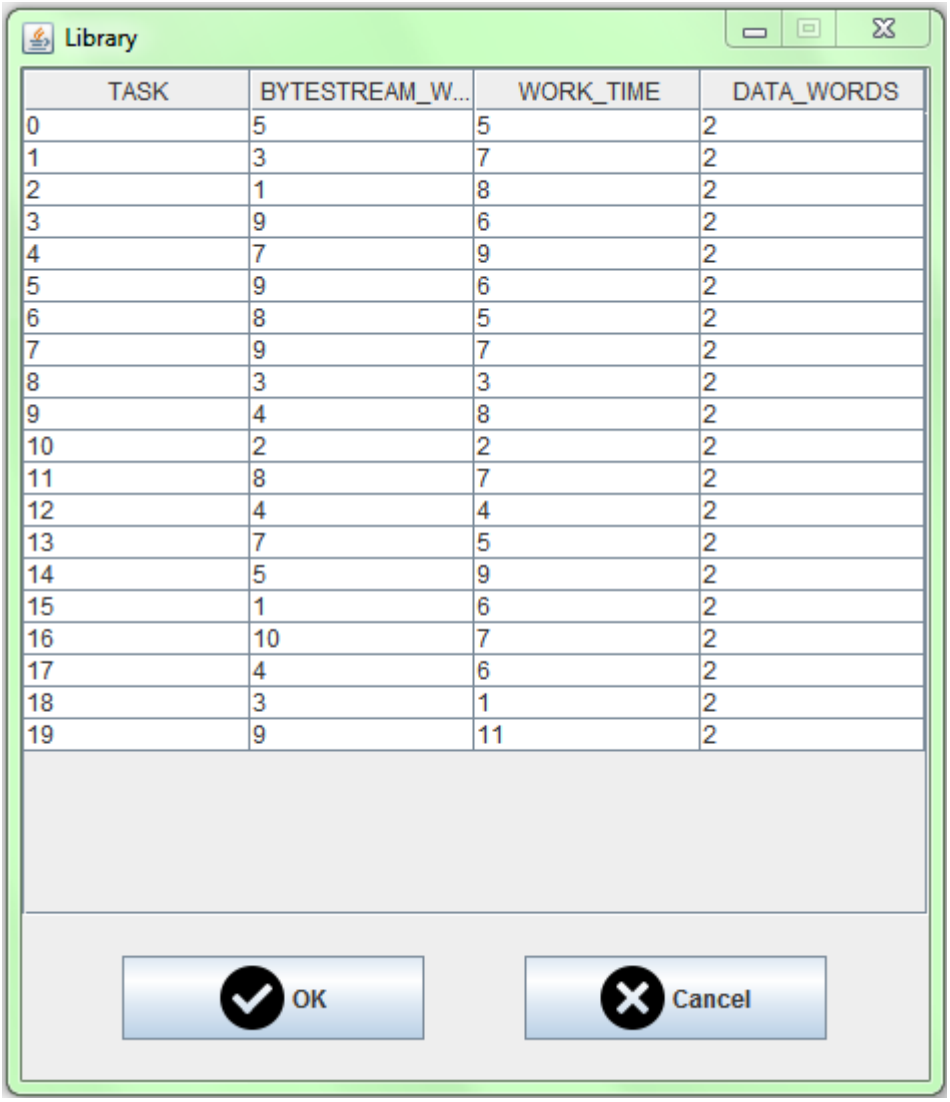


Рис. Д.7. Діалогове вікно БКД

В будь-який момент часу користувач може зберегти алгоритм за допомогою пункту меню ***Algorithm – Save*** чи відповідної кнопки на панелі інструментів. Діалогове вікно вводу назви файлу ілюструється на рис. Д.8.

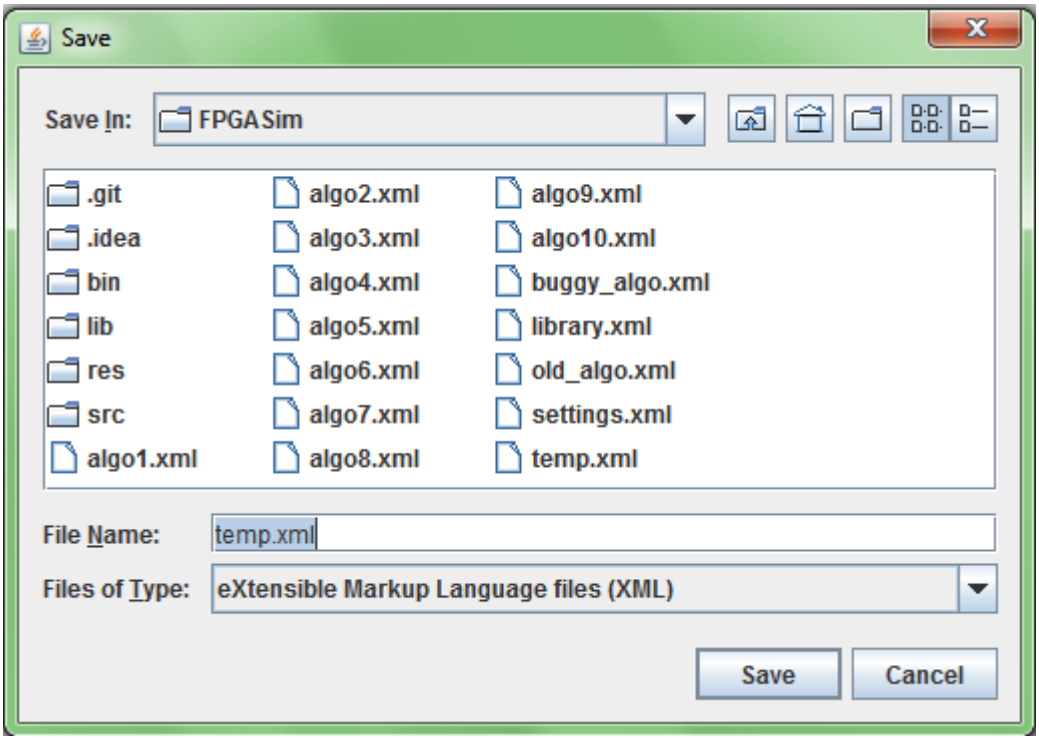


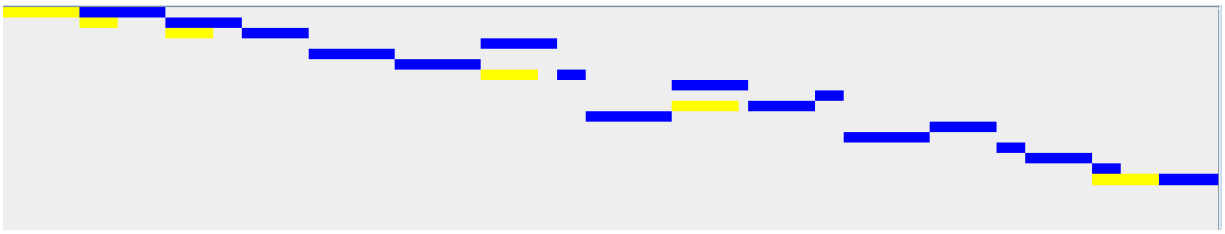
Рис. Д.8. Збереження алгоритму в файл

Для завершення роботи з програмою слід використати пункт меню ***?*** **–** ***Exit***. Ця дія виконується в диалоговому вікні рис. Д.9. Також, в цьому меню користувач може переглянути інформацію про автора (пункт ***? – About***).

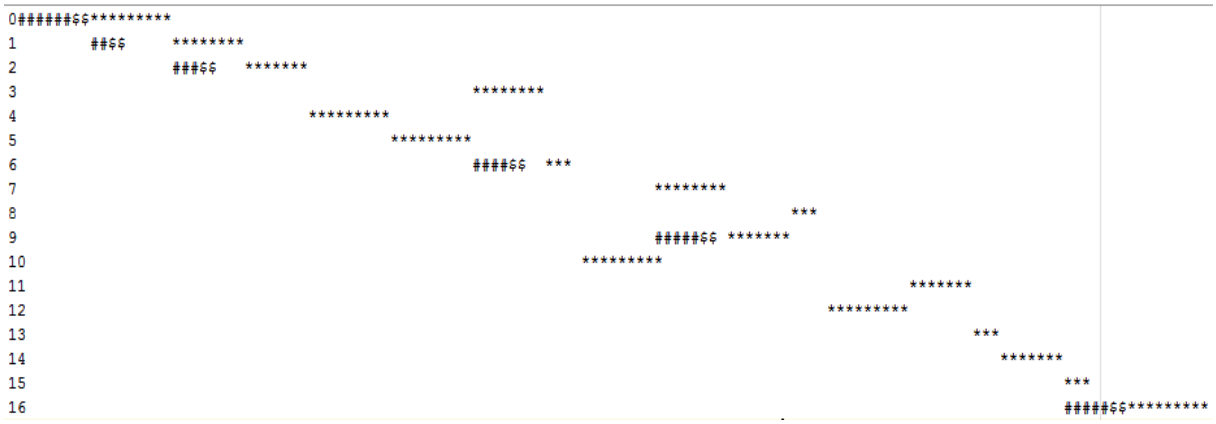
Рис. Д.9. Вихід з програми

Моделювання роботи алгоритму виконується вибором пункту меню ***Calculations – Simulate*** або натисканням на відповідну піктограму панелі інструментів. Після цього можна переходити на вкладку ***Gant***, щоб побачити зультати виконання моделювання у вигляді діаграми Ганта, що зображено на

рис. Д.10, *а*. У файлі *simulated.xml* зберігається низькорівневе відображення зображення діаграми Ганта в текстовому вигляді (рис. Д.10, *б*). Діаграми Ганта зображують спільні характеристики рівнів завантаження компонентів системи або відображують розклад їх роботи у вигляді відрізків, які розміщені на горизонтальній шкалі часу. Кожен відрізок відповідає окремому процесу. Процеси розміщуються по вертикалі.



а



б

Рис. Д.10. Результати виконання алгоритмів обчислювальних задач у вигляді діаграми Ганта: *а* – у графічному інтерфейсі користувача, *б* – у текстовому вигляді

ДОДАТОК Е

Програмний код реалізації основних модулів імітаційної моделі реконфігуровної комп'ютерної системи

```
1.   import com.thoughtworks.xstream.XStream;
2.   import gui.GraphPanel;
3.   import gui.TimeTracks;
4.   import sim.HardwareSystem;
5.   import sim.Simulator;
6.   import sim.SettingsHolder;
7.   import sim.Task;
8.   import java.io.File;
9.   import java.util.ArrayList;
10.   import java.util.HashSet;
11.  import java.util.List;
12.  import java.util.Set;
13.  /**
14.   * Class for automatic testing.
15.   */
16.  @SuppressWarnings("unused")
17.  public class Testbench {
18.   public static final int REPEAT_COUNT = 100;
19.   public static final int EXPERIMENTS_COUNT = 10;
20.   public static final int PROCESS_MIN = 0;
21.   public static final int PROCESS_AVG = 1;
22.   public static final int PROCESS_MAX = 2;
23.   public static final int WORK_AVG = 3;
24.   public static final int PROCESS_AVG_NO_PARALLEL = 4;
25.   static GraphPanel graph;
26.   public static void main(String[] args) {
27.         Double[] processTime = new Double[EXPERIMENTS_COUNT];
28.         Double[] workTime = new Double[EXPERIMENTS_COUNT];
29.         Double[] processTimeNoParallel = new Double[EXPERIMENTS_COUNT];
30.         String[]   files   =   {"algo1.xml",   "algo2.xml",   "algo3.xml",
"algo4.xml", "algo5.xml",
31.         "algo6.xml", "algo7.xml", "algo8.xml", "algo9.xml", "algo10.xml"};
32.         for (int i = 0; i < files.length; i++) {
33.               double[] res = experiment(files[i]);
34.               processTime[i] = res[PROCESS_AVG];
35.               workTime[i] = res[WORK_AVG];
36.               processTimeNoParallel[i]  =  res[PROCESS_AVG_NO_PARALLEL]  +
workTime[i];
37.         }
38.         System.out.println("Process time");
39.         for (Double aProcessTime : processTime) {
40.               System.out.println(aProcessTime.intValue());
41.         }
42.         System.out.println("Work time");
43.         for (Double aWorkTime : workTime) {
44.               System.out.println(aWorkTime.intValue());
45.         }
46.         System.out.println("Process without method time");
47.         for (Double aProcessTimeNoParallel : processTimeNoParallel) {
48.               System.out.println(aProcessTimeNoParallel.intValue());
49.         }
50.   }
```

```
51.   private static double[] experiment(String file) {
52.         TimeTracks[] repeating = new TimeTracks[REPEAT_COUNT];
53.         for (int i = 0; i < repeating.length; i++) {
54.               createEnvironment();
55.               open(file);
56.               repeating[i] = simulate(makeTaskLevels());
57.         }
58.         int sum = 0;
59.         int min = repeating[0].getMaxLength();
60.         int max = 0;
61.         int work = 0;
62.         int load = 0;
63.         for (TimeTracks r : repeating) {
64.               int maxLength = r.getMaxLength();
65.               sum += maxLength;
66.               min = Math.min(min, maxLength);
67.               max = Math.max(max, maxLength);
68.               work += r.getWorkingTime();
69.               load += r.getLoadingTime();
70.         }
71.         double allAvg = sum / repeating.length;
72.         double workAvg = work / repeating.length;
73.         double loadAvg = load / repeating.length;
74.         return new double[]{min, allAvg, max, workAvg, loadAvg};
75.   }
76.   private static void createEnvironment() {
77.         graph = new GraphPanel();
78.         Task.clearCounter();
79.   }
80.   private static void open(String file) {
81.         Object[]   memento   =   (Object[])   new   XStream().fromXML(new
File(file));
82.         int[][] trans = (int[][]) memento[0];
83.         for (int[] tran : trans) {
84.               graph.addVertex();
85.         }
86.         for (int i = 0; i < trans.length; i++) {
87.               for (int j = i; j < trans[i].length; j++) {
88.                     if (trans[i][j] == 1) {
89.                           graph.addEdge(i, j);
90.                     }
91.               }
92.         }
93.         graph.update(null, memento[1]);
94.         graph.update();
95.   }
96.   @SuppressWarnings("unchecked")
97.   private static List<Task>[] makeTaskLevels() {
98.         int[][] transitions = graph.createTransitions();
99.         int[] hwNumbers = graph.getPropertiesData();
100.              int levelsCounter = 0;
101.        int tasksCounter = 0;
102.        List<List<Task>> tasks = new ArrayList<>();
103.        List<Task> firstLevel = new ArrayList<>();
104.        firstLevel.add(new Task(hwNumbers[0]));
105.        tasksCounter++;
106.        levelsCounter++;
107.        tasks.add(firstLevel);
108.        while (tasksCounter != transitions.length) {
109.              List<Task> prevLevel = tasks.get(levelsCounter - 1);
110.              List<Task> level = new ArrayList<>();
111.              Set<Integer> visited = new HashSet<>();
112.              for (Task prevLevelTask : prevLevel) {
```

```
113.                    int[]                transitionsLine                =
transitions[prevLevelTask.getId()];
114.                    for (int j = 0; j < transitionsLine.length; j++) {
115.                          boolean notConnectedOnCurrentLevel = true;
116.                          for (Task levelTask : level) {
117.                    notConnectedOnCurrentLevel                           &=
transitions[levelTask.getId()][j] != 1;
118.                          }
119.                          boolean notVisited = !visited.contains(j);
120.                          boolean isConnected = transitionsLine[j] == 1;
121.                          if     (isConnected     &&     notVisited     &&
notConnectedOnCurrentLevel) {
122.                                level.add(new Task(hwNumbers[j]));
123.                                tasksCounter++;
124.                                visited.add(j);
125.                          }
126.                    }
127.              }
128.              tasks.add(level);
129.              levelsCounter++;
130.        }
131.        List<Task>[] result = new ArrayList[tasks.size()];
132.        int i = 0;
133.        for (List<Task> lst : tasks) {
134.              result[i] = lst;
135.              i++;
136.        }
137.        return result;
138.  }
139.  private static TimeTracks simulate(List<Task>[] levelsTasks) {
140.        SettingsHolder    settingsHolder    =    new    SettingsHolder(new
File("settings.xml"));
141.        Simulator        simulator        =        new        Simulator(new
HardwareSystem(settingsHolder), settingsHolder);
142.        return simulator.simulate(levelsTasks);
143.  }
144. }
1.  package sim;
2.  import java.util.ArrayList;
3.  import java.util.Collections;
4.  import java.util.HashSet;
5.  import java.util.Iterator;
6.  /**
7.   * Class represents state of system.
8.   */
9.  public class HardwareSystem {
10.   private final SettingsHolder settingsHolder;
11.   /**
12.    * contains hwN
13.    */
14.   private final ArrayList<Integer> FPGA = new ArrayList<>();
15.   /**
16.    * contains hwN, set is used to ignore adding 2 same tasks
17.    */
18.   private final HashSet<Integer> memory = new HashSet<>();
19.   /**
20.    * contains bonus points. Index bonus = index hwN in FPGA
21.    */
22.   private final ArrayList<Integer> bonuses = new ArrayList<>();
23.   public HardwareSystem(SettingsHolder settingsHolder) {
24.         this.settingsHolder = settingsHolder;
25.   }
26.   public State findConfiguration(Task task) {
```

```
27.         if (FPGA.contains(task.getHwN())) {
28.               return State.TSK_FPGA;
29.         } else {
30.               if (memory.contains(task.getHwN())) {
31.                     return State.TSK_MEM;
32.               } else {
33.                     return State.TSK_LIB;
34.               }
35.         }
36.   }
37.   public void load(Task task) {
38.         int fpgaMaxSize = settingsHolder.getFpgaSize();
39.         int maxBonus = settingsHolder.getBonus();
40.         int memorySize = settingsHolder.getMemorySize();
41.         if (FPGA.size() == fpgaMaxSize) {
42.               int smallestBonus = Collections.min(bonuses);
43.               int  smallestBonusIndex  =  Collections.binarySearch(bonuses,
smallestBonus);
44.               int smallestBonusHwN = FPGA.get(smallestBonusIndex);
45.               FPGA.remove(new Integer(smallestBonusHwN));
46.               bonuses.remove(smallestBonusIndex);
47.               if (memory.size() == memorySize) {
48.                     Iterator<Integer> iterator = memory.iterator();
49.                     iterator.next();
50.                     iterator.remove();
51.               }
52.               memory.add(smallestBonusHwN);
53.         }
54.         FPGA.add(task.getHwN());
55.         for (int i = 0; i < bonuses.size(); i++) {
56.               bonuses.set(i, bonuses.get(i) - 1);
57.         }
58.         bonuses.add(maxBonus);
59.   }
60.   public enum State {
61.         TSK_FPGA, TSK_MEM, TSK_LIB
62.   }
63. }
1.  package sim;
2.  import com.thoughtworks.xstream.XStream;
3.  import javax.swing.table.AbstractTableModel;
4.  import java.io.File;
5.  /**
6.   * Class represents library of hardware tasks.
7.   */
8.  public class Library extends AbstractTableModel {
9.    public static final String LIBRARY_FILE = "library.xml";
10.   private static final long serialVersionUID = -8149438991804788594L;
11.   private final int[][] data;
12.   public Library() {
13.         this(Library.LIBRARY_FILE);
14.   }
15.   public Library(String path) {
16.         data = (int[][]) new XStream().fromXML(new File(path));
17.         if ((data.length == 0) || (data[0].length != 3)) {
18.               throw new IllegalArgumentException("Wrong library file!");
19.         }
20.   }
21.   public int[][] getData() {
22.         return data;
23.   }
24.   @Override
25.   public int getRowCount() {
```

```
26.         return data.length;
27.   }
28.   @Override
29.   public int getColumnCount() {
30.         return 4;
31.   }
32.   public int getSize() {
33.         return getRowCount();
34.   }
35.   @Override
36.   public String getColumnName(int columnIndex) {
37.         switch (columnIndex) {
38.               case 0:
39.                     return "TASK";
40.               case 1:
41.                     return "BYTESTREAM_WORDS";
42.               case 2:
43.                     return "WORK_TIME";
44.               case 3:
45.                     return "DATA_WORDS";
46.         }
47.         return null;
48.   }
49.   @Override
50.   public boolean isCellEditable(int rowIndex, int columnIndex) {
51.         return true;
52.   }
53.   @Override
54.   public Object getValueAt(int rowIndex, int columnIndex) {
55.         if (columnIndex == 0) {
56.               return rowIndex;
57.         }
58.         return data[rowIndex][columnIndex - 1];
59.   }
60.   @Override
61.   public void setValueAt(Object value, int row, int column) {
62.         if (column != 0) {
63.               try {
64.                     data[row][column   -   1]   =   Integer.valueOf((String)
value);
65.               } catch (Exception e) {
66.                     System.err.println("Exception " + e.getMessage());
67.               }
68.         }
69.   }
70. }
1.  package sim;
2.  import com.thoughtworks.xstream.XStream;
3.  import java.io.File;
4.  /**
5.   * Class encapsulates way to get all necessary constants.
6.   */
7.  public class SettingsHolder {
8.    public static final int DEFAULT_MEMORY_MAX_SIZE = 32;
9.    private static final int DEFAULT_FPGA_MAX_SIZE = 9;
10.   private static final int DEFAULT_MAX_BONUS = 10;
11.   private static final int DEFAULT_MEMORY_ACCESS_TIME = 1;
12.   private static final int DEFAULT_LOAD_LAST_TIME = 10;
13.   private static final int DEFAULT_LOAD_DATUM_TIME = 2;
14.   private static final int DEFAULT_NETWORK_MAX_RANDOM_TIME = 5;
15.   private Settings settings = new Settings();
16.   public SettingsHolder(File file) {
17.         try {
```

```
18.               XStream stream = new XStream();
19.               settings = (Settings) stream.fromXML(file);
20.         } catch (Exception e) {
21.               e.printStackTrace();
22.               // forget, using defaults
23.         }
24.   }
25.   public int getLoadLastWordTime() {
26.         return settings.loadLastWordTime;
27.   }
28.   public int getFpgaSize() {
29.         return settings.fpgaSize;
30.   }
31.   public int getBonus() {
32.         return settings.bonus;
33.   }
34.   public int getMemoryAccessTime() {
35.         return settings.memoryAccessTime;
36.   }
37.   public int getLoadDatumTime() {
38.         return settings.loadDatumTime;
39.   }
40.   public int getNetworkMaxRandomTime() {
41.         return settings.networkMaxRandomTime;
42.   }
43.   public int getMemorySize() {
44.         return settings.memorySize;
45.   }
46.   private static class Settings {
47.         final int fpgaSize;
48.         final int bonus;
49.         final int memoryAccessTime;
50.         final int loadDatumTime;
51.         final int networkMaxRandomTime;
52.         final int loadLastWordTime;
53.         final int memorySize;
54.         private Settings() {
55.               this(
56.                           DEFAULT_FPGA_MAX_SIZE,
57.                           DEFAULT_MAX_BONUS,
58.                           DEFAULT_MEMORY_ACCESS_TIME,
59.                           DEFAULT_LOAD_DATUM_TIME,
60.                           DEFAULT_NETWORK_MAX_RANDOM_TIME,
61.                           DEFAULT_LOAD_LAST_TIME,
62.                           DEFAULT_MEMORY_MAX_SIZE
63.               );
64.         }
65.         private Settings(int fpgaSize, int bonus, int memoryAccess, int
loadDatum, int networkMaxRandom, int loadLast, int memorySize) {
66.               this.fpgaSize = fpgaSize;
67.               this.bonus = bonus;
68.               this.memoryAccessTime = memoryAccess;
69.               this.loadDatumTime = loadDatum;
70.               this.networkMaxRandomTime = networkMaxRandom;
71.               this.loadLastWordTime = loadLast;
72.               this.memorySize = memorySize;
73.         }
74.   }
75. }
1.   package sim;
2.   import gui.TimeTracks;
3.   import java.io.IOException;
4.   import java.io.UncheckedIOException;
```

```
5.   import java.nio.file.Files;
6.   import java.nio.file.Paths;
7.   import java.util.*;
8.   /**
9.    * New class for simulating system work.
10.    */
11.  public class Simulator {
12.   private final HardwareSystem hardwareSystem;
13.   private final SettingsHolder settingsHolder;
14.   public    Simulator(HardwareSystem    hardwareSystem,    SettingsHolder
settingsHolder) {
15.         this.hardwareSystem = hardwareSystem;
16.         this.settingsHolder = settingsHolder;
17.   }
18.   public TimeTracks simulate(List<Task>[] levelsTasks) {
19.         Objects.requireNonNull(levelsTasks, "Null levelsTasks!");
20.         if (levelsTasks.length < 2 || levelsTasks[0].size() != 1) {
21.               throw new IllegalArgumentException("Wrong algo!");
22.         }
23.         int memoryAccessTime = settingsHolder.getMemoryAccessTime();
24.         int                      networkMaxRandomTime                      =
settingsHolder.getNetworkMaxRandomTime();
25.         int loadLastWordTime = settingsHolder.getLoadLastWordTime();
26.         int loadDatumTime = settingsHolder.getLoadDatumTime();
27.  System.out.printf("Start working with time: memoryAccess = %s loadLastWord
= %s loadDatum = %s%n",
28.                     memoryAccessTime, loadLastWordTime, loadDatumTime);
29.         List<Task> allTasks = new ArrayList<>();
30.         for (List<Task> levelsTask : levelsTasks) {
31.               allTasks.addAll(levelsTask);
32.         }
33.         int tasksCount = allTasks.size();
34.         int level = 0;
35.         List<Task> currentLevel = levelsTasks[level];
36.         TimeTracks time = new TimeTracks(tasksCount);
37.         Set<Task> finished = new HashSet<>();
38.         // first level
39.         Task first = currentLevel.get(0);
40.         int id1 = first.getId();
41.         int libTime1 = first.getBytestreamWords() * memoryAccessTime;
42.         int randTime1 = new Random().nextInt(networkMaxRandomTime);
43.         time.addSearchingAndLoading(id1, libTime1 + randTime1);
44.         time.addLoadingLastWord(id1, loadLastWordTime);
45.         hardwareSystem.load(first);
46.         System.out.printf("Load %s from library. LibTime = %s, randTime =
%s%n", first, libTime1, randTime1);
47.         time.addLoadingData(id1, first.getDataCount() * loadDatumTime);
48.         time.addWaitingToLongest();
49.         time.addCounting(id1, first.getWorkingTime());
50.         finished.add(first);
51.         currentLevel = levelsTasks[++level];
52.         Collections.sort(currentLevel);
53.         while (tasksCount != finished.size()) {
54.               // loading configuration
55.               for (Task t : currentLevel) {
56.                     int id = t.getId();
57.                     switch (hardwareSystem.findConfiguration(t)) {
58.                           case TSK_FPGA:
59.                                 hardwareSystem.load(t);
60.                                 System.out.printf("Load  %s  from  FPGA%n",
t);
61.                                 break;
62.                           case TSK_LIB:
```

```
63.                                 int  libTime  =  t.getBytestreamWords()  *
memoryAccessTime;
64.                                 int         randTime         =         new
Random().nextInt(networkMaxRandomTime);
65.                                 time.addSearchingAndLoading(id,  libTime  +
randTime);
66.                                 time.addLoadingLastWord(id,
loadLastWordTime);
67.                                 hardwareSystem.load(t);
68.   System.out.printf("Load %s from library. LibTime = %s, randTime = %s%n",
t, libTime, randTime);
69.                                 break;
70.                           case TSK_MEM:
71.                                 int  memTime  =  t.getBytestreamWords()  *
memoryAccessTime;
72.                                 time.addSearchingAndLoading(id, memTime);
73.                                 time.addLoadingLastWord(id,
loadLastWordTime);
74.                                 hardwareSystem.load(t);
75.                     System.out.printf("Load  %s  from  memory.  MemTime  =
%s%n", t, memTime);
76.                                 break;
77.                           default:
78.                     assert false : "Task can be found in FPGA or memory or
library only!";
79.                     }
80.                     // loading data
81.                     time.addLoadingData(id,       t.getDataCount()       *
loadDatumTime);
82.                     time.addWaitingToLongest();
83.                     // simulate working tasks
84.                     time.addCounting(id, t.getWorkingTime());
85.                     finished.add(t);
86.               }
87.               // prepare to next iteration
88.               if (level < (levelsTasks.length - 1)) {
89.                     currentLevel = levelsTasks[++level];
90.                     Collections.sort(currentLevel);
91.               } else {
92.                     writeGantt(time);
93.               }
94.               // prediction
95.               for (Task t : currentLevel) {
96.                     int predictedLoadingTime = 0;
97.                     switch (hardwareSystem.findConfiguration(t)) {
98.                           case TSK_MEM:
99.         predictedLoadingTime = (t.getBytestreamWords() * memoryAccessTime)
+ loadLastWordTime;
100.                                      break;
101.                          case TSK_LIB:
102.                                predictedLoadingTime                    =
(t.getBytestreamWords()    *    memoryAccessTime)    +    loadLastWordTime    +
networkMaxRandomTime;
103.                                break;
104.                    }
105.              int    timeToCalculationsEnd    =    time.getMaxLength()    -
time.getLength(t.getId());
106.                    int      delta      =      timeToCalculationsEnd      -
predictedLoadingTime;
107.                    if (delta > 0) {
108.                          for (int i = 0; i < time.getTracksCount(); i++) {
109.                    boolean   notMaxLengthTrack   =   time.getLength(i)   !=
time.getMaxLength();
```

```
110.                                if (notMaxLengthTrack) {
111.                                      time.addWaiting(i, delta);
112.                                }
113.                          }
114.                    }
115.              }
116.        }
117.        return time;
118.  }
119.  private void writeGantt(TimeTracks time) {
120.        try {
121.              Files.write(Paths.get("simulated.txt"),
time.toString().getBytes());
122.        } catch (IOException e) {
123.              throw new UncheckedIOException(e);
124.        }
125.  }
126. }
1.  package sim;
2.  /**
3.   * Class represents simple hardware task with properties: number, work time,
4.   * byte-stream words and data words.
5.   */
6.  public class Task implements Comparable<Task> {
7.    private static int counter = 0;
8.    private static Library lib = new Library();
9.    private final int id = Task.counter++;
10.   private final int hwN;
11.   private final int workTime;
12.   private final int bytestreamWords;
13.   private final int dataCount;
14.   public Task(int hwN) {
15.         this.hwN = hwN;
16.         int[][] data = Task.lib.getData();
17.         workTime = data[hwN][1];
18.         bytestreamWords = data[hwN][0];
19.         dataCount = data[hwN][2];
20.   }
21.   public static void clearCounter() {
22.         counter = 0;
23.   }
24.   public int getId() {
25.         return id;
26.   }
27.   public int getHwN() {
28.         return hwN;
29.   }
30.   public int getWorkingTime() {
31.         return workTime;
32.   }
33.   public int getBytestreamWords() {
34.         return bytestreamWords;
35.   }
36.   public int getDataCount() {
37.         return dataCount;
38.   }
39.   @Override
40.   public boolean equals(Object o) {
41.         if (this == o) return true;
42.         if (o == null || getClass() != o.getClass()) return false;
43.         Task task = (Task) o;
44.         return id == task.getId();
45.   }
```

```
46.   @Override
47.   public int hashCode() {
48.         int result = id;
49.         result = 31 * result + hwN;
50.         result = 31 * result + workTime;
51.         result = 31 * result + bytestreamWords;
52.         result = 31 * result + dataCount;
53.         return result;
54.   }
55.   @Override
56.   public int compareTo(Task o) {
57.         if (o.workTime > workTime) {
58.               return 1;
59.         }
60.         if (o.workTime < workTime) {
61.               return -1;
62.         }
63.         return 0;
64.   }
65.   @Override
66.   public String toString() {
67.         return String.join("", "hwN = ", String.valueOf(hwN), " id = ",
68.                     String.valueOf(id));
69.   }
70. }
1.  package gui;
2.  import javax.swing.*;
3.  import java.awt.*;
4.  import static javax.swing.JOptionPane.*;
5.  /**
6.   * Abstract class for representing frames in application. Please, make sure,
7.   * that all of them use it as superclass.
8.   * <p>
9.   * Class extends class from the standard library - {@link JFrame}.
10.   */
11. public abstract class Frame extends JFrame {
12.   private static final long serialVersionUID = 1581013918976729599L;
13.   public Frame(String title) {
14.         super(title);
15.   }
16.   public static void showError(Throwable e) {
17.         showMessageDialog(null,     e.getMessage(),     "Error     "     +
e.getClass().getSimpleName(), ERROR_MESSAGE, icon("error_big"));
18.   }
19.   public static void showError(String msg) {
20.         showMessageDialog(null,     msg,     "Error",     ERROR_MESSAGE,
icon("error_big"));
21.   }
22.   public static void showInfo(String msg) {
23.         showMessageDialog(null,    msg,    "Info",    INFORMATION_MESSAGE,
icon("info_big"));
24.   }
25.   public static boolean showQuestion(String msg) {
26.         int    result    =    showConfirmDialog(null,    msg,    "Question",
YES_NO_OPTION, QUESTION_MESSAGE, icon("question_big"));
27.         return result == YES_OPTION;
28.   }
29.   public static void showWarning(String msg) {
30.         showMessageDialog(null,    msg,    "Warning",    WARNING_MESSAGE,
icon("warning_big"));
31.   }
32.   private static ImageIcon icon(String iconName) {
33.         return new ImageIcon("res\\" + iconName + ".png");
```

```
34.   }
35.   /**
36.    * Moves this component to a screen center location.
37.    */
38.   protected void moveToScreenCenter() {
39.         Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
40.         double x = (d.getWidth() - getWidth()) / 2;
41.         double y = (d.getHeight() - getHeight()) / 2;
42.         setLocation((int) x, (int) y);
43.   }
44.   /**
45.    * Abstract class for representing actions. Please, make sure, that all
of
46.    * them (in menu items, buttons and toolbar buttons) use it as superclass.
47.    * <p>
48.    *  Class  uses  default  implementation  of  {@link  javax.swing.Action}
interface
49.    * - abstract class {@link AbstractAction}.
50.    *
51.    * @author Mir4ik
52.    * @version 0.1 18.03.2015
53.    */
54.   protected abstract class Action extends AbstractAction {
55.         private static final long serialVersionUID = 3675827226774345460L;
56.         /**
57.          * Note, that constructor can take <code>null</code> values.
58.          *
59.          * @param name      action name
60.          * @param smallIcon path to small image
61.          * @param largeIcon path to large image
62.          */
63.         public Action(String name, String smallIcon, String largeIcon) {
64.               putValue(javax.swing.Action.NAME, name);
65.               putValue(javax.swing.Action.SMALL_ICON,                    new
ImageIcon(smallIcon));
66.               putValue(javax.swing.Action.LARGE_ICON_KEY,                new
ImageIcon(largeIcon));
67.         }
68.   }
69. }
1.  package gui;
2.  import javax.swing.*;
3.  import java.awt.*;
4.  /**
5.   * Class represents panel for diagram drawing.
6.   */
7.  public class GantDiagramPanel extends JPanel {
8.    private static final int SCALING = 10;
9.    private static final long serialVersionUID = -2465217224577309839L;
10.   private final TimeTracks data;
11.   public GantDiagramPanel(TimeTracks data) {
12.         this.data = data;
13.   }
14.   @Override
15.   public void paint(Graphics g) {
16.         for (int i = 0; i < data.getTracksCount(); i++) {
17.               String temp = data.getTrack(i);
18.               for (int j = 0; j < temp.length(); j++) {
19.                     switch (temp.charAt(j)) {
20.                           case ' ':
21.                     drawEmpty(g,   j   *   GantDiagramPanel.SCALING,   i   *
GantDiagramPanel.SCALING);
22.                                 break;
```

```
23.                             case '#':
24.                         drawLoad(g,   j   *   GantDiagramPanel.SCALING,   i   *
GantDiagramPanel.SCALING);
25.                                 break;
26.                             case '$':
27.                 drawLoadLast(g,   j   *   GantDiagramPanel.SCALING,   i   *
GantDiagramPanel.SCALING);
28.                                 break;
29.                             case '&':
30.                         drawData(g,   j   *   GantDiagramPanel.SCALING,   i   *
GantDiagramPanel.SCALING);
31.                                 break;
32.                             case '*':
33.                         drawWork(g,   j   *   GantDiagramPanel.SCALING,   i   *
GantDiagramPanel.SCALING);
34.                                 break;
35.                         }
36.                 }
37.             }
38.     }
39.     private void drawEmpty(Graphics g, int x, int y) {
40.             g.setColor(getBackground());
41.             draw(g, x, y);
42.     }
43.     private void drawLoad(Graphics g, int x, int y) {
44.             g.setColor(Color.YELLOW);
45.             draw(g, x, y);
46.     }
47.     private void drawLoadLast(Graphics g, int x, int y) {
48.             g.setColor(Color.YELLOW);
49.             draw(g, x, y);
50.     }
51.     private void drawData(Graphics g, int x, int y) {
52.             g.setColor(Color.YELLOW);
53.             draw(g, x, y);
54.     }
55.     private void drawWork(Graphics g, int x, int y) {
56.             g.setColor(Color.BLUE);
57.             draw(g, x, y);
58.     }
59.     private void draw(Graphics g, int x, int y) {
60.             g.drawRect(x,               y,              GantDiagramPanel.SCALING,
GantDiagramPanel.SCALING);
61.             g.fillRect(x,               y,              GantDiagramPanel.SCALING,
GantDiagramPanel.SCALING);
62.     }
63. }
1.    package gui;
2.    import com.mxgraph.layout.mxCompactTreeLayout;
3.    import com.mxgraph.model.mxCell;
4.    import com.mxgraph.model.mxICell;
5.    import com.mxgraph.swing.mxGraphComponent;
6.    import com.mxgraph.util.mxConstants;
7.    import com.mxgraph.util.mxUtils;
8.    import com.mxgraph.view.mxGraph;
9.    import com.mxgraph.view.mxStylesheet;
10.    import javax.swing.*;
11.   import java.awt.*;
12.   import java.util.*;
13.   import java.util.List;
14.   /**
15.    * Panel, containing graph of application.
16.    * <p>
```

```
17.   * Note, that this class uses pattern <tt>Observer</tt>, so it implements
18.   * interface {@link Observer}.
19.   */
20.  public class GraphPanel extends JPanel implements Observer {
21.   private static final long serialVersionUID = 5462589354217696759L;
22.   private static int counter = 0;
23.   private final mxGraph graph = new mxGraph();
24.   private final List<mxICell> vertexes = new ArrayList<>();
25.   private final List<mxICell> edges = new ArrayList<>();
26.   private int[] propertiesData;
27.   public GraphPanel() {
28.         setLayout(new BorderLayout());
29.         graph.getModel().beginUpdate();
30.         try {
31.               mxStylesheet styles = graph.getStylesheet();
32.               Map<String, Object> defVrt = styles.getDefaultVertexStyle();
33.               defVrt.put(mxConstants.STYLE_SHAPE,
mxConstants.SHAPE_ELLIPSE);
34.               styles.setDefaultVertexStyle(defVrt);
35.               Hashtable<String, Object> cstArr = new Hashtable<>();
36.               cstArr.put(mxConstants.STYLE_STROKECOLOR,
mxUtils.hexString(Color.BLACK));
37.               cstArr.put(mxConstants.STYLE_EDGE,
mxConstants.EDGESTYLE_TOPTOBOTTOM);
38.               styles.putCellStyle("customArrow", cstArr);
39.               Hashtable<String, Object> cstVrt = new Hashtable<>();
40.               cstVrt.put(mxConstants.STYLE_FILLCOLOR,
mxUtils.hexString(Color.LIGHT_GRAY));
41.               cstVrt.put(mxConstants.STYLE_STROKECOLOR,
mxUtils.hexString(Color.BLACK));
42.               cstVrt.put(mxConstants.STYLE_SHAPE,
mxConstants.SHAPE_ELLIPSE);
43.               cstVrt.put(mxConstants.STYLE_FONTCOLOR,
mxUtils.hexString(Color.WHITE));
44.               styles.putCellStyle("customVertex", cstVrt);
45.         } finally {
46.               graph.getModel().endUpdate();
47.         }
48.         graph.setCellsEditable(false);
49.         graph.setCellsResizable(false);
50.         graph.setCellsBendable(false);
51.         graph.setConnectableEdges(false);
52.         mxGraphComponent graphComponent = new mxGraphComponent(graph);
53.         add(graphComponent, BorderLayout.CENTER);
54.   }
55.   public void addVertex() {
56.         graph.getModel().beginUpdate();
57.         try {
58.               mxICell            vertex            =            (mxICell)
graph.insertVertex(graph.getDefaultParent(), null,
59.               GraphPanel.counter, 60, 60, 60, 60, "customVertex");
60.               GraphPanel.counter++;
61.               vertexes.add(vertex);
62.         } finally {
63.               graph.getModel().endUpdate();
64.         }
65.         repaint();
66.   }
67.   public void addEdge(int from, int to) {
68.         graph.getModel().beginUpdate();
69.         try {
70.               mxICell             edge             =             (mxICell)
graph.insertEdge(graph.getDefaultParent(), null,
```

```
71.               "", vertexes.get(from), vertexes.get(to), "customArrow");
72.               edges.add(edge);
73.         } finally {
74.               graph.getModel().endUpdate();
75.         }
76.         repaint();
77.   }
78.   public int[][] createTransitions() {
79.         int[][] transitions = new int[vertexes.size()][vertexes.size()];
80.         for (mxICell e : edges) {
81.               mxCell edge = (mxCell) e;
82.
      transitions[vertexes.indexOf(edge.getSource())][vertexes.indexOf(edge.ge
tTarget())] = 1;
83.         }
84.         return transitions;
85.   }
86.   public int[] getPropertiesData() {
87.         if (propertiesData == null) {
88.               propertiesData = new int[vertexes.size()];
89.               Arrays.fill(propertiesData, -1);
90.         }
91.         return propertiesData;
92.   }
93.   @Override
94.   public void update(Observable o, Object arg) {
95.         if (arg instanceof List) {
96.               @SuppressWarnings("unchecked")
97.               List<int[]> data = (List<int[]>) arg;
98.               Object[] edges = graph.getAllEdges(vertexes.toArray());
99.               for (Object edge : edges) {
100.                        graph.getModel().remove(edge);
101.              }
102.              this.edges.clear();
103.              for (int[] cur : data) {
104.                    addEdge(cur[0], cur[1]);
105.              }
106.              update();
107.        } else {
108.              propertiesData = (int[]) arg;
109.        }
110.  }
111.  public void update() {
112.        if (!vertexes.isEmpty()) {
113.              mxCompactTreeLayout layout = new mxCompactTreeLayout(graph,
false);
114.              layout.execute(null, vertexes.get(0));
115.              repaint();
116.        }
117.  }
118. }
1.  package gui;
2.  import com.thoughtworks.xstream.XStream;
3.  import sim.Library;
4.  import javax.swing.*;
5.  import java.awt.*;
6.  import java.awt.event.ActionEvent;
7.  import java.io.FileWriter;
8.  /**
9.   * Frame for displaying library data.
10.   */
11. public class LibraryFrame extends Frame {
12.   private static final long serialVersionUID = 5416000952627581896L;
```

```
13.   private final JTable table = new JTable();
14.   private final JButton ok = new JButton(new Ok(table));
15.   private final JButton cancel = new JButton(new Cancel());
16.   public LibraryFrame() {
17.         super("Library");
18.         setLayout(new BorderLayout());
19.         setResizable(false);
20.         setAlwaysOnTop(true);
21.         setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
22.         add(createContent(), BorderLayout.CENTER);
23.         init();
24.         pack();
25.         moveToScreenCenter();
26.   }
27.   private void init() {
28.         try {
29.               table.setModel(new Library());
30.         } catch (Exception e) {
31.               showError("Exception" + e.getMessage());
32.         }
33.   }
34.   private JPanel createContent() {
35.         JScrollPane pane = new JScrollPane(table);
36.         JPanel bottom = new JPanel(new GridLayout(1, 2, 50, 0));
37.         bottom.add(ok);
38.         bottom.add(cancel);
39.         bottom.setBorder(BorderFactory.createEmptyBorder(20, 50, 20, 50));
40.         JPanel content = new JPanel(new BorderLayout());
41.         content.add(pane, BorderLayout.CENTER);
42.         content.add(bottom, BorderLayout.SOUTH);
43.         return content;
44.   }
1.   package gui;
2.   import javax.swing.*;
3.   import javax.swing.table.AbstractTableModel;
4.   import javax.swing.table.TableColumnModel;
5.   import java.awt.*;
6.   import java.awt.event.ActionEvent;
7.   import java.util.ArrayList;
8.   import java.util.List;
9.   import java.util.Observer;
10.   /**
11.   * Frame for adding and removing edges in graph.
12.   */
13.  public class LinkerFrame extends Frame {
14.   private static final long serialVersionUID = 2071440340003245390L;
15.   private final JTable table = new JTable();
16.   private final JButton add = new JButton(new Add(table));
17.   private final JButton remove = new JButton(new Remove(table));
18.   private final JButton ok = new JButton(new Ok(table));
19.   private final JButton cancel = new JButton(new Cancel());
20.   private final Observer observer;
21.   public LinkerFrame(int[][] transtions, Observer o) {
22.         super("Links");
23.         observer = o;
24.         setLayout(new BorderLayout());
25.         setResizable(false);
26.         setAlwaysOnTop(true);
27.         setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
28.         add(createContent(), BorderLayout.CENTER);
29.         init(transtions);
30.         pack();
31.         moveToScreenCenter();
```

```
32.   }
33.   private void init(int[][] transtions) {
34.         table.setModel(new LinkerModel(transtions));
35.         String[] items = new String[transtions.length];
36.         for (int i = 0; i < items.length; i++) {
37.               items[i] = String.valueOf(i);
38.         }
39.         JComboBox<String> comboBox = new JComboBox<>(items);
40.         DefaultCellEditor editor = new DefaultCellEditor(comboBox);
41.         TableColumnModel tcm = table.getColumnModel();
42.         for (int i = 0; i < tcm.getColumnCount(); i++) {
43.               tcm.getColumn(i).setCellEditor(editor);
44.         }
45.   }
46.   private JPanel createContent() {
47.         JScrollPane pane = new JScrollPane(table);
48.         JPanel bottom = new JPanel(new GridLayout(2, 2, 50, 20));
49.         bottom.add(add);
50.         bottom.add(remove);
51.         bottom.add(ok);
52.         bottom.add(cancel);
53.         bottom.setBorder(BorderFactory.createEmptyBorder(20, 50, 20, 50));
54.         JPanel content = new JPanel(new BorderLayout());
55.         content.add(pane, BorderLayout.CENTER);
56.         content.add(bottom, BorderLayout.SOUTH);
57.         return content;
58.   }
59.   private class Add extends Action {
60.         private static final long serialVersionUID = 6237327684509614305L;
61.         private final JTable table;
62.         public Add(JTable table) {
63.               super("Add", "res\\add.png", "res\\add_big.png");
64.               this.table = table;
65.         }
66.         @Override
67.         public void actionPerformed(ActionEvent e) {
68.               LinkerModel model = (LinkerModel) table.getModel();
69.               model.addRow();
70.               model.fireTableRowsInserted(0, model.getRowCount());
71.         }
72.   }
73.   private class Remove extends Action {
74.         private static final long serialVersionUID = -8727965290316548686L;
75.         private final JTable table;
76.         public Remove(JTable table) {
77.               super("Remove", "res\\remove.png", "res\\remove_big.png");
78.               this.table = table;
79.         }
80.         @Override
81.         public void actionPerformed(ActionEvent e) {
82.               LinkerModel model = (LinkerModel) table.getModel();
83.               model.removeRow(table.getSelectedRow());
84.               model.fireTableRowsDeleted(0, model.getRowCount());
85.         }
86.   }
87.   private class Ok extends Action {
88.         private static final long serialVersionUID = -2799992289456640097L;
89.         private final JTable table;
90.         public Ok(JTable table) {
91.               super("OK", "res\\ok.png", "res\\ok_big.png");
92.               this.table = table;
93.         }
94.         @Override
```

```
95.         public void actionPerformed(ActionEvent e) {
96.               if (((LinkerModel) table.getModel()).isEmptyTransitions()) {
97.                     showWarning("Empty lines!");
98.               } else {
99.                     observer.update(null,                    ((LinkerModel)
table.getModel()).getData());
100.                          setVisible(false);
101.              }
102.        }
103.  }
104.  private class Cancel extends Action {
105.        private static final long serialVersionUID = -7291044240556796475L;
106.        public Cancel() {
107.              super("Cancel", "res\\cancel.png", "res\\cancel_big.png");
108.        }
109.        @Override
110.        public void actionPerformed(ActionEvent e) {
111.              setVisible(false);
112.        }
113.  }
114.  private class LinkerModel extends AbstractTableModel {
115.        private static final long serialVersionUID = 49018715810606502 1L;
116.        private final List<int[]> data = new ArrayList<>();
117.        public LinkerModel(int[][] transitions) {
118.              for (int i = 0; i < transitions.length; i++) {
119.                    for (int j = i; j < transitions[i].length; j++) {
120.                          if (transitions[i][j] == 1) {
121.                                data.add(new int[]{i, j});
122.                          }
123.                    }
124.              }
125.        }
126.        public List<int[]> getData() {
127.              return data;
128.        }
129.        public boolean isEmptyTransitions() {
130.              for (int[] i : data) {
131.                    if (i.length == 0) {
132.                          return true;
133.                    }
134.              }
135.              return false;
136.        }
137.        @Override
138.        public int getRowCount() {
139.              return data.size();
140.        }
141.        @Override
142.        public int getColumnCount() {
143.              return 2;
144.        }
145.        @Override
146.        public String getColumnName(int columnIndex) {
147.              if (columnIndex == 0) {
148.                    return "From";
149.              } else {
150.                    return "To";
151.              }
152.        }
153.        @Override
154.        public boolean isCellEditable(int rowIndex, int columnIndex) {
155.              return true;
156.        }
```

```
157.        @Override
158.        public Object getValueAt(int rowIndex, int columnIndex) {
159.              int[] row = data.get(rowIndex);
160.              if (row.length == 0) {
161.                    return "";
162.              }
163.              return row[columnIndex];
164.        }
165.        @Override
166.        public void setValueAt(Object value, int rowIndex, int columnIndex)
{
167.              if (value != null) {
168.                    int[] row = data.get(rowIndex);
169.                    if (row.length == 0) {
170.                          row = new int[2];
171.                    }
172.                    row[columnIndex] = Integer.parseInt((String) value);
173.                    data.set(rowIndex, row);
174.              }
175.        }
176.        public void addRow() {
177.              data.add(new int[]{});
178.        }
179.        public void removeRow(int index) {
180.              if ((index >= 0) && (index < getRowCount())) {
181.                    data.remove(index);
182.              }
183.        }
184.  }
185. }
```

```
1.   package gui;
2.   import com.thoughtworks.xstream.XStream;
3.   import sim.*;
4.   import javax.swing.*;
5.   import javax.swing.filechooser.FileNameExtensionFilter;
6.   import java.awt.*;
7.   import java.awt.event.ActionEvent;
8.   import java.io.File;
9.   import java.io.FileWriter;
10.   import java.util.ArrayList;
11.  import java.util.HashSet;
12.  import java.util.List;
13.  import java.util.Set;
14.  /**
15.   * Main frame for application.
16.   */
17.  public class MainFrame extends Frame {
18.   private static final long serialVersionUID = 8350407021970335634L;
19.   private final GraphPanel graph = new GraphPanel();
20.   private final Library library = new Library();
21.   private final JTabbedPane tabbed = new JTabbedPane();
22.   public MainFrame() {
23.         super("FPGA Sym");
24.         setLayout(new BorderLayout());
25.         setSize(Toolkit.getDefaultToolkit().getScreenSize());
26.         setExtendedState(java.awt.Frame.MAXIMIZED_BOTH);
27.         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
28.         setJMenuBar(createMenu());
29.         add(createToolBar(), BorderLayout.NORTH);
30.         add(createContent(), BorderLayout.CENTER);
31.   }
32.   public static void main(String[] args) {
```

```
33.         SwingUtilities.invokeLater(() -> {
34.               new MainFrame().setVisible(true);
35.               Thread.setDefaultUncaughtExceptionHandler((t, e) -> {
36.                     showError(e);
37.                     e.printStackTrace();
38.               });
39.         });
40.   }
41.   private JPanel createContent() {
42.         tabbed.addTab("Graph",    new    ImageIcon("res\\algo.png"),    new
JScrollPane(graph));
43.         tabbed.addTab("Gant", new ImageIcon("res\\gant.png"), null);
44.         JPanel content = new JPanel(new BorderLayout());
45.         content.add(tabbed, BorderLayout.CENTER);
46.         return content;
47.   }
48.   private JMenuBar createMenu() {
49.         JMenuBar menu = new JMenuBar();
50.         JMenu algorithm = new JMenu("Algorithm");
51.         algorithm.add(new Open());
52.         algorithm.addSeparator();
53.         algorithm.add(new Save());
54.         menu.add(algorithm);
55.         JMenu graph = new JMenu("Graph");
56.         graph.add(new AddVertex());
57.         graph.add(new AddEdge());
58.         graph.addSeparator();
59.         graph.add(new Properties());
60.         menu.add(graph);
61.         JMenu library = new JMenu("Library");
62.         library.add(new Edit());
63.         menu.add(library);
64.         JMenu calculations = new JMenu("Calculations");
65.         calculations.add(new Simulate());
66.         menu.add(calculations);
67.         menu.add(Box.createHorizontalGlue());
68.         JMenu help = new JMenu("?");
69.         help.add(new About());
70.         help.add(new Exit());
71.         menu.add(help);
72.         return menu;
73.   }
74.   private JToolBar createToolBar() {
75.         JToolBar toolBar = new JToolBar("Tools");
76.         toolBar.add(new Open());
77.         toolBar.add(new Save());
78.         toolBar.addSeparator();
79.         toolBar.add(new AddVertex());
80.         toolBar.add(new AddEdge());
81.         toolBar.add(new Properties());
82.         toolBar.addSeparator();
83.         toolBar.add(new Simulate());
84.         return toolBar;
85.   }
86.   @SuppressWarnings("unchecked")
87.   private List<Task>[] makeTaskLevels() {
88.         int[][] transitions = graph.createTransitions();
89.         int[] hwNumbers = graph.getPropertiesData();
90.         int tasksCounter = 0;
91.         int levelsCounter = 0;
92.         List<List<Task>> tasks = new ArrayList<>();
93.         List<Task> firstLevel = new ArrayList<>();
94.         firstLevel.add(new Task(hwNumbers[0]));
```

```
95.         tasksCounter++;
96.         tasks.add(firstLevel);
97.         levelsCounter++;
98.         while (tasksCounter != transitions.length) {
99.               List<Task> level = new ArrayList<>();
100.                    List<Task> prevLevel = tasks.get(levelsCounter - 1);
101.              Set<Integer> visited = new HashSet<>();
102.              for (Task prevLevelTask : prevLevel) {
103.                    int[]                transitionsLine                =
transitions[prevLevelTask.getId()];
104.                    for (int j = 0; j < transitionsLine.length; j++) {
105.                          boolean notConnectedOnCurrentLevel = true;
106.                          for (Task levelTask : level) {
107.                    notConnectedOnCurrentLevel                           &=
transitions[levelTask.getId()][j] != 1;
108.                          }
109.                          boolean isConnected = transitionsLine[j] == 1;
110.                          boolean notVisited = !visited.contains(j);
111.                          if     (isConnected     &&     notVisited     &&
notConnectedOnCurrentLevel) {
112.                                level.add(new Task(hwNumbers[j]));
113.                                tasksCounter++;
114.                                visited.add(j);
115.                          }
116.                    }
117.              }
118.              tasks.add(level);
119.              levelsCounter++;
120.        }
121.        List<Task>[] result = new ArrayList[tasks.size()];
122.        int i = 0;
123.        for (List<Task> lst : tasks) {
124.              result[i] = lst;
125.              i++;
126.        }
127.        return result;
128.  }
129.  private class Open extends Action {
130.        private static final long serialVersionUID = 8335479215032758045L;
131.        public Open() {
132.              super("Open", "res\\open.png", "res\\open_big.png");
133.        }
134.        @Override
135.        public void actionPerformed(ActionEvent e) {
136.              //TODO bug to clear all before opening
137.              JFileChooser chooser = new JFileChooser(".");
138.              chooser.setFileFilter(new FileNameExtensionFilter(
139.                          "eXtensible Markup Language files (XML)", "xml",
"XML"));
140.              chooser.setSelectedFile(new File("temp.xml"));
141.              int result = chooser.showOpenDialog(MainFrame.this);
142.              if (result == JFileChooser.APPROVE_OPTION) {
143.              Object[]        memento        =        (Object[])        new
XStream().fromXML(chooser.getSelectedFile());
144.                    int[][] trans = (int[][]) memento[0];
145.                    for (int[] tran : trans) {
146.                          graph.addVertex();
147.                    }
148.                    for (int i = 0; i < trans.length; i++) {
149.                          for (int j = i; j < trans[i].length; j++) {
150.                                if (trans[i][j] == 1) {
151.                                      graph.addEdge(i, j);
152.                                }
```

```
153.                         }
154.                   }
155.                   graph.update(null, memento[1]);
156.                   graph.update();
157.             }
158.       }
159.  }
160.  private class Save extends Action {
161.       private static final long serialVersionUID = 4266151357634163782L;
162.       public Save() {
163.             super("Save", "res\\save.png", "res\\save_big.png");
164.       }
165.       @Override
166.       public void actionPerformed(ActionEvent e) {
167.             JFileChooser chooser = new JFileChooser(".");
168.   chooser.setFileFilter(new   FileNameExtensionFilter("eXtensible   Markup
Language", "xml", "XML"));
169.             chooser.setSelectedFile(new File("temp.xml"));
170.             int result = chooser.showSaveDialog(MainFrame.this);
171.             if (result == JFileChooser.APPROVE_OPTION) {
172.                   int[][] trans = graph.createTransitions();
173.                   int[] prop = graph.getPropertiesData();
174.                   Object[] memento = new Object[2];
175.                   memento[0] = trans;
176.                   memento[1] = prop;
177.                   try        (FileWriter        fw        =        new
FileWriter(chooser.getSelectedFile())) {
178.                         new XStream().toXML(memento, fw);
179.                   } catch (Exception ex) {
180.                         showError("Exception " + ex.getMessage());
181.                   }
182.             }
183.       }
184.  }
185.  private class AddVertex extends Action {
186.       private static final long serialVersionUID = 2641072448265871581L;
187.       public AddVertex() {
188.             super("Add            vertex",            "res\\vertex.png",
"res\\vertex_big.png");
189.       }
190.       @Override
191.       public void actionPerformed(ActionEvent e) {
192.             graph.addVertex();
193.       }
194.  }
195.  private class AddEdge extends Action {
196.       private static final long serialVersionUID = 77073015103722950 44L;
197.       public AddEdge() {
198.             super("Add edges", "res\\edge.png", "res\\edge_big.png");
199.       }
200.       @Override
201.       public void actionPerformed(ActionEvent e) {
202.             LinkerFrame lf = new LinkerFrame(graph.createTransitions(),
graph);
203.             lf.setVisible(true);
204.       }
205.  }
206.  private class Properties extends Action {
207.       private static final long serialVersionUID = -8224726385156451096L;
208.       public Properties() {
209.             super("Properties", "res\\props.png", "res\\props_big.png");
210.       }
211.       @Override
```

```
212.        public void actionPerformed(ActionEvent e) {
213.  PropertiesFrame    pf    =    new    PropertiesFrame(library.getSize(),
graph.getPropertiesData(), graph);
214.              pf.setVisible(true);
215.        }
216.  }
217.  private class Edit extends Action {
218.        private static final long serialVersionUID = 6367966811361386786L;
219.        public Edit() {
220.              super("Edit", "res\\lib.png", "res\\lib_big.png");
221.        }
222.        @Override
223.        public void actionPerformed(ActionEvent e) {
224.              LibraryFrame lf = new LibraryFrame();
225.              lf.setVisible(true);
226.        }
227.  }
228.  private class Simulate extends Action {
229.        private static final long serialVersionUID = 7730347825572796898L;
230.        public Simulate() {
231.              super("Simulate", "res\\calc.png", "res\\calc_big.png");
232.        }
233.        @Override
234.        public void actionPerformed(ActionEvent e) {
235.              //TODO bug when empty, bug with scrollpane
236.              SettingsHolder   settingsHolder   =   new   SettingsHolder(new
File("settings.xml"));
237.        Simulator        simulator        =        new        Simulator(new
HardwareSystem(settingsHolder), settingsHolder);
238.              tabbed.setComponentAt(1,         new         JScrollPane(new
GantDiagramPanel(simulator.simulate(makeTaskLevels()))));
239.        }
240.  }
241.  private class About extends Action {
242.        private static final long serialVersionUID = -1814964833144105128L;
243.        public About() {
244.              super("About", "res\\info.png", "res\\info_big.png");
245.        }
246.        @Override
247.        public void actionPerformed(ActionEvent e) {
248.  showInfo("Written by Myroslav Rudnytskyi, Kyiv Politechnic Institute,
group IO-41m, 2015.");
249.        }
250.  }
251.  private class Exit extends Action {
252.        private static final long serialVersionUID = -1523012579322514186L;
253.        public Exit() {
254.              super("Exit", "res\\exit.png", "res\\exit_big.png");
255.        }
256.        @Override
257.        public void actionPerformed(ActionEvent e) {
258.              if (showQuestion("Do you want to exit application?")) {
259.                    System.exit(0);
260.              }
261.        }
262.  }
263. }
1.   package gui;
2.   import javax.swing.*;
3.   import javax.swing.table.AbstractTableModel;
4.   import javax.swing.table.TableColumnModel;
5.   import java.awt.*;
6.   import java.awt.event.ActionEvent;
```

```
7.   import java.util.Observer;
8.   /**
9.    * Frame for setting properties for vertexes in graph.
10.    */
11.  public class PropertiesFrame extends Frame {
12.   private static final long serialVersionUID = 5743609407800956841L;
13.   private final JTable table = new JTable();
14.   private final JButton ok = new JButton(new Ok(table));
15.   private final JButton cancel = new JButton(new Cancel());
16.   private final Observer observer;
17.   public PropertiesFrame(int tasksCount, int[] data, Observer o) {
18.         super("Properties");
19.         observer = o;
20.         setLayout(new BorderLayout());
21.         setResizable(false);
22.         setAlwaysOnTop(true);
23.         setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
24.         add(createContent(), BorderLayout.CENTER);
25.         init(tasksCount, data);
26.         pack();
27.         moveToScreenCenter();
28.   }
29.   private void init(int tasksCount, int[] data) {
30.         table.setModel(new PropertiesModel(data));
31.         String[] items = new String[tasksCount];
32.         for (int i = 0; i < items.length; i++) {
33.               items[i] = String.valueOf(i);
34.         }
35.         JComboBox<String> comboBox = new JComboBox<>(items);
36.         DefaultCellEditor editor = new DefaultCellEditor(comboBox);
37.         TableColumnModel tcm = table.getColumnModel();
38.         for (int i = 0; i < tcm.getColumnCount(); i++) {
39.               tcm.getColumn(i).setCellEditor(editor);
40.         }
41.   }
42.   private JPanel createContent() {
43.         JScrollPane pane = new JScrollPane(table);
44.         JPanel bottom = new JPanel(new GridLayout(1, 2, 50, 20));
45.         bottom.add(ok);
46.         bottom.add(cancel);
47.         bottom.setBorder(BorderFactory.createEmptyBorder(20, 50, 20, 50));
48.         JPanel content = new JPanel(new BorderLayout());
49.         content.add(pane, BorderLayout.CENTER);
50.         content.add(bottom, BorderLayout.SOUTH);
51.         return content;
52.   }
53.   private class Ok extends Action {
54.         private static final long serialVersionUID = -8517570522823621277L;
55.         private final JTable table;
56.         public Ok(JTable table) {
57.               super("OK", "res\\ok.png", "res\\ok_big.png");
58.               this.table = table;
59.         }
60.         @Override
61.         public void actionPerformed(ActionEvent e) {
62.               if                                          (((PropertiesModel)
table.getModel()).isEmptyProperties()) {
63.                     showWarning("Empty lines!");
64.               } else {
65.                     observer.update(null,               ((PropertiesModel)
table.getModel()).getData());
66.                     setVisible(false);
67.               }
```

```
68.         }
69.    }
70.    private class Cancel extends Action {
71.         private static final long serialVersionUID = -5138288617849048408L;
72.         public Cancel() {
73.               super("Cancel", "res\\cancel.png", "res\\cancel_big.png");
74.         }
75.         @Override
76.         public void actionPerformed(ActionEvent e) {
77.               setVisible(false);
78.         }
79.    }
80.    private class PropertiesModel extends AbstractTableModel {
81.         private static final long serialVersionUID = -3160038264112570310L;
82.         private final int[] data;
83.         public PropertiesModel(int[] data) {
84.               this.data = data;
85.         }
86.         public boolean isEmptyProperties() {
87.               for (int i : data) {
88.                     if (i == -1) {
89.                           return true;
90.                     }
91.               }
92.               return false;
93.         }
94.         public int[] getData() {
95.               return data;
96.         }
97.         @Override
98.         public int getRowCount() {
99.               return data.length;
100.              }
101.        @Override
102.        public int getColumnCount() {
103.              return 2;
104.        }
105.        @Override
106.        public String getColumnName(int columnIndex) {
107.              if (columnIndex == 0) {
108.                    return "Vertex";
109.              } else {
110.                    return "Hardware Task";
111.              }
112.        }
113.        @Override
114.        public boolean isCellEditable(int rowIndex, int columnIndex) {
115.              return columnIndex == 1;
116.        }
117.        @Override
1.    package gui;
2.    /**
3.     * Class represents command-line (text-based) version of Gant's diagram.
4.     */
5.    public class TimeTracks {
6.     private final StringBuilder[] data;
7.     public TimeTracks(int tracksCount) {
8.           data = new StringBuilder[tracksCount];
9.           for (int i = 0; i < data.length; i++) {
10.                data[i] = new StringBuilder();
11.          }
12.    }
13.    public void addWaiting(int track, int time) {
```

```
14.         addSymbol(track, time, ' ');
15.   }
16.   public void addSearchingAndLoading(int track, int time) {
17.         addSymbol(track, time, '#');
18.   }
19.   public void addLoadingLastWord(int track, int time) {
20.         addSymbol(track, time, '$');
21.   }
22.   public void addLoadingData(int track, int time) {
23.         addSymbol(track, time, '&');
24.   }
25.   public void addCounting(int track, int time) {
26.         addSymbol(track, time, '*');
27.   }
28.   private void addSymbol(int track, int time, char symbol) {
29.         requiredNotNegative(time);
30.         checkBounds(track);
31.         for (int i = 0; i < time; i++) {
32.               data[track].append(symbol);
33.         }
34.   }
35.   public void addWaitingToLongestLoading(int track) {
36.         checkBounds(track);
37.         int maxSize = 0;
38.         for (StringBuilder element : data) {
39.               int length = element.length();
40.               if ((length > 1) && (element.charAt(length - 1) == '&')) {
41.                     maxSize = Math.max(maxSize, length);
42.               }
43.         }
44.         int count = maxSize - data[track].length();
45.         for (int i = 0; i < count; i++) {
46.               data[track].append(' ');
47.         }
48.   }
49.   public int getMaxLength() {
50.         int maxSize = 0;
51.         for (StringBuilder track : data) {
52.               maxSize = Math.max(maxSize, track.length());
53.         }
54.         return maxSize;
55.   }
56.   public void addWaitingToLongest() {
57.         int maxSize = getMaxLength();
58.         if (maxSize == 0) return;
59.         for (StringBuilder track : data) {
60.               while (track.length() != maxSize) {
61.                     track.append(' ');
62.               }
63.         }
64.   }
65.   public int getLength(int track) {
66.         checkBounds(track);
67.         return data[track].length();
68.   }
69.   public void addWaitingToLongestCounting(int track) {
70.         checkBounds(track);
71.         int maxSize = 0;
72.         for (StringBuilder element : data) {
73.               int length = element.length();
74.               if ((length > 1) && (element.charAt(length - 1) == '*')) {
75.                     maxSize = Math.max(maxSize, length);
76.               }
```

```
77.         }
78.         int count = maxSize - data[track].length();
79.         for (int i = 0; i < count; i++) {
80.               data[track].append(' ');
81.         }
82.   }
83.   public int getTracksCount() {
84.         return data.length;
85.   }
86.   public String getTrack(int track) {
87.         checkBounds(track);
88.         return data[track].toString();
89.   }
90.   public int getWorkingTime() {
91.         int counter = 0;
92.         for (StringBuilder aData : data) {
93.               for (int i = 0; i < aData.length(); i++) {
94.                     if (aData.charAt(i) == '*') {
95.                           counter++;
96.                     }
97.               }
98.         }
99.         return counter;
100.        }
101.  public int getLoadingTime() {
102.        int counter = 0;
103.        for (StringBuilder aData : data) {
104.              for (int i = 0; i < aData.length(); i++) {
105.              if (aData.charAt(i) == '#' || aData.charAt(i) == '$' ||
aData.charAt(i) == '&') {
106.                          counter++;
107.                    }
108.              }
109.        }
110.        return counter;
111.  }
112.  @Override
113.  public String toString() {
114.        StringBuilder sb = new StringBuilder();
115.        for (int i = 0; i < data.length; i++) {
116.              sb.append(i);
117.              sb.append(data[i]);
118.              sb.append(System.lineSeparator());
119.        }
120.        return sb.toString();
121.  }
122.  private void requiredNotNegative(int i) {
123.        if (i < 0) {
124.              throw new IllegalArgumentException();
125.        }
126.  }
127.  private void checkBounds(int i) {
128.        requiredNotNegative(i);
129.        if (i >= data.length) {
130.              throw new IllegalArgumentException();
131.        }
132.  }
133. }
```

ДОДАТОК Ж

Товариство з обмеженою відповідальністю
"НАУКОВО-ВИРОБНИЧИЙ КОМПЛЕКС
"ГОЛОВНЕ ПІДПРИЄМСТВО ОБРОБКИ
ПОЛЬОТНОЇ ІНФОРМАЦІЇ
"АВІАЦІЙНІ ТЕХНОЛОГІЇ"
Код ЄДРПОУ 38826920
Адреса для листів:
Україна, 03058, м.Київ-58, а/с 68
тел./факс (044) 406-75-44
E-mail : hd_fdcp@ukr.net

АКТ

впровадження в розподілену систему оброблення та аналізу
польотної інформації в реальному часі
результатів дисертаційної роботи Клименко Ірини Анатоліївни
«Методи та засоби підвищення ефективності обробки інформації в
реконфігуровних комп'ютерних системах на базі ПЛІС»,
поданої на здобуття наукового ступеня доктора технічних наук
за спеціальністю 05.13.05 – комп'ютерні системи та компоненти

м. Київ «10» лютого 2016 р.

Комісія у складі голови – Мішаріної С.І.

та членів комісії – Косьянчука О.А.

– Кучеренка О.А.

постановила, що результати дисертаційної роботи Клименко І.А. «Методи та засоби підвищення ефективності обробки інформації в реконфігуровних комп'ютерних системах на базі ПЛІС» були впроваджені в розподілену систему оброблення та аналізу польотної інформації в реальному часі.

Новий спосіб організації віртуальної пам'яті даних, яка відрізняється від відомих багаторівневою багатофункціональною структурою для зберігання, пошуку і керування даними, впроваджений в комплекс програмно-апаратного забезпечення розподіленої системи оброблення та аналізу польотної інформації в реальному часі.

Спосіб організації віртуальної пам'яті даних використаний для створення модуля дворівневої кеш-пам'яті для локального зберігання даних і системи автоматичного керування даними на базі програмовних логічних інтегральних схем (ПЛІС). Застосування дворівневої кеш-пам'яті даних дозволяє зменшувати непродуктивні витрати часу в процесі

оброблення великих масивів даних. Реалізація автоматичного керування даними, що ґрунтується на застосуванні локальних керувальних засобів і швидкодійної асоціативної пам'яті керувальних даних, дозволило прискорити процеси обліку та пошуку даних в дворівневій пам'яті і зменшити апаратурні витрати для реалізації модуля пам'яті.

Використання розроблених апаратних засобів підвищило ефективність функціонування розподіленої системи оброблення та аналізу польотної інформації в реальному часі під час виконання завдань віддаленого оброблення великих масивів даних в режимі реального часу та зменшило її вартість.

Комісія відмічає теоретичний вклад Клименко І.А. у вирішенні проблеми підвищення ефективності оброблення інформації в розподілених комп'ютерних системах під час виконання завдань керування складними динамічними процесами та об'єктами в режимі реального часу.

Голова комісії _____ Мішаріна С.І.

Члени комісії _____ Косьянчук О.А.

_____ Кучеренко О.А.