МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЗАЛІЗНИЧНОГО ТРАНСПОРТУ

С. В. Лістровий, М. А. Мірошник, Л. А. Клименко

ТЕОРІЯ АВТОМАТИЧНОГО КЕРУВАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ І АВТОМАТИЗАЦІЯ ПРОЦЕСУ ПРИЙНЯТТЯ РІШЕННЯ

Навчальний посібник

Харків – 2019

УДК 004.415.03:[004.27:004.89], 681.3:324-326
Л 31

Рекомендовано вченою радою Українського державного університету залізничного транспорту як навчальний посібник (витяг з протоколу № 6 від 26 червня 2018 р.)

Рецензенти:
професори Г. Ф. Кривуля (ХНУРЕ),
О. О. Можаєв (ХНУВС)

Л 31 **Лістровий С. В., Мірошник М. А., Клименко Л. А.** Теорія автоматичного керування, штучний інтелект і автоматизація процесу прийняття рішення: Навч. посібник. – Харків: УкрДУЗТ, 2019. – 120 с., рис. 24, табл. 1.

ISBN

Викладено теоретичні основи автоматизації процесу прийняття рішення на базі паралельних обчислювальних систем. Велика увага приділяється теорії автоматичного управління у складних системах, питанню побудови циклічних паралельних обчислювальних систем для управління складними системами в реальному масштабі часу, процесу прийняття рішень та використанню штучного інтелекту. Розглянуто архітектури паралельних обчислювальних систем та формальні моделі, які виникають у процесі управління залізничним транспортом.

Навчальний посібник призначено для студентів денної та заочної форм навчання для освітніх програм 1-го та 2-го рівнів, що навчаються за спеціальністю 123 «Комп'ютерна інженерія», 126 «Інформаційні системи і технології», 151 «Автоматизація та комп’ютерно-інтегровані технології», 273 «Залізничний транспорт», а також для інших спеціальностей відповідних напрямків. Навчальний посібник містить спеціальний та додатковий матеріал з дисциплін «Теорія автоматичного керування», «Теорія автоматичного керування та штучний інтелект», «Комп'ютерні мережі, Internet та хмарні сервіси», «Паралельні та розподілені обчислення та Cloud технології», «Математичні методи та моделі в нових інформаційних технологіях», «Апаратне і програмне забезпечення комп'ютерних систем загального і спеціального призначення» та може застосовуватися при виконанні курсових і розрахунково-графічних робіт.

Навчальний посібник призначений для студентів закладів вищої освіти і фахівців у галузі комп'ютерних систем та програмування.

УДК 004.415.03:[004.27:004.89], 681.3:324-326

ISBN

Навчальний посібник

Лістровий Сергій Володимирович,
Мірошник Марина Анатоліївна**,**
Клименко Любов Анатоліївна

ТЕОРІЯ АВТОМАТИЧНОГО КЕРУВАННЯ, ШТУЧНИЙ ІНТЕЛЕКТ І АВТОМАТИЗАЦІЯ ПРОЦЕСУ ПРИЙНЯТТЯ РІШЕННЯ

Відповідальний за випуск Мірошник М. А.

Редактор Еткало О. О.

Підписано до друку 11.06.18 р.

Формат паперу 60х84 1/16. Папір писальний.

Умовн.-друк.арк. 6,5. Тираж 50. Замовлення №

Видавець та виготовлювач Український державний університет залізничного транспорту,
61050, Харків-50, майдан Фейєрбаха, 7.
Свідоцтво суб’єкта видавничої справи ДК № 6100 від 21.03.2018 р.

ЗМІСТ

ПЕРЕДМОВА

У навчальному посібнику викладено теоретичні основи автоматизації процесу прийняття рішення та штучного інтелекту на базі паралельних обчислювальних систем. Велика увага приділяється питанню побудови циклічних паралельних обчислювальних систем для керування складними системами в реальному масштабі часу та штучному інтелекту. Розглянуто архітектури паралельних обчислювальних систем та формальні моделі, які виникають у процесі управління залізничним транспортом.

Навчальний посібник призначено для студентів денної та заочної форм навчання для освітніх програм 1-го та 2-го рівнів, що навчаються за спеціальністю 123 «Комп'ютерна інженерія », 126 «Інформаційні системи і технології», 151 «Автоматизація та комп’ютерно-інтегровані технології», 273 «Залізничний транспорт», а також для інших спеціальностей відповідних напрямків.

Навчальний посібник містить спеціальний та додатковий матеріал з дисциплін «Теорія автоматичного керування», «Теорія автоматичного керування та штучний інтелект», «Комп'ютерні мережі, Internet та хмарні сервіси», «Паралельні та розподілені обчислення та Cloud технології», «Математичні методи та моделі в нових інформаційних технологіях», «Апаратне і програмне забезпечення комп'ютерних систем загального і спеціального призначення» та може застосовуватися при виконанні курсових і розрахунково-графічних робіт.

Навчальний посібник призначений для студентів закладів вищої освіти і фахівців у галузі комп'ютерних систем та програмування.

У розд. 1 розглянуто питання автоматизації процесу прийняття рішення, вимоги до обчислювальних систем щодо прийняття рішення та використання штучного інтелекту у складних системах керування, автоматизовану систему управління та рішення управлінця, логічну послідовність процесу вироблення рішення, типи задач, що розв’язуються в процесі прийняття рішень та в системах штучного інтелекту, та рівні архітектури інтерактивних систем і формальні моделі в задачах

прийняття рішень та системах штучного інтелекту. Велику увагу приділено організації паралельних обчислень при розв'язанні задач у телекомунікаційних системах і мережах.

У розд. 2 доступно викладено основні формальні моделі задач прийняття рішень та використання систем штучного інтелекту у складних системах керування, розглянуто динамічне програмування у задачах прийняття рішень та системах штучного інтелекту, проаналізовано варіаційні задачі керування для автоматизації процесу прийняття рішень, задачі варіаційного числення та оптимального керування в теорії прийняття рішень і системах штучного інтелекту та задачі дискретного програмування й оптимізаційні задачі на графах для систем прийняття рішень та штучного інтелекту, зроблено постановку екстремальних задач на графах для систем прийняття рішень та штучного інтелекту.

У розд. 3 розглянуто проблематику розбудови паралельних обчислювальних систем та основні шляхи їх розвитку, паралельні обчислювальні системи й алгоритми прийняття рішень у системах штучного інтелекту, досліджено архітектуру паралельних обчислювальних систем циклічного типу для систем штучного інтелекту.

У розд. 4 розглянуто систолічні матричні процесори для систем прийняття рішень та штучного інтелекту, досліджено паралельні обчислювальні системи на базі комп'ютерних технологій у системах прийняття рішень та штучного інтелекту, хвильові матричні процесори в системах прийняття рішень та штучного інтелекту.

Для нових рангових методів у задачах дискретної оптимізації і теорії графів, які в сучасній літературі розкидані по окремих статтях і монографіях, наведено систематичний виклад, надано багато прикладів.

Кожний розділ має приклади практичних завдань та контрольні запитання.

ОСНОВНІ СКОРОЧЕННЯ

IMM IDEAL (Ideal multifunction Machine) – багатофункціональна машина
АСУ – автоматизована система управління
БВН – блок вибору напрямку передачі інформації
ГЗ – граф залежності
ГПД – граф потоку даних
ЕОМ – електронно-обчислювальна машина
ЕРС – електроруйнівна сила
ЗПР – задача прийняття рішення
ІМ – ієрархічна пам'ять
К – комутатор
ЛПР – людина або група осіб, які приймають рішення
МВВ – модуль управління введення-виведення
МОП – модулі оперативної пам'яті
ОМ – обчислювальні модулі
ОС – операційна система
ПВВ – прилад введення-виведення
ПОС – паралельна обчислювальна система
ПД – пам'ять даних
ПЕ – процесорні елементи
ПК – пам'ять команд
ПУВВ – пристрій управління введення і виведення
ПрД – процесор даних
ПЗЗ – процес запису вхідної інформації в регістрі пам'яті
ПОС – паралельна обчислювальна система
ПП – периферійні прилади
ПУП – пристрій управління перенумерацією
ПрК – процесор команд
ПВІС – понад велика інтенгральна схема
СПР – система прийняття рішень
ЦОС – цифрова́ обробка сигна́лів

ВСТУП

Питання автоматизації систем керування залізничним транспортом належать до числа найбільш складних, дослідження яких потребує детальної оцінки як якісних, так і кількісних сторін, що характеризують процеси керування. При розробленні глобальної автоматичної системи керування залізничним транспортом виникає необхідність вирішення комплексу проблем, пов'язаних із забезпеченням ефективних пасажирських і вантажних перевезень та забезпеченням надійності функціонування систем залізничного транспорту, що потребує обробки великих масивів інформації в системі.

Виникає протиріччя між безупинно зростаючим обсягом вхідної збірної інформації і кількістю необхідних для її обробки обчислень, з одного боку, і часом, який зменшується і який мають органи керування для здійснення цих трудомістких процесів, з другого. Тому для вирішення цього протиріччя з метою підвищення оперативності і якості керування в багатьох зарубіжних країнах ведуться роботи, пов'язані з упровадженням у процеси керування математичних засобів швидкодіючої обчислювальної техніки, що працює на їх основі.

Сучасний рівень розвитку обчислювальної техніки і засобів передачі інформації відкриває широкі можливості з розроблення і впровадження технічної бази автоматизації. Наявні ж математичні засоби дають змогу формалізувати основні завдання керування.

Аналіз сучасних розподілених систем показує, що зросла роль часових факторів. Час, що система витрачає на доведення інформації про стан керованого процесу до пунктів обробки і прийняття рішення на основі отриманої інформації і доведення прийнятого рішення до виконавчих органів повинен зменшуватися.

Сучасні складні системи характеризуються винятковою складністю умов, у яких здійснюється керування. До них можна віднести значний обсяг завдань, що виникають раптово; жорсткий ліміт часу, відведеного на прийняття (уточнення) рішення щодо їх виконання; велику інтенсивність інформаційних потоків між різними ланками керування; високий динамізм зміни

обстановки; обмеженість ресурсу, призначеного для розв’язання задач. Існують об'єкти, у яких час доведення інформації про стан керованого процесу, до пунктів становить одиниці секунд. У таких системах час є найважливішим параметром, бо дуже часто потребує формування керованих дій у реальному часі.

Усі ланки системи перебувають під фізичними впливами зовнішнього середовища, що призводять до зміни стану системи. Крім зовнішніх фізичних впливів у системі, що призводять до погіршення її стану, існують внутрішні фізичні зв'язки, які можуть поліпшити її стан за рахунок засобів відновлення.

Загалом всі ланки системи з'єднані прямими і зворотними інформаційними зв'язками, призначеними для передачі команд, донесень про їх виконання і стан керованих об'єктів. Зараз з'являються системи, які належать до класу динамічних інформаційних недетермінованих систем. Їх динамічність полягає як у змінах структури системи, так і її параметрів під впливом зовнішніх умов. Функціонування таких систем здійснюється за допомогою передачі інформації, кількість якої не впливає однозначно на результат керування.

Для розв’язання кожної задачі потрібен певний обсяг інформації. Збільшення або зменшення кількості даних не призводить до однозначних змін ефективності прийнятих рішень і витрат часу. Механізм дії цього закону диктує необхідність конкретного вирішення різних питань удосконалення техніки і технології. Ефективність у мережах залежить від оперативності розв’язання задач динамічного управління потоками інформації, математичними моделями яких є широкий клас задач лінійного булевого програмування, що належать до NP-повних задач, і класу задач нелінійного булевого програмування, для яких ефективні методи розв’язання практично відсутні. Крім того, широкий клас задач побудови та синтезу інтелектуальних мереж, а також їх діагностики теж формалізується на основі зазначених математичних моделей. Тому є актуальним розроблення методів і архітектури обчислювальних систем, що дають змогу з єдиних позицій розв’язувати зазначені класи задач і з необхідною оперативністю забезпечувати динамічне управління потоками інформації в мережах.

Розділ 1. Автоматизація процесу прийняття рішення та побудова систем штучного інтелекту

1.1. Вимоги до обчислювальних систем щодо прийняття рішення та використання штучного інтелекту у складних системах управління

Розроблення глобальної автоматизованої системи управління (АСУ) залізничним транспортом потребує створення проблемно-орієнтованих паралельних обчислювальних систем (ПОС). Особливо це важливо для забезпечення надійності та безаварійності функціонування системи, що породжує самостійну групу проблем, пов'язану з організацією пам'яті процесорів, а також розробленням паралельних алгоритмів, що дають змогу у реальному масштабі часу здійснювати розпізнавання імовірних відмов у системі. Будь-яка підсистема управління являє собою, як правило, обчислювальний комплекс.

Вона вирішує такі завдання:

- збір, первинна обробка та коригування даних;
- розрахунок усіх операцій, пов’язаних з прийняттям рішень;
- коригування даних з урахуванням виконаних операцій;
- планування відновлення найбільш важливих технічних об'єктів.

Найбільш значними проблемами при створенні будь-якого варіанта такої системи є:

- вибір характеристик ПОС, забезпечення її живучості;
- аналіз інформації та прийняття рішень про загальну обстановку;
- забезпечення контролю функціонування системи.

Важливим напрямком при створенні підсистем управління є розроблення їх математичного забезпечення, як показано на рис. 1.1. Тому однією з найважливіших вимог до ПОС є спрощення програмних засобів та програмування як за рахунок вибору архітектури ПОС, так і за рахунок ускладнення самих апаратних засобів.

Слід зазначити, що якщо управління діями у системі повинні йти у масштабі реального часу, то саме обґрунтування рішень на управління здійснюється у прискореному масштабі часу.

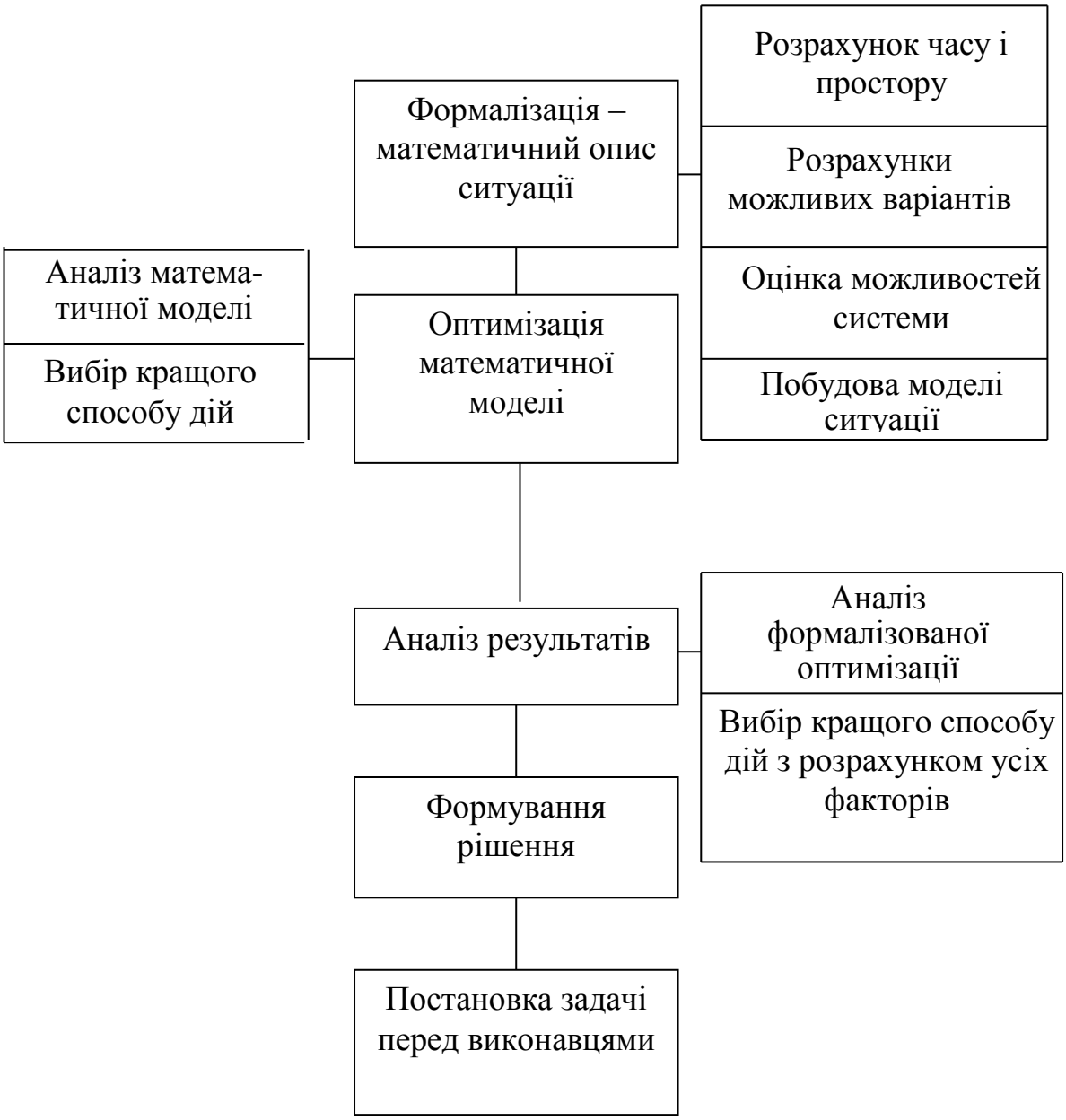

Рис. 1.1. Етапи розроблення математичного забезпечення

1.2. Автоматизована система управління та рішення управління

У кібернетичній системі розрізняють такі елементи: управляючий прилад; об'єкт управління; лінія зв'язку між ними, які зображено на рис. 1.2.

При цьому процес управління супроводжується постійним кругообігом інформації від управляючого органа до об'єкта управління та назад.

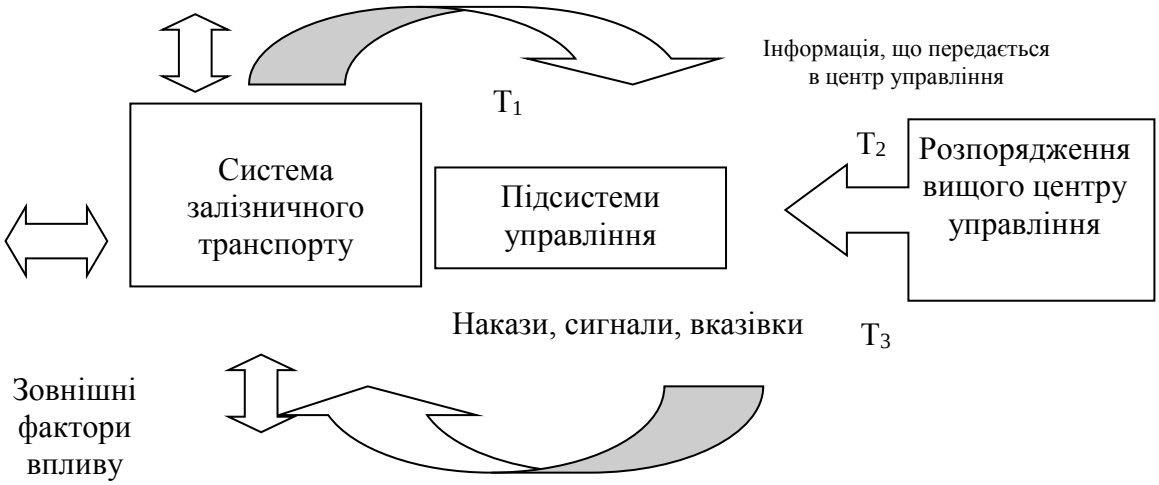

Рис. 1.2. Принципова схема системи управління

У кожній системі управління відбуваються такі самі процеси передачі інформації каналами зв'язку й обробки інформації, як і в управляючому обладнанні. Цикл управління також може бути охарактеризований часом $\mathbf{T_1}$, $\mathbf{T_2}$, $\mathbf{T_3}$, а тривалість виконання поставленої задачі для транспортної системи – часом $\mathbf{T_{дійсн}}$.

Щоб у кожному конкретному випадку можна було поставити числові вимоги до оперативності управління, потрібен уважний аналіз, якому може сприяти поняття критичного часу.

Критичний час $\mathbf{T_{крит}}$ може бути визначений як час, після закінчення якого дії управління не приведуть до поставленої мети взагалі або до тієї ефективності, що очікувалася і планувалася.

Величина критичного часу, так само як і час дії управління, тісно пов'язана з характером ситуацій, які виникають. З метою управління в конкретних випадках вона може набувати найрізноманітніших значень.

Управляти оперативно означає домагатися, щоб сума часу, витраченого на цикл управління $\mathbf{T_{упр}}$, і часу, необхідного для ефективного функціонування транспортної системи $\mathbf{T_{дійсн}}$, була менша, ніж критичний час, і повинна дотримуватися нерівність $\mathbf{T_{упр} + T_{дійсн} < T_{крит}}$.

Отже, умова оперативності управління може бути записана так:

$$\mathbf{T_{упр} < T_{крит} - T_{дійсн}}.$$

Нагадаємо, що час циклу управління у свою чергу є сумою часу приймання, обробки і передавання інформації, тобто

$$T_{УПР} = T_1 + T_2 + T_3.$$

Скорочення часу управління залежить від конкретних умов. В одних випадках час управління можна скоротити прискоренням приймання і передавання інформації (зменшується T_1 і T_3), в інших випадках — прискоренням процесу прийняття рішення (зменшиться T_2), а найчастіше — за рахунок хоча б невеликої економії, але в кожному процесі.

Важливим і перспективним напрямком для скорочення часу T_2 на обробку інформації (на прийняття рішень) є впровадження математичних засобів, швидкодіючої обчислювальної техніки, заздалегідь заготовлених лінійок, таблиць, графіків і номограм.

Практичне вирішення питання про те, у яких саме органах управління і для яких видів операцій у системі потрібна швидкодіюча обчислювальна техніка, а де можна обійтися іншими засобами, знову ж таки пов'язане з поняттям критичного часу.

Система управління містить у собі:

- збір інформації, її передавання та концентрування у певних місцях;
- обробку інформації та підготовку даних для прийняття рішення;
- контроль виконання рішення та коригування виконання рішення.

Автоматизація вироблення рішення означає перекладення функцій підготовки рішення на обчислювальну техніку. Для того, щоб рішення вироблялося автоматично, необхідно мати обчислювальні системи, які могли б:

- автоматично збирати всю інформацію про обстановку;
- перетворювати її у форму, придатну для роботи системи в цілому;
- аналізувати, класифікувати та узагальнювати прийняту інформацію;
- виробляти рішення за будь-яких обставин;
- виробляти управляючі сигнали;
- стежити за виконанням прийнятих рішень.

Найбільше значення при автоматизації процесу управління має застосування операційної системи (ОС) для реалізації математичних моделей. Саме етап вироблення рішень впливає найбільшою мірою на якість управління системою залізничного транспорту, оскільки від того, наскільки обґрунтованим буде рішення, залежить успіх її функціонування.

1.3. Логічна послідовність процесу вироблення рішення

Після виявлення задачі, яка виникла під впливом зовнішніх факторів або рішення керівної ланки управління, починається усвідомлення поставленої задачі, після цього оцінюють обстановку та роблять висновки до неї. На основі виявленої задачі виробляють план дій. План лягає в основу майбутнього рішення:

1) **усвідомлення поставленої задачі** – з'ясування плану рішення, деталізація своєї задачі, визначення місця та ролі підрозділів у виконанні операцій;

2) **оцінка обставин** – це розуміння об'єктивних умов майбутньої ситуації та знаходження можливих варіантів виконання поставленої задачі. Вона передбачає:

- оцінку обставин;
- оцінку своїх можливостей;

3) **вироблення послідовності виконання майбутніх дій.** План може містити один або декілька варіантів;

4) **розрахунок та вибір кращого варіанта.** Розрахунок розбивають на групи:

- розрахунок часу (дає змогу розподілити загальний ресурс часу за необхідними до виконання задачами);
- розрахунок простору (оцінюється просторовий розмах операцій);
- розрахунок технічних можливостей (оцінка потужності впливу ситуації);

5) **формулювання рішення** – оформлення рішення у вигляді документів. Зручна форма для використання у прийнятті рішення.

1.4. Типи задач, що розв'язуються в процесі прийняття рішень та в системах штучного інтелекту

Практикою управління встановлено, що процес вироблення рішень складається з таких складових компонентів: виявлення поставленої задачі, оцінка обстановки, вироблення рішення, формалізація предметної області, оптимізація математичної моделі, аналіз результатів, формулювання рішення, постановка задачі перед виконавцями (рис. 1.3).

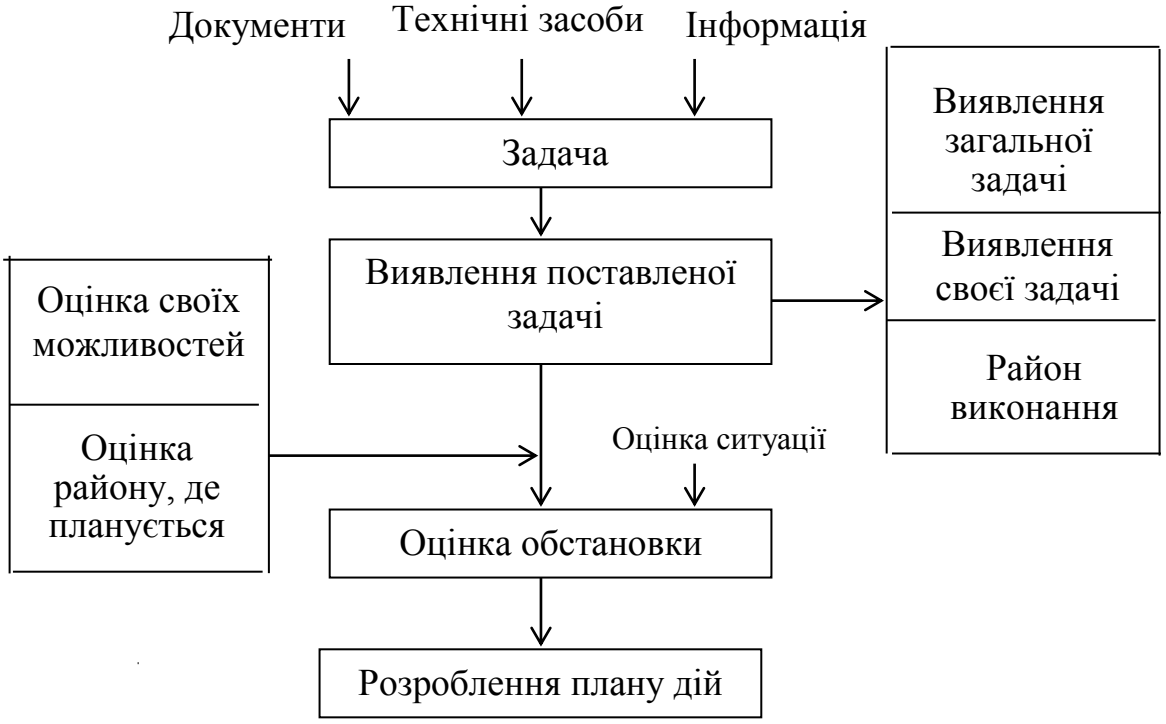

Рис. 1.3. Процес вироблення рішення

Закономірності перебігу процесів управління залізничним транспортом у сучасних умовах надзвичайно складні. У їх основі лежать кількісні відношення, що потребують застосування математики на підставі оптимальних рішень. Однак математика не може застосовуватися безпосередньо до тієї або іншої операції в системі, а лише до деякого математичного опису, тобто до математичної моделі. Шляхом аналізу моделі надається можливість вибрати один із засобів дій для досягнення найкращого результату. Такі моделі називають оптимізаційними і побудова їх становить один з етапів дослідження операцій. Математична

модель дій (операцій) установлює відповідність між значеннями змінних, що управляються, і змінних, що не управляються, і визначає результати цих дій у процесі управління.

У загальному вигляді математична модель операції може бути подана як

$$W = f(x_1,\ldots,x_m; y_1,\ldots,y_n),$$

$$q_k(x_1,\ldots,x_m; y_1,\ldots,y_n) \le 0, \quad k = (\overline{1,p}),$$

де W – критерій ефективності операції;

x_i $(i = \overline{1,m})$ – змінні, що управляються;

y_i $(i = \overline{1,n})$ – змінні або випадкові впливи, що не управляються;

$q_k (k = \overline{1,p})$ – функції, що відображають обмеження.

Обмеження істотно збільшують складність моделі. У цих випадках, навіть якщо критерій ефективності являє собою функцію, що диференціюється, максимум не може бути визначений за допомогою класичних методів математичного аналізу, якщо він не лежить усередині області, окресленої певною сукупністю обмежень. У тих випадках, коли критерій ефективності взагалі не можна диференціювати або кількість змінних дуже велика (значно перевищує обчислювальні можливості управляючого пристрою або системи), знайти оптимальний розв’язок можна тільки за допомогою методів математичного програмування. Класифікація математичних моделей може бути проведена з урахуванням математичних засобів, до яких слід віднести [2] методи варіаційного аналізу, способи розв’язання задач комбінаторної і дискретної оптимізації, алгоритми теорії ігор і теорії статистичних рішень. Основним етапом прийняття рішення є його обґрунтування на основі перерахованих математичних методів, що приводить до висновку, що для автоматизації необхідні ПОС, які дають змогу розв’язувати в масштабі реального часу оптимізаційні задачі великої вимірності, зумовлені великою масштабністю технологічних процесів під час управління залізничним транспортом [11] і широким спектром задач, які доводиться розв’язувати для їх оптимального планування.

Слід пам'ятати, що необхідність розв’язання оптимізаційних задач виникає практично на всіх етапах прийняття рішень: при постановці задачі, при побудові самої моделі та її оптимізації, при експертних оцінках отриманих рішень. При цьому оперативність прийняття рішення істотно залежить від автоматизації процесу розв’язання оптимізаційних задач і потребує організації обчислень практично в прискореному масштабі часу.

1.5. Рівні архітектури інтерактивних систем і формальні моделі в задачах прийняття рішень та системах штучного інтелекту

Розглянемо коротко особливості різноманітних рівнів архітектури інтерактивних систем прийняття рішень (СПР), не претендуючи на узагальнення.

1. Цільовий рівень. На цьому найзагальнішому рівні визначається мета, якій підпорядкована система, здійснюється взаємозв'язок понятійної бази (термінології, тезаурусу), мети і задач прийняття рішень.

2. Рівень постановок задач прийняття рішень (ЗПР). З можливих постановок ЗПР відзначимо такі основні постановки:

- лінійне впорядкування альтернативи (об'єктів, варіантів);
- виділення кращої альтернативи;
- виділення впорядкованої підмножини кращих альтернатив;
- часткове впорядкування альтернативи;
- впорядковане розбиття альтернатив (групове упорядкування);
- впорядковане розбиття альтернатив (класифікація).

У рамках окремих типів задач можливе введення додаткових умов і обмежень, що призводить до додаткового роздроблення і виділення підтипів. Зазначимо, що перераховані постановки тісно взаємопов‘язані між собою. Так, лінійне упорядкування альтернатив водночас автоматично задає і кращу альтернативу, упорядковане розбиття породжує підмножину кращих альтернатив. Такий взаємозв'язок дасть можливість використовувати проміжні результати і проміжні структури, отримані при розв’язанні однієї ЗПР для розв’язання інших.

3. Рівень процедур. На цьому рівні підключаються загальні процедури розв’язання задач, обробки результатів

вимірювань, характеристик альтернатив, отримання й обробки експертних оцінок, організації діалогу з експертами та людиною або групою осіб, які приймають рішення (ЛПР).

Як основні процедури порівняння та оцінювання альтернатив традиційно використовуються парні порівняння [4], що дає змогу спиратися на добре розроблений апарат теорії бінарних відношень. Крім того, використовуються й інші можливі засоби оцінювання та порівняння:

- множинні порівняння;
- регулювання;
- переупорядкування;
- вектори переваг;
- упорядковані розбиття;
- класифікація.

4. Рівень формальних моделей. Не визначаючи досить точного поняття моделі рішення ЗПР, зазначимо, що серед екстремальних моделей, як показує практика, є моделі дискретної оптимізації. Такі моделі можуть виникати не тільки як засіб формалізації вхідних задач, але й бути в рамках різноманітних процедур. Наприклад, при формуванні груп експертів, формуванні питань або ж обробці відповідей ЛПР. У тих випадках, коли метою порівняння альтернатив є виявлення факту (і ступеня) переваги однієї альтернативи над іншою, для опису структури, яка виникає, слід скористатися апаратом теорії орієнтованих графів. Якщо ж порівняння проводяться з метою виявлення факту (і ступеня) схожості альтернатив, то для опису відповідної структури схожості (відмінності) може бути використаний апарат теорії неорієнтованих графів. І в тому, і в іншому випадку апроксимація побудованої структури може проводитися на базі підбору в тому або іншому сенсі найкращого графа з будь-якого заданого класу. Формалізація проблеми обробки структури переваг призводить до різноманітних комбінаційно-оптимізаційних задач на графах.

5. Алгоритмічний рівень. Зазначимо, що навіть для однієї конкретної моделі може існувати ціла низка алгоритмів розв’язання, що мають цілком різноманітні характеристики (наприклад, обчислювальну складність, ресурси, що використовуються, точність та ін.). Пошук оптимального алгоритму в сенсі тієї або іншої моделі рішення нерідко буває пов’язаний з

необхідністю вирішення проблеми повного перебору всіх можливих варіантів. Для цього можуть бути використані:

- різноманітні схеми і варіанти засобів віток і кордонів;
- методи динамічного програмування;
- методи цілочислового (зокрема булевого) лінійного програмування;
- алгоритми послідовного аналізу варіантів;
- методи типу вектора спаду;
- алгоритми локальної оптимізації;
- різні евристичні схеми.

Вибір конкретного алгоритму залежить, звичайно, як від типу задач, які розв’язує система, так і від моделей, що використовуються, а також наявних ресурсів.

6. Програмний рівень. Призначення цього рівня досить очевидне. При цьому слід розуміти, що один і той самий конкретний алгоритм може припускати цілком різноманітні програмні реалізації.

7. Рівень ресурсів. На цьому рівні як ресурс виступають:

- обчислювальні ресурси (необхідний час роботи і пам'ять обчислювальних систем);
- витрати праці експертів;
- витрати праці ЛПР;
- витрати на вимірювання характеристик альтернатив;
- витрати на організацію експертизи і діалогів з експертами й ЛПР.

При побудові процедур прийняття рішення перераховані ресурси виступають як імітовані фактори. Особливо жорсткі вимоги при створенні автоматизованих систем прийняття рішень висуваються до обчислювальних систем, на базі яких вони будуються, з погляду підвищення їх швидкодії й обсягів пам'яті.

Контрольні питання

1. Пояснити структуру прийняття рішення.
2. Як працює система прийняття рішення?
3. Пояснити логічну послідовність прийняття рішення.
4. Які типи задач розв’язуються у процесі прийняття рішення?
5. Як можна формалізувати операції?
6. Які існують рівні інтерактивних систем у задачах прийняття рішень?

Розділ 2. Формальні моделі задач прийняття рішення та використання систем штучного інтелекту у складних системах управління

2.1. Динамічне програмування у задачах прийняття рішень та системах штучного інтелекту

Булеве програмування є формальною моделлю широкого класу задач теорії прийняття рішень, пов'язаних з виділенням неупорядкованої множини кращих об'єктів.

Найпростіша модель такого типу має вигляд

$$L = \sum_{j=1}^{n} \alpha_j \cdot x_j \to \max \qquad (2.1)$$

при обмеженнях

$$\sum_{j=1}^{n} \beta_{ij} \cdot x_j \le b_i, \qquad (2.2)$$

$$x_j \in \{0,1\}; \quad i = \overline{(1,m)}; \quad j = \overline{(1,n)}. \qquad (2.3)$$

У багатьох практичних ситуаціях у процесі прийняття рішення потрібно не тільки відібрати об'єкти, але й вибрати для кожного відібраного об'єкта варіант реалізації, що характеризується індивідуальними показниками ефективності і ресурсними обмеженнями. Це призводить до моделі сходового типу:

$$\sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_{ij} \cdot x_{ij} \to \max \qquad (2.4)$$

при обмеженнях

$$\sum_{i=1}^{m} \sum_{j=1}^{n} \beta_{ij} x_{ij} \le b, \qquad (2.5)$$

$$\sum_{j=1}^{c_i} x_{ij} \le 1, \qquad (2.6)$$

$$x_{ij} \in \{0,1\};\ \alpha_{ij} \ge 0;\ \beta_{ij} \ge 0; b \ge 0;\ i = \overline{(1,m)};\ j = \overline{(1,n)}. \qquad (2.7)$$

Динамічне програмування основане на принципі оптимальності. Задачі цілочислового динамічного програмування у загальному випадку можна подати у такому вигляді:

$$f(x) = f_1(x_1) \top f_2(x_2) \top \ldots f_n(x_n) \top \to \max \qquad (2.8)$$

при обмеженнях

$$\sum \varphi_j(x_j) \le b;\ x_j \in D_j.;\quad j = \overline{(1,n)}, \qquad (2.9)$$

де b – ціле число;

$\top$ – символ додавання, при якому функція f(x) сепарабельно-адитивна або символ множення, при якому функція f(x) сепарабельно-мультиплікативна;

$\varphi_j(x_j)$ – функції, що зберігають цілочисельність;

D_i – деяка множина цілих чисел.

Практично будь-яка задача динамічного програмування може бути зведена до задачі визначення найкоротшого шляху на графі G(V, E), у якому t_{ij} відповідає часу передачі одиниці потоку з вершини i у вершину j, тобто величина потоку по ребру задовольняє умову $0 \le x_{ij} \le 1$, і задача визначення найкоротшого шляху в графі G від вершини i у вершину j може бути записана таким чином:

$$\sum_i \sum_j t_{ij} x_{ij} \to \min \qquad (2.10)$$

при обмеженнях

$$\sum_j x_{ij} \le 1; \qquad (2.11)$$

$$\sum x_{jt} \geq 1; \qquad (2.12)$$

$$\sum_j x_{ij} - \sum_j x_{ij} = 0\ ,\ \ i \neq s, i \neq t\ ; \qquad (2.13)$$

$$x_{ij} \geq 0\,. \qquad (2.14)$$

Слід зазначити, що умова (2.13) означає збереження потоку в мережі, а нерівності (2.11) і (2.12) при підстановці відповідних оптимальних рішень переходять у рівності.

2.2. Варіаційні задачі управління для автоматизації процесу прийняття рішень

Оптимальне управління може розглядатися як узагальнене варіаційне обчислення. Усі задачі варіаційного обчислення пов'язані з максимізацією або мінімізацією інтегралів

$$P = \int_0^{t_k} G[y_1(x), y_2(x),\ldots,y_n(x); \dot{y}_1(x), \dot{y}_2(x),\ldots,\dot{y}_n(x), k]\,dx. \qquad (2.15)$$

Вони можуть бути сформовані як задачі оптимального управління, якщо здійснити заміну

$$x \equiv t, \quad x_0 \equiv t_0, \quad x_k \equiv t_k,$$

$$y_i(x) = x_i(t), \quad \frac{dx_i}{dt} = u_i(t) \equiv \dot{y}_i(t), \quad i = \overline{(1,n)}. \qquad (2.16)$$

Підстановка співвідношень (2.16) у функціонал (2.15) зводить задачу варіаційного обчислення до задачі оптимального управління з функціоналом

$$P = \int_{t_0}^{t_k} G(x_1, x_2,\ldots,x_n; u_1, u_2,\ldots,u_n)dt\ , \qquad (2.17)$$

де $\frac{dx_i}{dt} = u_i, \quad i = \overline{1,n}$.

Варіаційні задачі ефективно моделюються на ґратках. При цьому попередньо заданий вираз (2.15) перетворюється на вирази (2.10)–(2.14) і після цього розв'язується задача визначення найкоротших шляхів.

2.3. Задачі варіаційного числення та оптимального управління в теорії прийняття рішень та системах штучного інтелекту

У наш час бурхливо розвиваються теоретичні узагальнення варіаційних задач, а також близьких до них задач оптимального управління та диференціальних ігор. У загальному випадку вираз, функціонально залежний від n-функцій $y_i(x)$, де $i = \overline{1,n}$, має вигляд

$$J(y) = \int_{x_1}^{x_2} F(x, y_1, y_2, \ldots, y_n, y_1', \ldots, y_n')dx; \qquad (2.18)$$

$$y_i(x_1) = y_{i1}; y_i(x_2) = y_{i2}; i = \overline{1,n}, \qquad (2.19)$$

де y_{i1} і y_{i2} – граничні умови, задані у вигляді чисел;

F – безперервна підінтегральна функція, яка має безперервні частинні похідні до третього порядку включно за всіма аргументами.

Варіаційна задача містить у собі визначення умов, яке задовольняє вектор-функцію (y_1, y_2 ,..., y_n), вона доставляє функціоналу (2.18) екстремум при граничних умовах (2.19). Функції можуть мати і похідні вищих порядків

$$J(y) = \int_{x_1}^{x_2} F(x, y, y', y'', y^n)\, dx \qquad (2.20)$$

на функціях $C_n[x_1, x_2]$, які мають безперервну n-ну похідну на $[x_1, x_2]$, при цьому граничні умови записуються як

$$y^{i}(x_1) = A_i;\ y^{i}(x_2) = B_i;\ i = \overline{1, n-1}. \qquad (2.21)$$

Розв'язання варіаційної задачі полягає у знаходженні функції, яка досягає екстремуму функціонала (2.20) і на кінцях задовольняє умови (2.21). Щодо підінтегральної функції F, то розуміється існування безперервних за сукупністю всіх аргументів похідних до (n+1)-го порядку включно. Така варіаційна задача називається задачею Лагранжа. Розв'язання варіаційної задачі, у якій функціонал має похідну вищих порядків, може бути зведено до розв'язання варіаційної задачі функціоналом (2.18), що залежить від кількох функцій, шляхом введення всіх похідних вище першого порядку як нових незалежних змінних, зв'язавши їх одна з одною умовами

$$y'' = \frac{dy^{I}}{dx};\ y''' = \frac{dy^{II}}{dx};\ \ldots\ ;\ y^{(n)} = \frac{dy^{(n-1)}}{dx}.$$

Якщо на функцію y(x), яка дає екстремум функціонала, не накладені будь-які умови, то екстремум у цьому випадку називається безумовним. Існує також ряд варіаційних задач на умовний екстремум. Прикладом такої задачі є ізопериметрична задача [2], яка формулюється так: серед усіх кусково-рівних векторів-функцій $y = \{y_1(x), y_2(x), \ldots, y_n(x)\}$, які набувають заданого значення на кінцях інтервалу $[x_1, x_2]$, знайти функцію, що досягає екстремуму функціонала

$$J_0(y) = \int_{x_1}^{x_2} f_0(x, y, y')dx$$

за умови

$$J_i(y) = \int_{x_1}^{x_2} f_i(x, y, y')dx = C_i,\ i = \overline{(1, m)},$$

де C_i – константа.

Функції $f_0(x, y, y')$ і $f_i(x, y, y')$ визначені та мають безперервні за сукупністю своїх аргументів похідні другого порядку, коли

точка (x, y) розміщена у якійсь області G простору (x, y), а вектор «y» проходить будь-які кінцеві значення. Варіації функціоналів $I_i(y)$, що взяті на мінімізованому векторі, лінійно незалежні.

Крім ізопериметричної задачі, до варіаційних задач на умовний екстремум належать задачі Лагранжа, Майєра і Больца. Загальна задача Лагранжа визначається таким чином [2]: серед усіх кусково-рівних векторів-функцій y знайти вектор-функцію $y = \{y_1(x), y_2(x), ..., y_n(x)\}$, яка досягає екстремуму функціонала

$$J_0(y) = \int_{x_1}^{x_2} f_0(x, y, y')dx$$

за умови

$$f_i(x, y, y') = 0; \quad j=\overline{1,m} < n$$

і умовах на кінцях

$$\psi_k(x_1, y(x_1), x_2, y(x_2)) = 0, \quad k = 1, 2, ... ,P \le 2n + 2,$$

де функції f_τ (x, y, y') $(\tau= 0, 1, 2 ,..., m)$ визначені та мають безперервні за сукупністю всіх своїх аргументів особисті похідні третього порядку.

Матриця $\left\| \frac{df_i}{dy_i} \right\|$ має ранг m в усіх точках (x, y), які належать до якоїсь області простору (x, y), коли вектор y' пробігає будь-які значення на кінцях. Матриця $\left\| \frac{d\psi_k}{dx_1}; \frac{d\psi_k}{dy_{i1}}; \frac{d\psi_k}{dx_2}; \frac{d\psi_k}{dy_{i2}} \right\|$ має ранг p. Функції ψ_k мають безперервні особисті похідні третього порядку. Зв'язок називається голономним, якщо він не має похідних або може бути зведеним до вигляду, який не має похідних. Неголономні зв'язки мають як самі непохідні функції $y_1(x)$, $y_2(x)$, ... , $y_n(x)$, так і їх похідні.

Варіаційна задача на умовний екстремум у формі задачі Майєра ставиться таким чином: серед систем гладких функцій $y_0(x)$, $y_1(x)$, ... , $y_n(x)$, які задовольняють умову

$$\varphi_i(x, y, y') = 0,\ i = 0, 1, 2, ..., m < n$$

і умови на кінцях

$$y_0(x_1) = a;$$

$$y_1(x_1) = a_1, ..., y_n(x_1) = a_n;$$

$$y_1(x_2) = b_1, ..., y_n(x_2) = b_n,$$

знайти таку систему функцій, у якій $y_0(x)$ має при $x = x_2$ екстремум.

До задачі Лагранжа можуть бути зведені задача Майєра та ізоперіметрична задача.

Необхідно зазначити, що ізоперіметрична задача, задача Лагранжа і задача Майєра можуть розглядатися як окремі випадки задачі Больца, хоча задачі Лагранжа, Майєра і Больца характеризуються рівним ступенем загальності. Дуже важливу роль в управлінні АСУ ЗТ мають задачі, що пов'язані з управлінням процесами або об'єктами різної фізичної природи. Стан фізичного процесу або об'єкта характеризується змінним станом (фазовими координатами $x_1(t)$, $x_2(t)$,..., $x_n(t)$).

Фізичний процес або динаміка об'єкта описується системою диференціальних рівнянь (рівнянь стану), наприклад

$$\frac{dx_i}{dt} = f_i(x_1, x_2, ..., x_n, u_1, u_2, ..., u_n)\ i = \overline{1, n},$$

де u_i – змінні управління;

t – час.

Вільні змінні u_i дають змогу ставити задачу оптимального управління в найкращому розумінні заданого критерію – вибору змінних управління. Задача оптимального управління полягає у визначенні змінних управління $u_i = u_i(t)$, $i = 1, 2, ..., n$ в інтервалі $t_0 \le t \le t$, які забезпечують екстремум критерію якості, заданого у вигляді функціонала

$$P(t_k) = \int_{t_0}^{t_k} G(x_1, x_2, ..., x_n, u_1, u_2, ..., u_n)\, dt,$$

і задовольняють обмеження, наприклад

$$Q_j(u_1, u_2, ..., u_n) \le 0, \quad j = 1, 2, ..., N,$$

які визначають замкнену область допустимих управлінь. Оптимальне управління $u_i(t)$ визначає оптимальну траєкторію $x_i = x_i(t)$ в n-вимірному фазовому просторі, рух у якому з початкового стану в кінці забезпечує досягнення оптимального значення функціонала (критерію якості). Розв'язання задачі оптимального управління потребує надання початкових $x_i(t_0)$ і кінцевих $x_i(t_k)$ станів. У багатьох задачах управління початковий стан $[x_1(t_0), x_2(t_0),...,x_n(t_0)]$ задається у вигляді $(n-n_0)$-вимірного простору початкових станів (гіперповерхня, лінія або точка в просторі станів)

$$A_j[x_1(t), x_2(t),..., x_n(t)]=0, \; j = 1,2,..., n_0 \le n.$$

Так само може бути задано кінцеве положення $[x_1(t_k), x_2(t_k),..., x_n(t_k)]$:

$$B_j[x_1(t_k), x_2(t_k), ..., x_n(t_k)] = 0, \; i = 1, 2, ..., n_k \le n.$$

Задачу оптимального управління можна розглядати як варіаційну задачу на умовний екстремум, наприклад, як задачу Лагранжа, Майєра або Больца. Однак застосування варіаційних методів до задач оптимального управління зустрічає відповідні труднощі, оскільки задачі оптимального управління містять низку особливостей, не врахованих у варіаційних задачах. Це пов'язано з тим, що значення управління, яке розглядається як одна з невідомих функцій, що належить замкненій множині U, може бути задано вектором управління, який обмежений умовами $[U(t)] \le 1]$. Крім того, підінтегральний вираз функціонала і рівняння руху об'єкта розглядаємо як рівняння зв'язку, незалежне від похідної управління u', що призводить до

виродженого вигляду одного з рівнянь Ейлера, яке в цьому випадку не буде диференціальним. Варто також зазначити, що у варіаційних задачах необхідні умови мінімуму функціонала виведені в припущенні, що невідомі функції належать до класу двічі диференційованих функцій, а в задачі оптимального управління розглядається більш широке коло кусково-безперервних функцій. У задачах оптимального управління екстремум функціонала часто досягається при управлінні u(t), що має точки розриву першого роду, і за рівнянням руху призводить до наявності точок розриву похідної оптимальної траєкторії, а положення і кількість точок розриву заздалегідь невідомі. Оптимальне рівняння може розглядатися як узагальнене варіаційне числення. Усі задачі варіаційного числення, пов'язані з максимізацією або мінімізацією інтегралів

$$P = \int_{x_0}^{x_k} G[y_1(x), y_2(x) ,..., y_n(x), y_1'(x), y_2'(x) ,..., y_n'(x), x]\, dx,$$

при відповідних обмеженнях і граничних умовах можуть бути сформульовані як задачі оптимального управління, якщо провести заміну

$$x \equiv t,\ x_0 \equiv t_0,\ x_k \equiv t_k;$$

$$y_i(x) \equiv x_i(t),\ \frac{dx_i}{dt} \equiv u_i(t) \equiv y_i'(x),\ i = 1, 2 ,..., n.$$

Ця підстановка зводить задачу варіаційного числення до задачі оптимального управління з функціоналом

$$P(t_k) = \int_{t_0}^{t_k} G\,(x_1, x_2 ,..., x_n, u_1, u_2 ,..., u_n)\, dt\ ,$$

де

$$\frac{dx_i}{dt} = u_i,\ \ i = 1, 2 ,...., n.$$

Багато задач управління формалізуються у вигляді диференціальних ігор. Стан фізичного процесу або об'єкта характеризується змінами стану $x_1(t)$, $x_2(t)$,..., $x_n(t)$, зміна яких описується системою диференціальних рівнянь

$$\frac{dx_i}{dt} = f_i(x_1, x_2 ,..., x_n, u_1, u_2 ,..., u_ι, v_1, v_2 ,..., v_m), \; i=\overline{1,n} \qquad (2.22)$$

або векторним диференціальним рівнянням

$$\frac{dx}{dt} = f(x, u, v), \qquad (2.23)$$

де $x=(x_1,x_2 ,...,x_n)$ – точка n-вимірного фазового простору R_n, яка визначає стан процесу і належить області $x \in R_n$;

$u=(u_1,u_2 ,...,u_n)$; $v=(v_1,v_2 ,...,v_m)$ – управляючі параметри першого і другого гравців, які відповідно належать замкнутим і обмеженим множинам E_u і E_v в евклідових просторах R_e і R_m,

$$u\in E_u; \; v \in E_v; \qquad (2.24)$$

$f = (f_1, f_2 ,..., f_n)$ – дійсна вектор-функція, визначена на прямому множенні $X\times E_u \times E_v$.

Управляючі параметри u(x) і v(x), які вибираються в кожний момент часу t залежно від стану процесу x, прийнято називати стратегіями гравців. Стратегії гравців u(x) і v(x), визначені на X, набувають відповідно значення Eu і Ev і звичайно вибираються з умови оптимізації якогось критерію, який називається ціною. Розв'язання системи диференціальних рівнянь (2.22) при обраних стратегіях u(x) і v(x) і початкових умовах $x(t_0) =x$, де t_0 – момент початку гри, називається траєкторією або партією, яка починається в точці x_0 і визначається як $X[x_0, t, u(x), v(x)]$. Розвиток гри проходить в області X, якій належать усі траєкторії. Гра вважається закінченою, коли точка x досягає термінального простору M. Ціна може бути запропонована для широкого класу ігор у формі

$$P[x_0, t, u(x), v(x)] = \int_{t_0}^{t_k} G(x, u, v)\, dt + S[x(t_k)], \qquad (2.25)$$

де G – задана функція, яка визначена на X × Eu × Ev;

S[x(tk)] – функція кінцевого стану, визначена на термінальному просторі M.

Інтегрування проводиться вздовж траєкторії від моменту початку гри t0 до моменту закінчення tk, відповідного моменту досягнення точкою x термінального простору M. У випадку S[x(tk)] = 0 плата P називається інтегральною, а при G = 0 – термінальною. Зазвичай у диференціальних іграх цілі гравців вважаються протилежними, за вибір стратегій u(x) і v(x) приймають принцип мінімаксу, перший гравець формує стратегію u(x), що мінімізує плату P, за умови максимізації плати другим гравцем

$$\min_{u(x) \in E_u} \max_{v(x) \in E_v} P[x_0, u(x), v(x)] = v_1,$$

а другий гравець формує стратегію v(x), що максимізує плату P, за умови мінімізації плати першим гравцем

$$\max_{u(x) \in E_u} \min_{v(x) \in E_v} P[x_0, u(x), v(x)] = v_2.$$

Стратегії u0(x) і v0(x) називаються оптимальними, якщо

$$v_1 = v_2 = P[x_0, u_0(x), v_0(x)].$$

Виконання цієї умови свідчить про наявність сідлової точки гри, яка має таку особливість, що будь-яке відхилення від оптимальної стратегії одним гравцем призводить до втрати в платі за умови вибору оптимальної стратегії другим гравцем

$$P[x_0, u(x), v_0(x)] \geq \min_{u(x) \in E_u} P[x_0, u(x), v_0(x)];$$

$$P[x_0, u_0(x), v(x)] \leq \max_{v(x) \in E_V} P[x_0, u_0(x), v(x)].$$

Плата P = [x0, u0(x), v0(x)], яка відповідає оптимальним стратегіям u0(x) і v0(x), називається ціною гри. Основна задача диференціальних ігор полягає у визначенні ціни гри оптимальних стратегій гравців і траєкторій, відповідних оптимальним стратегіям. Ця постановка задачі належить до класу диференціальних ігор двох гравців з нульовою сумою, коли виграш одного гравця дорівнює програшу другого. Диференціальні ігри є найбільш загальним класом оптимізаційних задач. Наприклад, задачі оптимального управління можуть розглядатися як окремий випадок диференціальної гри. Якщо зауважити, що задачі оптимального управління можуть розглядатися як узагальнене варіаційне обчислення, то між диференціальними іграми, оптимальним управлінням і варіаційними задачами існує зв’язок щодо їх математичних моделей.

2.4. Задачі дискретного програмування та оптимізаційні задачі на графах для систем прийняття рішень та штучного інтелекту

При роботі автоматизованих систем управління як задачі управління їх функціонуванням, так і задачі, які автоматизує сама система, тісно пов'язані з розв’язанням широкого кола оптимізаційних задач. У безперервних задачах звичайно описується множина дійсних чисел чи деяка функція, а в комбінаторних задачах – деякий об'єкт з кінцевою чи нескінченною обчислювальною множиною. Такими об'єктами, зазвичай, бувають цілі числа, множина, перестановка чи граф. На перетині комбінаторної і безперервної оптимізації перебуває лінійне програмування. З одного боку, це безперервна оптимізаційна задача, а з другого – її можна розглядати як комбінаторну. У загальному випадку вона служить основою для вивчення суто комбінаторних задач.

Визначення 1. Індивідуальна задача оптимізації – це пара (F, C), де F – довільна множина, область допустимих точок, C – функція вартості, що здійснює відображення $C : F \to R^1$. Необхідно знайти точку $f \in F$, для якої $C(f) \le C(y)$ для всіх $y \in F$. Така точка називається глобально оптимальним розв'язком.

Визначення 2. Задача оптимізації – це множина індивідуальних задач оптимізації.

В індивідуальній задачі задано вхідні дані та існує достатньо інформації для отримання розв'язку, у той час як загальна задача – це набір індивідуальних задач, що породжуються однаковим способом. Говорячи про індивідуальну задачу комівояжера, ми розуміємо, що задано матрицю відстаней, якщо ж ми говоримо про задачу комівояжера в цілому, то маємо на увазі, що набір усіх індивідуальних задач відповідає всім матрицям відстаней.

Оскільки задачі оптимізації будуть розглядатися в обчислювальному аспекті, корисно спочатку домовитися, як буде подана індивідуальна комбінаторна задача оптимізації на вході обчислювальної машини. Можна, звичайно, виписати всі допустимі розв'язки і для кожного вказати значення C. Проте для більшості задач деякі індивідуальні задачі будуть мати непропорційно велику кількість допустимих розв'язків, як це буває в задачі комівояжера. У подальшому будемо припускати, що F і C задані неявно за допомогою двох алгоритмів A_F і A_C. Алгоритм A_F за цим комбінаторним об'єктом f множиною параметрів S вирішує, чи є f множиною розв'язків F, що визначаються цими параметрами. У свою чергу A_C за цим допустимим розв'язком f та іншою множиною параметрів Q видає значення C(f). Тоді індивідуальну комбінаторну задачу можна визначити [19] як подання параметрів S і Q з використанням фіксованого кінцевого алфавіту та деякого стандартного, розумного кодування.

Наприклад, у задачі комівояжера з N місцями параметром S є ціле число N, а параметрами Q – елементи симетричної N×N матриці відстаней $\| l_{ij} \|$. Алгоритм A_F за даним об'єктом f і параметром N буде перевіряти, чи є f обходом N місць, тобто циклічною перестановкою на множині {1, 2, …, N}.

Алгоритм A_C за даними обходу f та матриці $\left\| l_{ij} \right\|$ обчислює вартість C(f), складаючи всі елементи матриці $\left\| l_{ij} \right\|$, відповідні відстаням, які проходять при цьому обході.

Таким чином, комбінаторна задача оптимізації – це обчислювальна задача такого виду: за даними подання параметрів S і Q для алгоритмів A_F і A_C знайти оптимальний допустимий розв'язок. Такий варіант називають оптимізаційним варіантом цієї задачі.

Проте комбінаторну задачу можна поставити в такій формі: за даними S і Q знайти вагу оптимального розв'язку. Такий варіант називають обчислювальним варіантом комбінаторної задачі оптимізації. Припустимо, що алгоритм A поліноміальний. Якщо вартість C просто обчислити, то обчислювальний варіант комбінаторної задачі не може бути набагато важчий, ніж оптимізаційний варіант.

Третій варіант комбінаторної задачі оптимізації особливо важливий при визначенні обчислювальної складної задачі. Цей варіант ще називається варіантом розпізнавання і має такий вигляд: для цієї індивідуальної задачі 1, тобто подання параметрів S і Q, та цілого числа L визначити, чи існує такий допустимий розв'язок $f \in F$, що $C(f) \le L$.

На відміну від двох раніше розглянутих варіантів, варіант розпізнавання являє собою питання, на яке можна відповісти «так» чи «ні». Очевидно, відповісти на це питання не набагато важче, ніж розв'язати відповідну обчислювальну задачу, тому що після її розв'язання залишається тільки порівняти оптимальну вагу $C^*(f)$ з L і дати відповідь «так» у тому випадку, якщо $C^*(f) \le L$. Таким чином, використовуючи тільки припущення про те, що C легко обчислити, ми встановили, що кожний з варіантів (оптимізаційний, обчислювальний і розпізнавальний) не складніший за попередній. Виникає питання: «Чи не мають ці варіанти однакову складність?» – або, іншими словами, чи не можна знайти обчислювальний варіант, ефективно використовуючи гіпотетичний алгоритм як обчислювальну задачу у варіанті розпізнавання, і чи не можна зробити те саме для оптимізаційного й обчислювального варіантів? Якщо вага оптимального розв'язку є цілим числом, алгоритм якого

обмежений поліномом від розміру входу, тоді і варіант його розпізнавання також може бути розв’язаний ефективно. Те саме твердження буде справедливим і для обчислювального варіанта [19]. Слід зазначити справедливість припущення про те, що коли logC(f) обмежений поліномом від розміру входу, тоді для розв’язання обчислювальної задачі можна ефективно використовувати будь-який алгоритм, який розв’язує задачу розпізнавання. Не відомо загального способу для знаходження оптимізаційного варіанта. Проте до деяких задач застосовувались певні варіанти методу «динамічного програмування».

Низка задач оптимізації припускає графову постановку і може бути ефективно розв’язана за допомогою методів теорії графів. Ці ж задачі добре описуються мовою математичного (дискретного) програмування, у зв'язку з чим їх можна вважати частковими задачами цього напрямку. Урахування специфіки задач, а також постановка їх мовою теорії графів дає змогу одержувати більш прості методи розв’язання задачі на порядок більше, ніж розміри цих же задач у загальній постановці математичного програмування [16]. З другого боку, теоретико-графова інтерпретація забезпечує наочність і припустимість постановки досить складних прикладних задач та методів їх розв’язання.

Розглянуті нами визначення для варіантів розпізнавання задач оптимізації дають змогу вивчати складність задач обох видів у єдиній постановці. Виділяють клас P-задач розпізнавання, що можуть бути розв’язані деяким поліноміальним алгоритмом. Клас P визначається дуже точно [8] за допомогою будь-якого формального визначення алгоритмів, такого як машина Тьюрінга. Клас P – це клас відносно простих задач розпізнавання, для яких існують ефективні алгоритми. Більш ємнісний клас розпізнавання – це клас NP, для якого досі не знайдено поліноміальних алгоритмів. Як показано в роботах Кука, задачі цього класу характеризуються дивною властивістю: вони всі поліноміально зводяться одна до одної, тому якщо буде знайдений поліноміальний алгоритм для розв’язання якоїсь із них, то будуть знайдені алгоритми для всього цього класу задач. Не можна сказати останнього слова про складність розв’язання NP-повних задач до того часу, доки не буде доведена або спростована гіпотеза, що $P \neq NP$. Незважаючи на велику теоретичну

важливість цієї задачі та зацікавлення, що висловлюються до неї багатьма фахівцями з обчислювальних наук, доказів цієї гіпотези ще не видно. Задачі з класу NP важко піддаються розв'язанню навіть при використанні сучасних ЕОМ. Так, вимірність задач комівояжера, що розв'язуються на сучасних машинах точними методами, не перевищує 70. Дуже наочно показана обмеженість можливостей ЕОМ для розв'язання цих задач у табл. 2.1, взятої з роботи М. Гері та Д. Джонсона [8].

Таблиця 2.1

Розміри найбільшої задачі, що розв'язується на ЕОМ за одну годину

Функція часової складності	Сучасні ЕОМ	ЕОМ, які в 100 разів швидші	ЕОМ, які в 1000 разів швидші
n	N_1	$100 \cdot N_1$	$1000 \cdot N_1$
n^2	N_2	$10 \cdot N_2$	$31{,}6 \cdot N_2$
n^3	N_3	$4{,}64 \cdot N_3$	$10 \cdot N_3$
n^5	N_4	$2{,}5 \cdot N_4$	$3{,}98 \cdot N_4$
2^n	N_5	$6{,}64+N_5$	$9{,}97+N_5$
3^n	N_6	$4{,}19+N_6$	$6{,}29+N_6$

Як видно з наведеної таблиці, складність перебірних алгоритмів недопустимо велика. Тому актуальною є проблема побудови алгоритмів невеликої складності або так званих ефективних алгоритмів. При цьому під ефективним розуміється такий алгоритм, складність якого виражається поліноміальною функцією від вимірності задачі. І тут ми стикаємось з важливим моментом. Усі задачі оптимізації можна розділити на два підкласи: задачі, для яких існують ефективні алгоритми, і задачі, для яких такі алгоритми поки не відомі. Причому серед останніх багато задач ,які мають таку властивість, що за побудованим хоча б для однієї з них ефективним алгоритмом можна легко знайти такий алгоритм для всієї цієї множини задач. Зазначимо, що для оцінки складності для одного й того самого алгоритму розв'язання використовуються різні показники.

Наприклад, складністю в гіршому випадку називається така кількість t(n) тактів роботи алгоритму, де будь-яка індивідуальна

задача вимірності n буде розв’язана цим алгоритмом за кількість кроків, що не перевищує t(n). Якщо ж підсумувати кількість тактів усіх задач цієї вимірності і поділити на кількість цих задач, то буде отримано іншу міру складності – складність у середньому. Як правило, у цій роботі йтиметься про складність у гіршому випадку. Крім того, слід ураховувати, що складність розв’язання цих задач тісно пов'язана з використанням пам'яті. Наприклад, нехай потрібно визначити кількість одиниць у подвійній послідовності довжини n. Алгоритм, що підраховує одиниці, послідовно переглядає всі елементи послідовності і має витрати праці O(n) (один перегляд вважається за один крок роботи) і потребує O(n) одиниць пам'яті. Можна припустити інший алгоритм, що потребує 2n комірок пам'яті для зберігання всіх подвійних послідовностей довжиною n, причому номер комірки – це подвійна послідовність. Задача тепер розв’язується за одне звернення до пам'яті, тобто витрати праці алгоритму O(1), а потрібна пам'ять O(2n). Широкий клас задач дискретної оптимізації може бути ефективно розв’язаний з використанням теорії графів.

Екстремальні задачі на графах відзначаються великою різноманітністю як за змістом практичних задач, що їх породили, так і за можливими підходами до способів їх розв’язання. Можна виділити такі класи екстремальних задач на графах: про структурний аналіз (на досяжність і зв’язність графа); про незалежні домінуючі та покриваючі множини; про фарбування графів; про екстремальні шляхи, про екстремальні потоки та ін.

Серед множини екстремальних задач на графах виділяють два великих класи: задачі про екстремальні шляхи і задачі про екстремальні множини вершин із заданими властивостями. Виділення цих задач значною мірою зумовлено універсальністю їх застосування як для розв’язання суміжних задач теорії графів, так і для широкого кола прикладних проблем, а також подальшим вивченням можливостей, пов'язаних зі створенням спеціалізованих обчислювальних засобів. Важливий клас задач у теорії прийняття рішень й управління становлять варіаційні задачі, що тісно пов'язані з розв’язанням задач оптимального управління, ізоперіметричних задач, задач Лагранжа, Майєра, Больца і диференціальних ігор.

2.5. Постановка екстремальних задач на графах для систем прийняття рішень та штучного інтелекту

Наведемо постановки деяких екстремальних задач на графах у змістовному вигляді і в термінах математичного програмування.

2.5.1. Задача про найкоротший шлях

Задано граф G(V, E), кожному ребру якого приписана вага $\|l_{ij}\|$. Вага ребра в загальному випадку може набувати як негативного, так і позитивного значення. Фіктивним ребрам приписується вага $l_{ij} = 0$, кожній парі (i, j), для яких не існує ребра, що з'єднує їх, приписується вага $l_{ij} = \infty$. Якщо для послідовності вершин $v_0, v_1, ..., v_p$`` визначаємо шлях у G, то його довжина визначається як сума

$$\sum_{i=1} b(v_{i-1}, v_i). \qquad (2.26)$$

Нас буде цікавити знаходження найкоротшого шляху з вершини $s \in V$ до $t \in V$. Довжину такого найкоротшого шляху ми будемо позначати l(s, t). Зазначимо, що якщо кожний контур нашого графа має позитивну довжину, то найкоротший шлях буде елементарним, тобто в ньому не буде повторень вершин. Якщо в графі існує контур негативної довжини, то відстань між деякими парами вершин невизначена. Обходячи цей контур достатню кількість разів, ми можемо знайти шлях між цими вершинами з довжиною, меншою від довільного числа. У такому випадку можна було б говорити про довжину найкоротшого елементарного шляху, однак задача, поставлена таким чином, буде значно складнішою, оскільки вона містить у собі задачу існування гамільтонового шляху. У термінах математичного програмування задача формується таким чином:

$$\min f = \sum_i \sum_j l_{ij} f_{if} \qquad (2.27)$$

за умови

$$\sum_j f_{sj} - \sum_f f_{js} = 1; \tag{2.28}$$

$$\sum_j f_{ij} - \sum_j f_{ji} = 0; \tag{2.29}$$

$$\sum_j f_{ij} - \sum_j f_{it} = -1; \tag{2.30}$$

$$f_{ij} \geq 0.$$

Тобто ми розглядаємо цю задачу як задачу знаходження шляху з вершини s в t, для якого вартість проходження одиниці потоку по цьому шляху мінімальна. Згідно з умовою (2.28) одиниця потоку витікає з джерела. Рівність (2.29) гарантує збереження даної одиниці потоку при протіканні в графі G. Згідно з рівністю (2.30) одиниця потоку впадає в стік. Якщо ребро (i, j) належить найкоротшому шляху, то $f_{ij} = 1$, інакше $f_{ij} = 0$. Значення функціонала визначає довжину найкоротшого шляху. Слід зазначити, що аналогічно формується задача про найдовший шлях, але при цьому потрібно не мінімізувати, а максимізувати функціонал (2.27).

Можна навести багато практичних інтерпретацій задачі про найкоротші шляхи. Маса ребра може відповідати вартості або часу передачі інформації між вершинами. У такому випадку ми шукаємо найбільш дешевий (або найбільш швидкий) шлях передачі інформації. Якщо вага ребер дорівнює ймовірності P(i, j) безаварійної роботи каналу передачі інформації шляхом передачі інформації, що дорівнює добутку ймовірності його ребер, то задачу знаходження найбільш надійного шляху можна звести до задачі про найкоротший шлях, замінюючи вагу l_{ij} на $a_{ij} = -\log P(i,j)$.

Одним з додатків задачі про найкоротший шлях на графі є нижченаведена задача. Дискретна система автоматичного управління може перебувати в одному із станів $q_1, q_2, \ldots, q_m$, які утворюють кінцеву множину. Вплив, що управляється, – кінцева

множина значень, що становлять C_1, C_2 ,…., C_p . Перехід системи з одного стану в інший здійснюється в дискретні моменти часу залежно від впливу, що управляється, і його стану на поточний момент часу.

Початковий стан системи q_0, а процеси управління розглядаються в дискретні моменти часу t = 0, 1, 2 , …, T, де T – фіксоване. Розв'язання задачі управління в такій постановці випливає з певної послідовності U_1, U_2,…, U_T впливів, які управляються і переводять систему на час T у стан q_T і мінімізують величину

$$\sum_{к=0}^{T} c(q^k, U^k),$$

де $C(q^k, U^k)$ – функція стану q^k і управління, що несе U_k;

q^k – елемент $\{q_1, q_2, \ldots, q_m\}$;

U^k – елемент $\{U_1, U_2, \ldots, U_p\}$.

Будується граф з множиною вершин декартового множення

$$\{q_{11}, q_2, \ldots, q_m\} \times \{0,1,\ldots,S\}.$$

Для кожної вершини $q_i^t = \{q_i, t\}$ визначаються ребра $\{q_i^t, q_i^t+1\}$, якщо існує вплив U_{ij}, що переводить системи зі стану q_i у стан q_j. Довжина кожного шляху відповідно дорівнює $C(q_i, U_{ij})$.

Кожному шляху, який веде з вершини $(q^0, 0)$ у вершину (q^T, T) побудованого графа, відповідає деяка послідовність впливів, що переводять систему з q^0 у q^T. Тоді найкоротший з цих шляхів від $(q^0, 0)$ до (q^T, T) визначає оптимальне управління на множині $\{U_1, U_2, \ldots, U_p\}$. Аналогічно формулюється задача про проектування транспортних магістралей з урахуванням забезпечення безперервно зростаючих перевезень [14]. Якщо в розглядуваному графі на найкоротші шляхи накласти додаткове обмеження, то можна сформулювати більш загальну задачу про найкоротший допустимий шлях [12].

2.5.2. Задача про найкоротший допустимий шлях

Існує граф G(V, E). Нехай кожній вершині v_i поставлена у відповідність множина T_i і функції $t_i(\mu_i)$, визначені на множині шляхів μ_i, що проходять через вершину v_i. Елементи множини T_i – довільні вектори підмножини. Якщо $t_i(\mu_i) \in T_i$, то серед множини всіх можливих шляхів μ_i, що проходять через вершину v_i, виділяється певний клас шляхів. Шлях μ_{im}, для вершини v_m називається допустимим, якщо

$$t_{ik}(\mu_{ik}) \in T_{ik}, k = 1,2,..m.$$

Якщо кожному допустимому шляху поставлена у відповідність вага $L(\mu)$, то розв'язання полягає у знаходженні допустимого в даному сенсі шляху в графі G з найменшою вагою $L(\mu)$. У множині вершин E можна виділити дві підмножини A, і B, одна з яких складається з початкових вершин усіх можливих допустимих шляхів. Таким чином, задачу можна поставити ширше: визначити додатковий шлях

$$\mu_{im} = (v_0, v_1,...,v_k, v_{k+1},...v_m)$$

з множини A в множину B, для якого

$$t_{im}(\mu_{im}) \in T^*_{im}, \quad T_{im} \subseteq T_{im},$$

з найменшою довжиною, тобто v_0 (початок) і v_m (кінець) не фіксовані, але відомо, що $v_0 \in A$, $v_m \in A$. Підмножина T_i задана, якщо $i \in B$. Функція $t_i(\mu_i)$ ставить у відповідність шляхові μ_i перелік номерів усіх його вершин.

Номер кожної вершини повторюється стільки разів, скільки разів через неї проходить шлях. Множини T_i складаються з усіх можливих підмножин множини {1, 2,…, N} номерів вершин графа. Будь-яка підмножина T_i містить однаковий набір номерів вершин, що становлять шлях, який починається в A і закінчується в B.

Багато практичних задач у теоретико-графовій постановці зводяться до пошуку деяких підмножин з множини вершин V графа G з визначеними властивостями. Розглянемо декілька найбільш типових і важливих задач цього класу.

2.5.3. Задача про максимально незалежну множину вершин графа

У неорієнтованому графі G(V, E) множина вершин S, де будь-які дві вершини в ньому несуміжні, тобто не зв'язані ребром, називається незалежною множиною вершин чи внутрішньотривкою множиною

$$\begin{aligned} &S \subset V, \\ &S \cap E \quad (S) = 0. \end{aligned} \tag{2.31}$$

Незалежна множина вершин S називається максимальною, якщо вона задовольняє умову (2.31), а також умову

$$P \cap E \quad (P) \neq 0 \quad \forall P \supset S,$$

тобто немає іншої незалежної множини, у яку воно б входило. У графі може бути більше однієї максимальної незалежної множини. Нехай H – сімейство всіх максимально незалежних множин графа G, тоді число $\alpha(G) = \max\{S\}$, а множина S', яка забезпечує максимальне значення α, називається найбільшою незалежною множиною. Задача полягає у знаходженні максимальної множини вершин (однієї з усіх можливих) з тією властивістю, що в цій множині між вершинами наявний зв'язок. Формально задача може бути подана таким чином [14].

Дано граф G(V, E), V = N, S – максимальна незалежна множина. Необхідно максимізувати

$$S = \max \sum_{i=1}^{N} y_i \tag{2.32}$$

за таких обмежень:

$$\nu_i \sum_{j=1}^{N} a_{ij} y_j \le 1; i = 1, 2, \ldots, N; a_{ij} = 1; j = 1, 2, \ldots, N; \qquad (2.33)$$

$$a_{ij} = \begin{cases} 0, \text{ якщо } l_{ij} \notin E \; i \neq j; \\ 1, \text{ якщо } l_{ij} \in E \; i, j = 1,2,\ldots,N, \end{cases}$$
$$y_i = \begin{cases} 0, \text{ якщо } \nu_i \notin s; \\ 1, \text{ якщо } \nu_i \in s \; i, j = 1,2,\ldots,N, \end{cases} \qquad (2.34)$$

де y_i – елементи максимальної незалежної множини S;

a_{ij} – елементи матриці суміжності графа з одиничними діагональними елементами.

Перше обмеження (2.33) відображає той факт, що якщо в множину входить якась вершина ν_i, то жодна суміжна вершина не може входити в цю множину. Прикладом використання цієї задачі може служити задача вибору проектів. Існує N – проектів, які повинні бути виконані, і припустимо, що для виконання проекту ν_i потрібна деяка підмножина R_i за наявністю ресурсів з множини {1, 2 ,…, P}. Припустимо, що кожний проект може бути виконаний за однакові проміжки часу. Можна побудувати граф G, кожна вершина якого відповідає деякому проекту, а ребро (ν_i, ν_j) – наявності спільних засобів забезпечення у проектах ν_i та ν_j, тобто умові $R_i \cap R_j = 0$. Тоді максимальна незалежна множина графа G являє собою максимальну множину проектів, що можна виконати водночас за один і той самий проміжок часу.

2.5.4. Задача розфарбування графа

Граф G називається r-хроматичним, якщо його вершини можуть бути пофарбовані з використанням r-кольорів (фарб) так, щоб ніякі дві суміжні вершини не були пофарбовані одним кольором. Найменше число r, при якому граф ще розфарбовується визначеним способом, називається хроматичним числом, а задача пофарбування зводиться до знаходження цього числа і підмножини вершин, які фарбуються

однаково. Хроматичне розфарбування розбиває множину вершин графа на підмножини, кожна з яких є незалежною, оскільки вершини, розфарбовані одним кольором, не повинні бути суміжними. Мовою 0-1 програмування задача пофарбування графа G з N вершинами формулюється таким чином.

Мінімізувати

$$Z = \sum_{j=1}^{q} \sum_{i=1}^{N} \rho_j \varepsilon_{ij} \tag{2.35}$$

при обмеженнях

$$\sum_{j=1}^{q} \varepsilon_{ij} = 1, \tag{2.36}$$

$$L\left(1 - \varepsilon_{ij}\right) - \sum a_{ik} \varepsilon_{kj} \geq 0, \; i = \overline{1, N}; \; j = \overline{1, q}; \tag{2.37}$$

$$\varepsilon_{ij} = \begin{cases} 1, & \text{якщо вершина } \nu_i \text{ розфарбована кольором j;} \\ 0, & \text{у протилежному випадку.} \end{cases} \tag{2.38}$$

де L – велике позитивне ціле число, більше від N;

α_{ik} – елемент матриці суміжності графа з нульовими діагональними елементами.

Обмеження (2.36) відповідає необхідності пофарбування вершин тільки в один колір.

Обмеження (2.37) забезпечує допустимість розфарбування, тобто якщо вершина νi пофарбована в колір j, то немає суміжної вершини з таким самим кольором. У функціоналі (2.35) li – відмітка для кольору j (кожному кольору відповідає відмітка Pj+1 > Pj). Матриця $\|\varepsilon_{ij}\|$ – це матриця розфарбування, що при мінімальному значенні Z визначає мінімальне розфарбування. Задача розфарбування широко застосовується у великій кількості прикладних задач. Розглянемо задачу розподілу n-задач між процесорами у багатопроцесорній системі.

Нехай кожне значення відповідає певній вершині графа. Кожного разу, коли два значення ν_i та ν_j не можуть бути виконані в одному процесорі, у граф G вводиться ребро (ν_i, ν_j). Якщо процесори мають необмежену продуктивність, тобто кожний момент може обробити скільки завгодно завдань, то задача знаходження найменшої кількості процесорів еквівалентна знаходженню хроматичного числа графа G, причому кожному процесору відповідає певний колір. Насправді процесори характеризуються обмеженою продуктивністю. Припустимо, що продуктивність кожного процесора однакова і дорівнює Q. Це можна інтерпретувати так, що в один колір зафарбовуються не більше Q вершин. У термінах 0-1 програмування вказана задача завантаження може бути сформульована таким чином. Мінімізувати

$$Z = \sum_{j=1}^{q} \sum_{i=1}^{N} \rho_j \varepsilon_{ij} \qquad (2.39)$$

при обмеженнях (2.37), (2.38) і

$$\sum_{i=1}^{N} \varepsilon_{ij} \le Q, i = \overline{1,q}.$$

Розв’язком цієї задачі є матриця $\|\varepsilon_{ij}\|$, що описує оптимальне розміщення (розподіл) завдань між процесорами, інакше кажучи, групування завдань за кольором. Число q є верхньою оцінкою кількості процесорів.

Процесори можуть мати різноманітну продуктивність і обсяг пам'яті. Нехай j-й процесор має продуктивність Q_i і обсяг пам'яті ν_j. Припустимо, що завданню протиставлена вага, відповідна трудовитратам завдання, скажімо і-му завданню відповідає вага w_i. Потрібно так розподілити завдання між процесорами, щоб виконувалися умови, які накладаються графом G, та задовольняли обмеження на продуктивність процесорів і найменшу спільну пам'ять для виконання завдань.

Тоді задача буде мати такий вигляд, мінімізувати функцію

$$Z = \sum_{j=1}^{q} \psi_j \nu_j \qquad (2.40)$$

при обмеженнях (2.37), (2.38) і

$$\begin{cases} \sum_{i=1}^{N} \varepsilon_{ij} w_i \le Q_i ; \\ \sum_{i=1}^{N} \varepsilon_{ij} \le L \psi_j , i = \overline{1,q} \end{cases} \qquad (2.41)$$

де $\psi_i = \begin{cases} 1, \text{якщо процесор виконує будь-яке завдання;} \\ 0, \text{у протилежному випадку;} \end{cases}$

L – довільне позитивне число, більше ніж N;

q – загальне число наявних процесорів.

Функціонал (2.40) означає, що жоден з процесорів не перевантажується.

Обмеження (2.41) гарантує здійсненність умови: якщо $\psi_{j0} = 0$ (тобто процесор вільний), то всі $\varepsilon_{ij} (i = \overline{1, N})$ теж дорівнюють 0.

У загальному випадку необхідно ввести додаткові змінні ψ_j, оскільки процесори (кольори) не можна відмітити тим способом, який характеризується виразом (2.36). Слід зазначити, що чим більш загальною стає задача, тим менш важливим стає її «розфарбований аспект». У розглянутій задачі можна виділити дві підзадачі: про «розфарбування» (відповідне обмеженню (2.38)) і про «ранець» (обмеженню, що визначається формулою (2.40)). Обмеження (2.37) і (2.41) є обмеженнями структурного характеру. Через ці два аспекти спільна задача розфарбування значно складніша для розв’язання, ніж «чиста» задача розфарбування.

Важливе значення має розв’язання задачі «розфарбування» в теорії розкладання. Так, наприклад, при укладанні графіків перевірки огляду можна поставити у відповідність вершину деякого графа, причому вершини графа будуть з'єднані ребром лише тоді, коли відповідні огляди не можна здійснити водночас.

Потрібно скласти такий графік оглядів, який був би зв'язаний з найменшими вершинними витратами з урахуванням обмежень на сукупність. Ця задача еквівалентна задачі про розфарбування вершин графа з використанням найменшого числа кольорів. Хроматичне число графа в цьому випадку відповідає огляду, що вимагає найменших часових витрат.

Задача про розфарбування графа може бути використана також при розподілі ресурсів.

Нехай для виконання будь-яких N робіт треба розподілити m наявних ресурсів. Вважаємо, що кожна робота виконується за деякий (однаковий для всіх робіт) проміжок часу і що для виконання i-ї роботи потрібна підмножина ресурсів Si. Побудуємо граф G: кожній роботі відповідає певна вершина графа, а ребро (vi, vj) існує в графі тоді і тільки тоді, коли для виконання i-ї роботи потрібен хоч би один спільний ресурс, тобто коли $S_i \cap S_j \neq 0$. Це означає, що i-та і j-та роботи не можуть виконуватися водночас. Розфарбування графа G визначає деякий розподіл ресурсів (за виконаними роботами), причому роботи, відповідні вершинам одного кольору, виконуються водночас. Найкраще використання ресурсів (тобто виконання всіх N робіт за найменший час) досягається при оптимальному розфарбуванні вершин графа G.

2.5.5. Задача про покриття

Нехай у графі G(V, E) знайдена множина вершин S V, де для кожної вершини v_i з множини V-S існує ребро, направлене з деякої вершини множини S у вершину v_j. Інакше кажучи, множина S інцидентна іншим вершинам V-S графа G, тобто

$$S \bigcup E(S) = V$$

і немов би покриває всі вершини графа G (враховуючи і вершини самої множини S). Множина S називається домінуючою множиною. У задачах про домінуючі множини цікавляться знаходженням множини S мінімальної потужності. У теорії графів задача про покриття полягає в описі множини ребер E′ графа G(V, E), де кожна вершина графа G інцидентна хоча б

одному із E′ і його потужність при цьому є мінімально можливою.

Більш спільна постановка задачі може бути такою. Існує кінцева множина $P = \{P_1, P_2, \ldots, P_n\}$ та деяке кінцеве сімейство її підмножин B_j, $j = 1, 2, \ldots, N$. Необхідно знайти мінімальне покриття множини S, тобто мінімальний набір таких підмножин B_j, де будь-який елемент множини P належить хоча б одній з виділених підмножин.

Мінімізувати

$$\sum_{j=1}^{N} U_j$$

за умови

$$\sum_{j=1}^{N} a_{ij} U_j \geq 1, \quad i = 1,2,\ldots,m,$$

$$a_{ij} = \begin{cases} 1, & \text{якщо } P_j \in B_j; \\ 0, & \text{у протилежному випадку;} \end{cases}$$

$$U_{ij} = \begin{cases} 1, & \text{якщо } B_j \text{ входить у покриття;} \\ 0, & \text{у протилежному випадку.} \end{cases}$$

Задача про покриття широко застосовується для інформаційного пошуку [20], у задачах знаходження множини маршрутів – для спрощення логічних виразів [20].

2.5.6. Задача комівояжера

Дано повний граф G, ребрам якого приписані довільні ваги $C = \|C_{ij}\|$. Знайти такий гамільтонів цикл, що має найменшу вагу. Задача знаходження найкоротшого гамільтонового циклу добре відома як задача комівояжера. У термінах лінійного програмування її можна сформулювати так.

Нехай ε_{ij} матриця N × N, у якій $\varepsilon_{ij} = 1$, якщо комівояжер їде безпосередньо з U_i в U_j, і $\varepsilon_{ij} = 0$ – у протилежному випадку. Тоді нам потрібно знайти значення величини ε_{ij}, яке мінімізує

$$Z = \sum_{j=1}^{N} \sum_{i=1}^{N} C_{ij}\varepsilon_{ij},$$

за умови, що

$$\sum_{j=1}^{N} C_{ij} = \sum_{i=1}^{N} \varepsilon_{ij} = 1;\ \forall(i,j) = \overline{1,N};\ \varepsilon_{ij} = \{0,1\}.$$

Умови гарантують, що розв'язок буде циклічним, тобто в кожну вершину входить і виходить одне ребро. Задача комівояжера тісно пов'язана із задачею про найкоротший остов і відкритою задачею комівояжера (задачею знаходження найкоротшого гамільтонового шляху).

Задача знаходження найкоротшого гамільтонового шляху була вперше досліджена Део і Хакімі і її постановка подана мовою лінійного програмування для повного графа з N вершинами. З постановки маємо N(N+1) змінних та $N(N+3)/2+1$ обмежень, які не можна сформулювати явно, але які не можна розглядати і неявно, причому за один раз (після кожного ітераційного застосування симплекс – методу) вводиться декілька з них.

Алгоритм розв'язання задачі комівояжера та її варіанти мають велику кількість практичних додатків. Наприклад, потрібно виготовити ряд виробів – скажімо N, використовуючи єдиний комплект апаратури. Вартість переналагодження апаратури для переходу з випуску виробу P на виготовлення виробу P_i постійна і не залежить від виробу, який іде за ним. Виникає питання: чи може бути знайдена циклічна послідовність виробництва продуктів $P_i = \overline{(1,N)}$, яка не потребує переналагодження апаратури. Нехай G – граф, вершини якого являють собою вироби. Існування ребра (V_i, V_j) означає, що

виріб P_j може йти за виробом P_i без переналагодження апаратури. Тоді відповідь на поставлене питання залежить від того, чи має цей граф гамільтонів цикл. Інші додатки містять складання розкладу виконання операцій на машинах, проектування електричних мереж, управління автоматичними лініями.

Контрольні питання

1. Пояснити модель булевого програмування.
2. Пояснити принцип оптимальності в задачі динамічного програмування.
3. Пояснити зв'язок задачі оптимального управління з варіаційним численням.
4. Пояснити зв'язок задач управління із задачами диференціальних ігор.
5. Що таке оптимальна стратегія у диференціальній грі?
6. Що таке задача оптимізації та індивідуальна задача оптимізації?
7. Як оцінюють складність алгоритмів розв’зання оптимізаційних задач?
8. Як формалізуються задачі про найкоротший шлях?
9. Як формалізуються задачі про незалежну множину?
10. Як формалізуються задачі про фарбування графів?
11. Як формалізуються задачі про покриття?
12. Як формалізується задача комівояжера?

Розділ 3. Проблематика розбудови паралельних обчислювальних систем та основні шляхи їх розвитку

3.1. Паралельні обчислювальні системи та алгоритми прийняття рішень у системах штучного інтелекту

3.1.1. Проблеми, пов'язані з розвитком паралельних обчислювальних систем та їх математичного забезпечення

Існує принципова різниця між реалізацією алгоритмів на класичній однопроцесорній ЕОМ та реалізацією цих алгоритмів на паралельних системах. Вона визначається тим, що будь-який алгоритм можна описати напрямленим графом, вершини якого відповідають операціям, що виконуються, а ребра – інформаційним зв'язкам між ними. Процес його реалізації на класичній однопроцесорній ЕОМ можна трактувати як процес послідовного вилучення по одній з вхідних вершин графа до повного його вичерпання. Оскільки такий процес реалізується для будь-якого ациклічного графа, то немає принципових обмежень в ефективній реалізації будь-якого алгоритму на однопроцесорній ЕОМ. Але якщо обчислювальна система містить P процесорів і P – достатньо велике, то становище змінюється радикально. Ефективна реалізація алгоритму тепер означає, що кожний раз у графі повинно вилучатися P вершин, але алгоритм через свою внутрішню структуру може не мати можливості на кожному кроці роботи виключати P вершин. Це означає, що таку обчислювальну систему не можна завантажити повністю, тобто деякі процесори будуть простоювати і через це не можна досягнути максимальної продуктивності системи. Ефективній реалізації алгоритму можуть також заважати особливості комунікаційного середовища, що пов’язують процесори між собою. Обмежені можливості швидких зв'язків у мережі можуть також бути неузгодженими зі структурою зв'язків в алгоритмі, при цьому на ефективність роботи паралельної обчислювальної системи суттєво впливає структура пам'яті системи. Слід зазначити, що підвищення швидкодії за рахунок розпаралелювання процесу може повністю нівелюватися на обмінних операціях та при конфліктних ситуаціях у разі

звертання до одного і того самого обчислювального ресурсу декількох процесів водночас. Звідси неминуче випливає висновок, що для існування можливості ефективної реалізації алгоритму на будь-якій паралельній обчислювальній системі необхідно, щоб структура алгоритму була узгоджена зі структурою системи. Відповідь на питання, існує таке узгодження чи ні, далеко не очевидна. Незважаючи на те, що кількість робіт, присвячених розпаралелюванню, рахується вже десятками тисяч, ясності в тому, що ж реально зроблено або як це можна робити, досі немає.

Системний підхід до питань теорії паралельних алгоритмів та паралельних обчислювальних систем відображається у напрямку досліджень, сформованих Г. І. Марчуком, який одержав назву «відображення проблем обчислювальної математики на архітектуру обчислювальних систем». Серед великого потоку інформації з теорії паралельних алгоритмів слід усе ж таки виділити ряд робіт, що нині можуть скласти деякий базис для подальшого просування вперед у розумінні цього питання. До них слід віднести роботи Г. І. Марчука, В. В. Воєводіна, В. Є. Котова, В. В. Васильєва, Д. Івенса, Є. Валяха та ін. У роботі [18] алгоритм ототожнюється з графом, причому графи алгоритмів виділяються тим, що вони ациклічні, тобто не мають контурів. Фактично такий орієнтований ациклічний мультиграф можна вважати графом деякого алгоритму, якщо ототожнити його з якимись операціями, узгодивши при цьому ребра графів з передачею інформації від операції. Тому виникає бажання ототожнити множину алгоритмів, що вивчаються, з множиною всіх орієнтованих ациклічних графів та розпочати вивчення структур таких графів. Але, як зазначає В. В. Воєводін, у цьому випадку більшість цікавих та найбільш важливих задач на графах належить до так званих NP-складних задач. Знаходження розв’язку таких задач пов'язане з повним перебором усіх варіантів. На підставі цього у роботі [18] зроблено висновок про те, що не варто сподіватися на отримання ефективного розв’язку загальної задачі про відображення проблем обчислювальної математики на архітектуру обчислювальних систем за допомогою деякого універсального методу. Будь-яка обчислювальна система являє

собою сукупність взаємопов’язаних і одночасно працюючих функціональних приладів. Опис процесу її роботи пов'язаний із зазначенням послідовності виконання на цих приладах деяких базових операцій над вхідними даними та результатами проміжних обчислень. Важливою характеристикою функціональних приладів є поняття завантаження, яке можна визначити як відношення часу виконання корисної роботи до часу зайнятості [23]. Завантаженість характеризується числом між 0 та 1. Вона дорівнює 0, якщо жоден з приладів системи не виконує жодної роботи. Максимальної швидкості можна досягнути лише в тому випадку, коли максимально завантажені всі функціональні прилади, що реалізують операції алгоритму, необхідні для розв’язання задачі. Крім того, умова досягнення максимальної завантаженості є тим критерієм, що показує, наскільки добре архітектура обчислювальної системи відповідає структурі алгоритмів, що реалізуються.

Функціональні прилади обчислювальної системи повинні обмінюватися між собою інформацією. Зв'язки між приладами можуть бути як змінними, так і постійними. Зміна зв'язків завжди здійснюється в результаті спрацювання визначеного приладу, на що відводиться відносно великий час. Тому для досягнення максимальної продуктивності системи в цілому доцільно мати постійні зв'язки між приладами впродовж максимально великого відрізка часу та прагнути до того, щоб за цей відрізок часу пропустити максимально можливий потік інформації. Створення комунікаційних мереж у паралельних обчислювальних системах, що забезпечують швидкі необхідні зв'язки між окремими функціональними приладами, є однією з найважливіших проблем у побудові паралельних обчислювальних систем. Поки швидкість спрацювання окремих приладів була не дуже великою, основними факторами, що перешкоджали створенню потрібних комунікаційних мереж, була кількість ліній зв'язку та складність комутаторів. Однак із зростанням швидкості спрацювання приладів поряд з цими факторами набула значення також і довжина ліній зв'язку. У певному відношенні найпростішою та найефективнішою є комунікаційна мережа, у якій немає комутаторів, усі прилади з'єднані безпосередньо один з одним та довжина всієї лінії зв'язку мінімальна. З кінця 1970-х рр.

з'явилися роботи, у яких рекомендується будувати спеціалізовані обчислювальні системи, що отримали назву «систолічні масиви». Ці системи сконструйовані гранично просто і в той же час мають нескладну комунікаційну мережу. Тому вони є досить ефективними за швидкодією, але кожна з них орієнтована на розв’язання надто вузького класу задач.

Якщо для послідовних числових методів майже всі питання, пов’язані з побудовою архітектури обчислювальної системи, уже визначені, то у паралельних методах поки відбувається в основному процес накопичування окремих методів та робляться спроби їх осмислення. Задача розроблення паралельних методів та їх програмного забезпечення виявилася складнішою в першу чергу через те, що на її розв’язання значно впливають індивідуальні особливості паралельних систем. Різноманітні паралельні обчислювальні системи вимагають особистих засобів організації обчислень. У свою чергу різноманітні засоби організації обчислень потребують різноманітних засобів організації даних, потребують створення різноманітних числових методів та алгоритмів, різноманітного числового програмного забезпечення, нових засобів та мов спілкування з обчислювальною технікою. Останнім часом формування паралельних обчислювальних систем пішло шляхом створення проблемно-орієнтованих систем для розв’язання задач первинної обробки інформації в автоматизованих системах управління, розв’язання задач лінійної алгебри, систем диференціальних рівнянь, рівнянь у частинних похідних, задачах комбінаторної оптимізації та теорії графів. Нагромадження великої кількості проблем навколо таких обчислювальних систем потребує комплексного та системного підходу до їх вирішення. Одним з функціональних аспектів розроблення комплексного підходу до вирішення проблеми відображення обчислювальних систем є вибір засобів математичного опису всіх об'єктів. Складність вибору визначається не стільки тим, що за допомогою таких засобів треба фіксувати опис деяких об'єктів, скільки необхідністю задавати та розпізнавати різноманітні їх якості, а також порівнювати та перетворювати описи окремих об'єктів. Ключовим завданням проблеми відображення є вивчення структури алгоритмів та структури обчислювальних систем.

Розв’язання інших задач здебільшого визначається тим, як і наскільки ефективно буде розв’язана саме ця задача. Математичний аналіз структури алгоритмів, а також структури супер-ЕОМ та інших високопродуктивних систем значною мірою ускладнюється їх великою різноманітністю, неоднозначністю та недостатнім визначенням понять, відсутністю формалізованих принципів. Існує велика кількість типів паралельних систем, а саме: конвеєрні, матричні, векторні, багатопроцесорні з архітектурою, що програмується, паралельні систолічні, спецпроцесори та ін. Звичайно виникають питання:

- що спільного в усіх цих системах та чим вони відрізняються;
- як ефективно реалізувати заданий алгоритм на заданій паралельній системі;
- як забезпечити всі паралельні системи необхідними програмами;
- яка повинна бути структура паралельних алгоритмічних мов?

Вирішення цих питань потребує принципово нових нестандартних підходів.

3.1.2. Взаємозв'язок алгоритмів та процесів їх реалізації у багатопроцесорних обчислювальних системах

Алгоритми можна визначати графами. Говорячи про алгоритм, звичайно припускають наявність якого-небудь правила, що описує сукупність виконуваних операцій та зв'язок окремих операцій між собою, або зазначення порядку, що описує їх послідовність. Таке правило може бути задане у вигляді математичних формул, програми, викладеної алгоритмічною мовою, таблицею дій, що виконуються, тощо. Не завжди правило розуміється однозначно. Тому будемо вважати, що алгоритм або те, що описує його правило, дають змогу якимось чином визначити:

- множину змінних, у перетворенні яких полягає реалізація алгоритму;
- множину операцій, що виконуються в процесі реалізації алгоритму;

- відповідність, що показує, які результати операцій, виконаних до цього, являють собою аргументи для кожної операції.

Множині операцій алгоритму поставимо у взаємовідносну відповідність множину точок. Якщо аргумент є результатом виконання іншої операції, то відповідні точки з'єднаємо ребром, направленим з точки, звідки береться результат. Побудований таким чином граф будемо називати графом алгоритму. Як показано в роботі [1], при реалізації алгоритму на паралельній обчислювальній системі великого значення набуває знання його паралельних форм.

Нехай заданий деякий алгоритм та його граф G (V, E). Припустимо, що множина вершин розбита на такі неперетинні підмножини V_1, V_2, V_3,…, V_a, що якщо $U \in V_i$, $u \in V_j$ та існує ребро (U, u), то i < j. У цьому випадку роздроблення $V = \bigcup_{i=1}^{a} V_i$ називається паралельною формою алгоритму в підмножині, а V_1, V_2 ,…, V_a – її ярусами. Число a називається висотою паралельної форми, $|V_i|$ – шириною і-го ярусу, а максимальне з цих чисел – шириною паралельної форми.

Паралельна форма мінімальної висоти називається максимальною, а її висота – висотою алгоритму. Максимальна ширина h паралельних форм називається шириною алгоритмів. Основними якостями паралельних форм є такі:

- ніякі дві вершини одного ярусу будь-якої паралельної форми не зв'язані ні одним ребром;
- висота алгоритму визначається рангом критичного шляху графа, і якщо V = N, то максимальний ранг критичного шляху не може перевищити r = N - 1;
- множення висоти будь-якої паралельної форми на її ширину не менше від загальної кількості вершин графа алгоритму.

Обчислювальні системи можна також подати орієнтованими графами. При цьому звичайно вершинам графів поставити у відповідність різноманітні функціональні прилади, а ребрами відзначати передачу інформації між ними.

Можна виділити дві основні задачі, які визначають взаємозв'язок алгоритмів та процесів їх розв’язання за допомогою обчислювальних систем з багатьма функціональними приладами.

Задача 1

Дано орієнтований граф, що описує обчислювальний алгоритм, та зазначено специфікацію операцій, що ототожнюються з його вершинами. Відома номенклатура приладу, що в сукупності виконує усі операції алгоритму. Необхідно в межах виділених лімітів вибрати склад функціональних приладів, побудувати орієнтований граф обчислювальної системи та вказати програму або множину програм, що забезпечують реалізацію цього алгоритму за мінімальний час.

Задача 2

Дано орієнтований граф, що описує обчислювальний алгоритм, та зазначено специфікацію операцій, що ототожнюються з його вершинами. Дано орієнтований граф, що описує обчислювальну систему, та вказано специфікацію приладів, що ототожнюються з його вершинами. Необхідно відповісти на питання, чи можна реалізувати цей описаний алгоритм на цій системі. Якщо позитивна відповідь, вказати програму або множину програм, що забезпечують реалізацію цього алгоритму за мінімальний час.

Обидві задачі називають задачами відображення алгоритмів на обчислювальних системах [18].

Установимо взаємовідносну відповідність між вершинами графів та деякою множиною спрацювання функціональних елементів (ФЕ) обчислювальної системи. Нехай задано граф G(V, E) алгоритму. Візьмемо як граф системи той самий граф G, вважаючи, що ФЕ, поміщені в певну вершину, можуть виконувати відповідну операцію. Будемо вважати, що кожний ФЕ одержує інформацію для своєї роботи згідно з графом G. Таку обчислювальну системи та всю сукупність програм, що реалізують на ній базовий алгоритм, називають базовими для даного алгоритму [1].

Пронумеруємо яким-небудь способом вершини графа G та позначимо через t_i момент додавання ФЕ, відповідного і-й вершині, а через τ_i – час виконання операцій і-го ФЕ. Вектор $t = (t_1, t_2, \ldots, t_n)$ називають часовою розверсткою базової обчислювальної системи. Зрозуміло, що різноманітних часових

розверсток існує нескінченна множина. Час виконання алгоритму визначається довжиною критичного шляху в G.

Задача пошуку k найкоротших шляхів розглядалася в роботах [19, 22], і для її розв'язання запропоновані алгоритми, що мають складність O(n×k×log n) і O(n×k×log k).

3.1.3. Стан питання з дослідження сучасних паралельних обчислювальних систем

Підвищення швидкодії ОС досягається створенням спеціалізованих приладів. Спеціалізовані засоби орієнтовані на розв'язання певного класу або групи задач, що відзначаються спільністю математичного подання. Вони вважаються найбільш ефективними й економічними приладами обробки інформації, ураховуючи їх орієнтацію на виконання алгоритмів певного класу задач.

Є ряд проблем, ефективне вирішення яких можливе тільки на спеціалізованих приладах. Це зумовлено тим, що останнім часом розвиток технології різко знизив вартість елементної бази технічних засобів. Окрім того, визначається тенденція до зростання вартості програм і обслуговування. За оцінками фахівців, середня вартість апаратури і програм для великих і середніх ЕОМ становить 20 і 80 % відповідно [23].

Останнім часом для розв'язання задач управління і при дослідженні різноманітного роду систем широко застосовуються графові моделі. У цих моделях вузлам або дугам графа присвоюються різноманітні вагові коефіцієнти, що по своїй суті є фізичним виразом ваги, вартості, довжини тощо. При використанні графових моделей сформувалася велика група екстремальних задач. Це задачі мережного і багатоетапного планування, комбінаційні задачі, задачі структурного аналізу, задача про потоки, задачі на побудову екстремальних допустимих шляхів та ін. Урахування їх специфіки, а також формулювання мовою теорії графів дають змогу одержувати більш прості методи розв'язання, при цьому часто можуть бути розв'язані практичні задачі таких розмірів, що принаймні на порядок більші, ніж розміри цих же задач у загальному поданні математичного програмування.

Екстремальні задачі на графах відзначаються більшою різноманітністю. Для розв’язання задач цього типу зараз використовуються електронні прилади двох типів: аналогові і цифрові.

Розглянемо аналоговий підхід у моделюванні задач на графах. У його основі лежить принцип екстремальності електричних кіл. Було доведено, що розрахунок електричного кола, складеного з джерел напруги, струму, ідеальних діодів і трансформаторів, еквівалентний визначенню оптимального розв’язання задачі лінійного програмування [24]. Важливим фактом є і те, що будь-яка задача лінійного програмування може бути промодульована електричним колом, подана сполученням перерахованих елементів. Дж. В. Деніс запропонував такі кола, що містять джерела напруги, струму і діоди, з'єднані в різноманітних сполученнях. На підставі принципу екстремальності для електричних кіл з джерелами ЕРС і діодами при вмиканні джерела струму одиничної величини в коло між заданими полюсами його струм пройде по тих гілках, сума величин E_{ij} джерел ЕРС яких буде найменшою для цієї схеми (значення струмів x_{ij} у цих гілках буде дорівнювати 1, а в інших – 0).

Недоліком такої моделі є необхідність мати велику кількість ізольованих одне від одного джерел ЕРС з малим внутрішнім опором.

Цей недолік вдалося обійти в динамічних моделях, у яких регульовані джерела ЕРС замінені конденсаторами постійної ємності.

Перевагою аналогових моделей є висока швидкодія. Але потрібно рахуватися і з притаманними цим моделям принциповими обмеженнями. Це в першу чергу низька точність і складність автоматичного введення-виведення даних, а також значне напруження при розв’язанні задач великої вимірності.

Більш перспективним є напрямок цифрового моделювання задач на графах. Цифровий принцип моделювання у вигляді часової аналогії вперше був запропонований для моделей екстремальних задач про шляхи на графах. Змінною в таких моделях є часовий інтервал, що формується як час затримки електричного сигналу на виході моделі гілки на час появи його на вході. Елементами затримки є регістри та лічильники.

Якщо в структурі, що складається з елементів, які виконують функції затримки, у початковий вузол подамо сигнал, то він, розповсюджуючись і затримуючись у кожній гілці на пропорційну довжинам відповідних гілок величину, потрапляє в потрібну нам точку. Час його затримки пропорційний довжині екстремального шляху.

При розв'язанні поставленої задачі таким способом виникає ряд труднощів, у першу чергу – неможливість ідентифікувати шлях. Для подолання цієї проблеми модель повинна бути значно ускладнена: у неї необхідно ввести елементи, що запам'ятовують та фіксують моменти часу, відповідні початку і кінцю проходження сигналом гілки.

У цифрових моделях задач про екстремальні шляхи, що являють собою багатопроцесорні однорідні паралельні обчислювальні структури, обробка інформації виконана в операціях алгебраїчного складання і вибірки екстремуму серед величин, заданих пропорційними часовими інтервалами.

Переваги цифрового підходу не тільки в розвитку функціональних можливостей, але і в тому, що на цій основі можна забезпечити автоматизацію введення-виведення й автоматизацію формування структури моделі як на початковому етапі, так і в ході самого моделювання.

Однорідні багатопроцесорні системи є ідеальною основою для побудови обчислювальної структури в безперервному технологічному процесі. Завдяки простоті елементів і регулярності їх сполучень на першому етапі в єдиному технологічному процесі виробляється обчислювальне середовище. На другому етапі кожний користувач програмним способом налагоджує обчислювальну структуру на реалізацію будь-якої універсальної або спеціалізованої ЕОМ, максимально пристосованої до особливостей задачі, що розв'язується.

Однорідні обчислювальні системи є зараз найбільш перспективним напрямком обчислювальної техніки. Вони дають змогу зняти неподолані для машин третьої генерації обмеження із забезпечення високої продуктивності для розв'язання складних задач при різкому зниженні вартості.

Однорідна система, названа ідеальною машиною IMM (IDEAL multifunction Machine), повинна складатися з N

ідентичних процесорів, кожний з яких має звичайний набір команд і може працювати асинхронно з іншими процесорами. Вважається, що кількість процесів N може бути настільки великою, як це потрібно для розв’язання задач. Кожний процесор може передавати свої результати будь-якому іншому процесору без втрат часу. Вважається також, що не буде проблем, пов'язаних з пам'яттю, і що індексування, вибірка, запис та інші дії такого роду не займатимуть часу. Отже, машина ІММ є тільки ідеалізованим інструментом для подання програми в паралельній формі [24].

Дещо більш реалістичною концептуальною системою є інтерактивна машина Хоманда. Ця машина складається з приладів (модулів) і має структуру матриці. Кожний модуль містить регістр пам'яті, набір команд і має здатність до комутації. У стандартному варіанті матриця модулів є двовимірною, а кожний модуль безпосередньо пов'язаний з чотирма сусідніми. Кожний модуль може бути в активному або пасивному стані, причому модулі працюють незалежно один від одного. У процесі їх функціонування відбувається передача програм, що виконуються, від модуля до модуля, тобто реалізується концепція «робочих програм, що плавають в морі обладнання». На базі машин Хоманда в комбінації з концепцією матричної обробки розроблена концепція «розподіленої» машини, по вузлах якої розподіляється як можна більша кількість робіт і операцій, що виконуються водночас і незалежно одна від одної. Машина складається з ідентичних блоків, які містять у собі універсальні процесорні елементи і пам'ять найбільшого обсягу. Блоки розбиті на групи. Усередині кожної групи блоки зв'язані між собою. Групи блоків також зв'язані між собою. Паралелізм обробки може бути реалізований як між групами, так і всередині групи. Керуючий блок може управляти роботою як окремих груп блоків, так і зв'язками між групами блоків.

Машина блокового типу складається з великої кількості ідентичних блоків обробки, що становлять матричну конфігурацію машини. Блоки можуть бути зв'язані між собою, причому можуть комутуватися програмними засобами, щоб утворювати різноманітні схеми. У машині можливі два види зв'язку між блоками, які забезпечують шини різноманітного призначення.

Перший використовується для послідовної обробки й очікування доступу до даних у пам'яті блоків, а другий – для зв'язку між блоками. Система характеризується високим ступенем паралелізму обробки і високою надійністю. При відмові одного з блоків диспетчерська програма вилучає його з роботи, а система продовжує обчислення з незначною втратою продуктивності.

У реальних однорідних багатопроцесорних системах набули значного поширення структури трьох типів: з перехресними зв'язками, з множиною шин та із загальною шиною. У системах першого типу N процесорів зв’язані за допомогою матриць з M модулями пам'яті. Модифікація такої схеми у вигляді концептуальної машини запропонована в роботі [23]. Для вибірки команди або слова даних процесор розміщує запит на них у спеціальну секцію. Користуючись цими запитами, кожний модуль пам'яті виконує необхідну операцію, після чого інформація розподіляється по процесорах. У результаті реалізується багатопроцесорний конвеєр операцій (машина Шварца).

Як приклад першого типу машин розглянемо систему D825. Вона є першою модульною системою з ідентичними процесорами і загальними модулями оперативної пам'яті. Ця система була укомплектована операційною системою AOSP (Automatic Operating and Scheduling Program). Ця машина використовувалася в системі управління армії США. Перший зразок системи був випущений у 1962 р. Можна вважати, що зі створенням системи D825 почався етап побудови сучасних багатопроцесорних систем.

До складу системи входять: від одного до чотирьох обчислювальних модулів (ОМ), від одного до шістнадцяти модулів оперативної пам'яті (МОП), один або два комплекти приладів введення-виведення (ПВВ). При максимальній комплектації в систему входить або четвертий ОМ, або другий комплект ПВВ. До комплекту ПВВ входить від одного до десяти модулів управління введення-виведення (МВВ) і від одного до шістдесяти чотирьох периферійних приладів (ПП).

Модульна побудова системи забезпечує високу надійність при безперервному функціонуванні в реальному масштабі часу і можливість зміни складу обладнання без змін програм. Модулі функціонально незалежні і працюють незалежно один від одного.

МОП системи виконані на магнітних сердечниках і мають ємність у 4096 49-розрядних слів, цикл звернення становить 4,33 мкс, а час передачі слова в мегабайтах близько 2 мкс. ОМ мають схеми для виконання арифметико-логічних операцій. Робоча тактова частота – 3 МГц. МВВ являють собою невеликі спеціалізовані ЕОМ, що управляються ОС AOSP. МВВ з'єднані з ПП за допомогою свого матричного перемикача, що забезпечує підключення будь-якого ПП і швидкість передачі інформації в 2 Мбод.

Одним з головних завдань, що виникають при створенні ЕОМ, є підвищення швидкодії. В універсальних програмних автоматах, що реалізують послідовний спосіб виконання операцій, це завдання вирішується за рахунок збільшення тактової частоти. Як відомо, продуктивність обмежена через кінцеву швидкість передачі сигналів по каналах зв'язку між елементами. У зв'язку з цим необхідно зменшувати канали зв'язку, довжини яких обмежуються гранично допустимим рівнем мікроміниатюризації.

Розроблення і виготовлення мікроміниатюрних схем пов'язано з рядом труднощів. У першу чергу це проблеми тепловідводу, сполучень елементів, технологічного виходу виробів тощо. Чи можуть ці проблеми бути вирішеними? У вирішенні цих проблем існують два підходи: технологічний і фізичний.

Як основні фактори при технологічному підході, що обмежують розміри елементів, приймаються і чіткість краю виробу при виготовленні, теплове розсіювання тощо. З урахуванням цих факторів отримана теоретична межа щільності упаковки для більшості електронних приладів − 10 елементів на 1 $см^3$. При цьому гранична продуктивність при послідовному виконанні операції не перевищує 10 операцій за секунду.

При фізичному підході гранична швидкодія оцінюється з огляду на принципи квантування механіки, теорію інформації і реальні можливості побудови елементів. Ця межа становить 10^{-26} с. При побудові складних обчислювальних систем гранично можлива швидкодія буде визначатися взаємним розташуванням елементів. Гранична теоретична швидкодія буде визначатися величинами 10^9–10^{14} операцій за секунду. Таким чином,

використання принципу послідовного виконання операцій призводить до того, що швидкодія обчислювальних приладів має теоретичну межу.

Виникає питання, чи не можна подолати цей бар'єр за рахунок паралельного виконання операцій.

При реалізації принципу паралельного виконання операцій настільки жорстких вимог до розмірів елементів нема. Завдяки паралельному виконанню операцій можна отримати значне збільшення продуктивності, виконуючи водночас декілька операцій на різноманітних приладах. Використання принципу паралельного виконання операцій дає змогу збільшувати швидкодію системи за рахунок збільшення кількості елементів.

При заданні тактової частоти гранична кількість елементів ОС визначається формулою

$$n_p = \rho\alpha\left(\frac{c}{\gamma}\right)^3 \cdot \frac{1}{\nu^3},$$

де ρ − максимально допустиме число елементів в одиниці обсягу;

α − коефіцієнт, що враховує геометричну конфігурацію системи;

С − швидкість світла;

γ − коефіцієнт, що враховує тактову частоту роботи елемента.

Продуктивність роботи при паралельному виконанні операцій прямо пропорційна тактовій частоті ν і кількості елементів n у пристрої

$$P = Bn\nu,$$

де В – коефіцієнт пропорційності.

З наведених формул легко побачити, що гранична продуктивність обернено пропорційна квадрату частоти.

$$n_p = B\rho\alpha\left(\frac{c}{\gamma}\right)^3 \cdot \frac{1}{\nu^2}.$$

Зменшення частоти на один порядок дає змогу збільшити кількість елементів на три порядки й отримати продуктивність вищу на два порядки.

З огляду на це можна зробити висновок, що швидкодія системи при паралельному виконанні операцій переважно визначається кількістю елементів, а не тактовою частотою. А це, іншими словами, говорить про можливість отримати теоретично необмежену швидкодію при паралельній організації обчислень.

За останнє десятиріччя інтенсивно розвивається і багато виробляється ЕОМ і систем, основаних на використанні різноманітних видів паралелізму. Засоби проектування паралельних високопродуктивних ЕОМ спрямовані:

- на підвищення ефективності виконання команд процесора;
- збільшення швидкості передачі даних до виконавчої зони процесора;
- вибір структури апаратних зв'язків з урахуванням топологічної структури задач, що розв’язуються.

Вони визначають різні підходи до побудови ПОС.

Перший підхід пов'язаний з використанням конвеєрної обробки, логіки з попереднім виконанням команд, підвищення галуження, багатофункціональних блоків, з’єднання векторних команд тощо.

Другий підхід пов'язаний з організацією конвеєрної пам'яті, високошвидкісної регістрової та інших видів пам'яті, розширенням каналів передачі даних, що дає змогу прискорити доступ до даних в одиницю часу.

Третій підхід полягає в розробленні структури ОС, спеціально пристосованої для ефективного використання тільки в одному або в дуже обмеженому наборі застосування. У таких машинах виграш у продуктивності досягається завдяки застосуванню асоціативної адресації або виконанню векторних і матричних операцій.

Високопродуктивні паралельні ЕОМ, що характеризуються великою різноманітністю структурних рішень, розрізняються за своїми характеристиками. У першу чергу за продуктивністю, що перебуває в діапазоні від мільйона до десятка мільярдів операцій за секунду [8]. Крім того, ці машини розрізняються за видом і

рівнем паралелізму, засобами обробки даних, засобами управління, що використовуються, обробкою даних і їх призначенням.

Слід зазначити, що паралелізм тією або іншою мірою наявний майже в усіх сучасних цифрових машинах незалежно від їх потужності і типу на різноманітних рівнях (логічному, процесорному, процесорного управління, системної обробки).

Рівень логіки є найнижчим рівнем застосування паралелізму. На цьому рівні забезпечується апаратна реалізація елементарних функцій за допомогою регістрів, буферів, дешифраторів, суматорів, каналів передачі даних та інших операційних блоків, побудованих на базі комбінаційної логіки.

Наступний рівень паралелізму пов'язаний зі здійсненням елементарних функцій за допомогою мікрооперацій. При цьому використовуються два типи примітивних функцій: а) керівні функції та обробка переривань; б) командні функції, що забезпечують алгоритми перетворення інформації. Ці функції реалізуються апаратними і програмними засобами. Тут під паралелізмом розуміється можливість виконання декількох елементарних функцій за один машинний крок. Апаратна реалізація дасть більше можливостей для вибору паралельно логічних функцій, що виконуються на кожному кроці, однак є більш дорогою і менш гнучкою.

Паралелізм на рівні процесорного виконання забезпечується блоками управління процесором, що може бути реалізований, як і попередній, апаратними і програмними засобами.

Найвищим рівнем паралелізму є рівень системної обробки. Паралелізм у цьому випадку забезпечується за допомогою множини процесорних елементів або процесорів, що обробляють відповідну множину потоків даних під управлінням одного або декількох потоків команд.

При проектуванні паралельних ЕОМ слід ураховувати, що ступінь розпаралелювання обчислювального процесу залежить від необхідної ефективності вирішення складних завдань.

У 1958 р. Грош сформулював емпіричний закон, згідно з яким продуктивність ЕОМ зростає пропорційно квадрату її вартості. Цей закон був підтверджений Найтом на близько ста прикладах ЕОМ, перевірений на емпіричних даних машин трьох генерацій.

З другого боку, згідно з гіпотезою Мінського продуктивність пропорційна log m, де m – відношення кількості потоків даних, що обробляються водночас, до кількості потоків команд.

Слід зазначити, що закон Гроша і гіпотеза Мінського висловлюють загальні закономірності, а відповідні їм оцінки характеризують продуктивність у середньому для універсальних машин і типових задач. Для конкретних ОС відхилення оцінок можуть бути значними.

Експериментальні результати показують, що оцінка Мінського занижена. Так, на наборі близько 14 звичайних фортранівських програм для 10-кратного прискорення потрібно близько 35 паралельних процесорів, при цьому досягається завантаження більше ніж 30 %. У багатьох інших випадках прискорення виявилося приблизно пропорційним кількості процесорів.

Що стосується гіпотези Мінського, то існують конкретні приклади, які показують, що завантаженість процесорів може бути значно вищою від log n. У більш досконалих багатопроцесорних системах можна забезпечити високий ступінь завантаженості або за допомогою «мультипрограмування» нового типу, або за допомогою алгоритмів, що добре адаптуються до структури обчислювальної системи. За рахунок цього може бути досягнута і «повна зайнятість» − ситуація, яка взагалі не розглядається Мінським. Однак зусилля для досягнення повної продуктивності слід обмежити, оскільки n-кратне збільшення продуктивності насправді бути не може, тобто існують так звані «загальні втрати». Ці втрати часу в першу чергу пов'язані з необхідністю обміну інформацією між процесорами.

Як основні ознаки класифікації, що характеризують організацію структури і функціонування ОС з погляду паралельності роботи, зазвичай обирають такі:

1) тип потоку команд у центральній частині ОС (ОК – одиничний, МК – множинний);

2) тип потоку даних у центральній частині ОС (ОД – одиничний, МД – множинний) ;

3) спосіб обробки даних у центральних приладах обробки (С – послівний, Р – порозрядний) ;

4) ступінь зв’язаності компонентів ОС (Нз – низький, Вз – високий) ;

5) ступінь однорідності основних компонент ОС (Ор – однорідний, Нр – неоднорідний) ;

6) тип внутрішніх зв'язків в ОС (Кн – «канал–канал»; Пм – через загальну зовнішню пам'ять; Пр – безпосередньо між процесорами; Зш – через загальну шину з розподілом за часом; Мш – через множини шин при використанні багатовхідних моделей оперативної пам'яті; Пк – з перехресними зв'язками за допомогою матричного комутатора).

Особливо серед паралельних обчислювальних систем можна виділити систолічні масиви, що являють собою гранично просту комунікаційну мережу. Загальний підхід до конструювання систолічних масивів оснований на такій ідеї. Нехай у нашому розпорядженні є достатньо велика кількість функціональних приладів, що реалізують операції одного або декількох типів. Будемо називати ці функціональні прилади систолічними комірками (процесорними елементами, чипами). Конструктивно кожна систолічна комірка виконується однотипно у вигляді деякого плоского багатокутника, причому входи виведені на його межі. Тепер почнемо складати з багатокутників різноманітні фігури, приєднуючи без накладення послідовно багатокутника за багатокутником і припускаючи, що сполучені багатокутники торкаються один одного. Якщо в місцях торкання з'єднати входи і виходи сусідніх систолічних комірок, то отримана обчислювальна система при деяких додаткових умовах називається систолічним масивом. Систолічні комірки завжди влаштовані дуже просто, особливістю їх є те, що вони не мають особистої пам'яті для зберігання результатів проміжних обчислень, хоча вони можуть мати пам'ять для виконання операцій. Тому без великої втрати спільності можна вважати, що систолічні комірки являють собою конвеєрні функціональні елементи, а систолічний масив є конвеєрним обчислювачем.

Незважаючи на те, що систолічний масив є конвеєрним обчислювачем, неможливо на основі лише цього факту виявити всі особливості обчислювальної системи цього типу. Для цього потрібно взяти до уваги специфічні характеристики структури систолічних масивів і відобразити їх у специфіці графа конвеєрного обчислювача.

Зі стислого огляду стану питання і наведених ознак класифікації ПОС, що характеризують їх функціонування, видно, що найбільш важливим з елементів у створенні ПОС є середовище обміну інформацією між процесорними елементами системи. Усередині середовища обміну фактично закладається ідеологія організації паралельних обчислень. Сучасні ПОС, як правило, вузькоспеціалізовані. Складність створення проблемно-орієнтованих ПОС зумовлена невирішеністю цілого комплексу проблем, які визначають взаємозв'язок між архітектурою ПОС і структурою паралельних алгоритмів, призначених для реалізації на цих ПОС. Особливо гостро ця проблема виникає при розв’язанні так званих «сильно пов’язаних» задач, розбиття яких на підзадачі призводить до необхідності інтенсивних обмінів на кожному кроці розв’язання окремих підзадач. Саме до такого класу задач належать оптимізаційні задачі, які потрібно багато раз розв’язувати в процесі прийняття рішень. Складність створення ПОС, які орієнтовані на широкий клас оптимізаційних задач, пов'язаний ще з великою вимірністю оптимізаційних задач, що розв’язуються, і досить жорсткими вимогами до оперативності прийняття рішень при управлінні складними системами.

3.1.4. Паралельні обчислювальні системи й абстрактні машини

Ми розглядали в основному моделі обчислень на базі потоку управління, потоку даних, потоку запитань.

Наступним, більш детальним, рівнем є абстрактні машини, що реалізують моделі обчислень. Найбільш детальний рівень становлять самі машинні реалізації. Вони подаються програмісту через мову асемблера.

Абстрактні машини містять абстрактні функціональні прилади чотирьох типів:

1) процесор команд ПрК, що здійснює інтерпретацію команд відповідно до моделі обчислень;

2) процесор даних ПрД, що здійснює перетворення даних з використанням базових арифметичних операцій;

3) ієрархічна пам'ять ПМ, що здійснює зберігання інформації і передачу її процесорам і від них. Вона функціонально поділяється на пам'ять команд ПК і пам'ять даних ПД, що конструктивно у різних машинних реалізаціях можуть бути як суміщеними, так і роздільними;

4) комутатор К, що здійснює зв'язування між собою перерахованих функціональних приладів.

Таким чином, якщо модель обчислень може охоплювати декілька класів архітектури, що її реалізують, то абстрактні машини відображають класи архітектури, але без детальної індивідуалізації всередині класів, притаманної машинним реалізаціям. Розглянемо абстрактну машину на базі потоку управління (рис. 3.1). Процесори працюють синхронізовано при підтримці приладів пам'яті, що реалізують звертання за рахунок ієрархії структурної побудови (регістри, кеш, операційна пам'ять, диски та інші прилади). При цьому схеми роботи ПрК і ПрД мають вигляд, показаний на рис. 3.2 і 3.3.

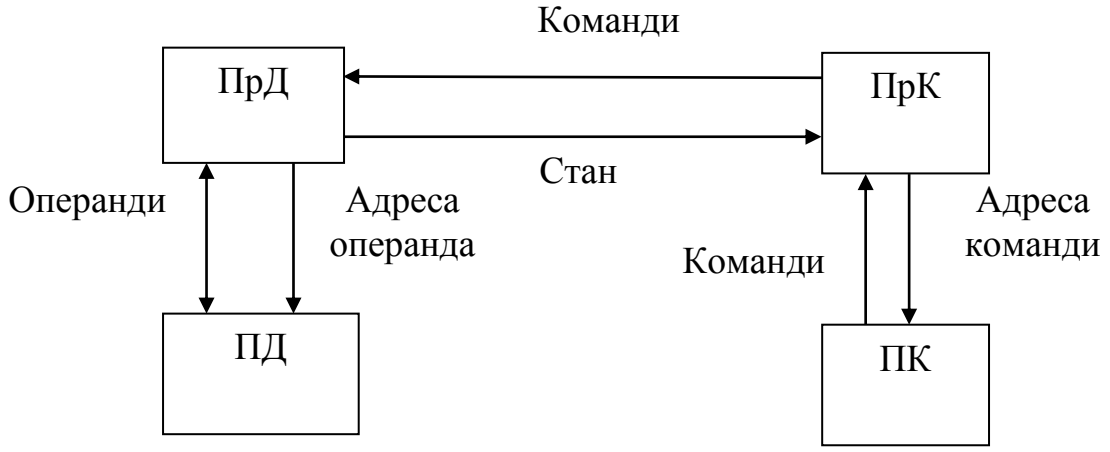

Рис. 3.1. Структура абстрактної машини

Для підвищення продуктивності вихідну архітектуру можна модифікувати у рамках її класу, сумістивши у часі ті або інші дії приладів (наприклад, відсилання процесором даних інформації про стан процесора команд і результат операції у пам'ять даних можна виконувати одночасно).

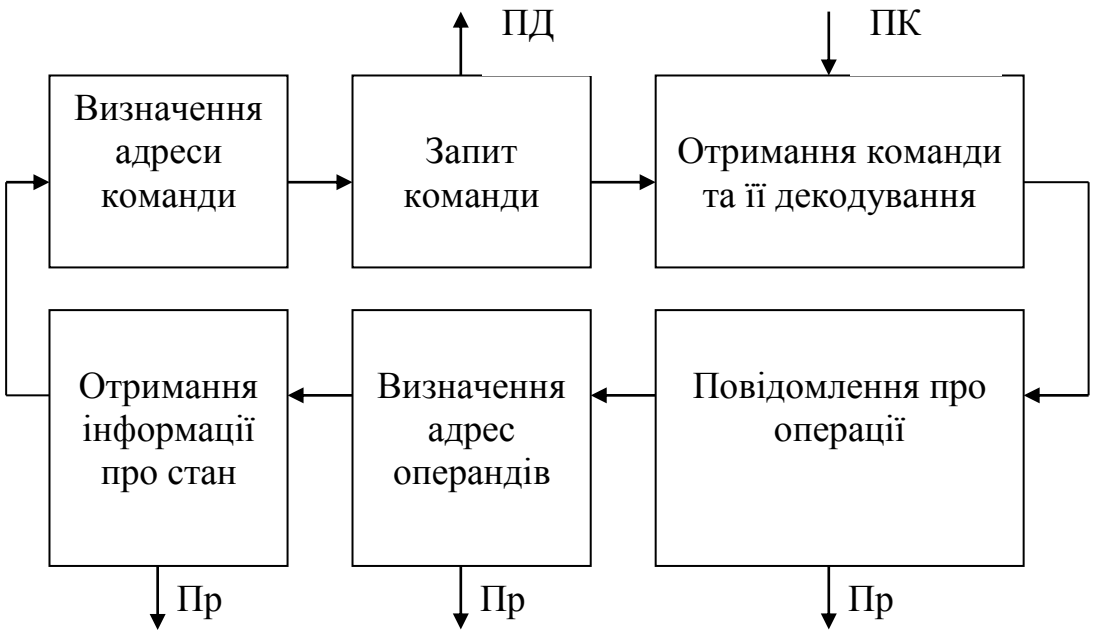

Рис. 3.2. Схема роботи процесора команд

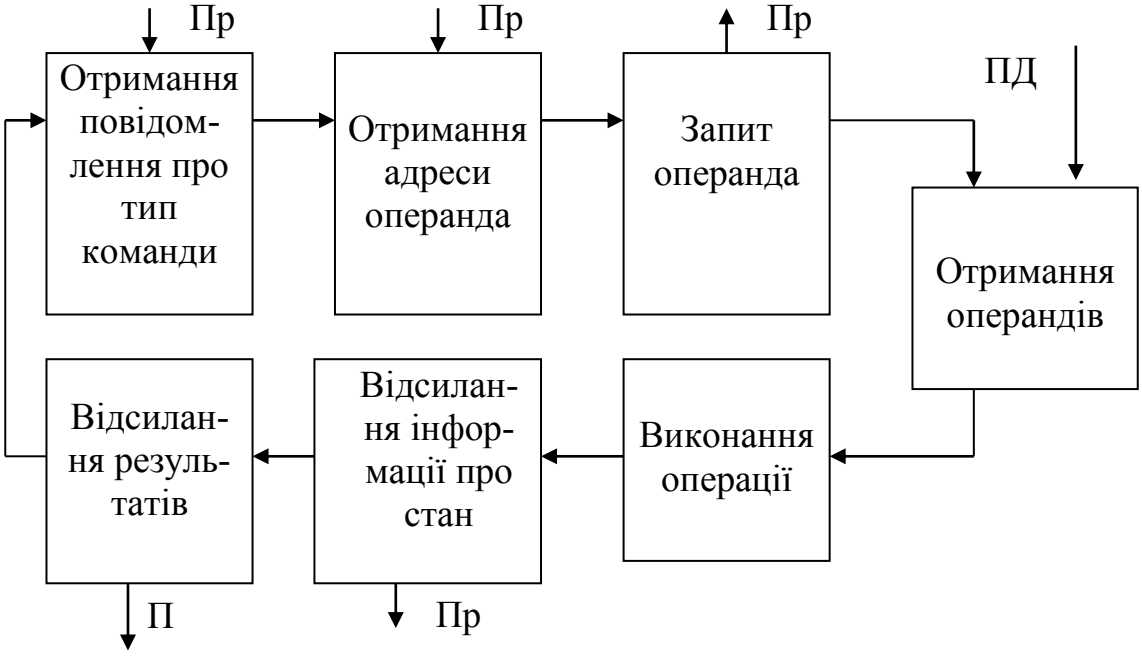

Рис. 3.3. Схема роботи процесора даних

Більш висока продуктивність за рахунок архітектури досягається її подальшим удосконаленням шляхом побудови конвеєрних і матричних архітектур, що дадуть більш високе

суміщення у часі. Для таких архітектур потрібні комутатори. Можна виділити чотири типи абстрактних комутаторів:

1) l - 1: єдиний функціональний прилад одного типу зв'язаний з єдиним функціональним приладом іншого типу. Фізичний зв'язок може організовуватися через одну або різні лінії. Потік інформації може бути в обидва боки;

2) n - n: n раз повторюється копія комутатора типу 1 - 1 для двох наборів з n функціональних приладів у кожному $(i = \overline{1,m})$;

3) l × n: один функціональний прилад зв'язаний зі всіма n приладами іншого набору функціональних приладів;

4) n × n: забезпечується зв'язування кожного приладу одного з наборів зі всіма приладами цього або іншого набору.

3.1.5. Типи абстрактних машин

Матричний процесор можна уявити як абстрактну машину з єдиним процесором команд, єдиною пам'яттю команд, великою кількістю процесорів даних і приладів пам'яті даних, у якій процесор команд зв'язаний з процесором даних через комутатор типу 1 × n. При цьому процесори даних можуть бути зв'язані з приладами пам'яті даних через комутатор типу n - n і між собою через комутатор типу n × n (машина типу 1, рис. 3.4) або ж можуть бути зв'язані з приладами пам'яті даних через комутатор типу n × n (машина типу 2, рис. 3.5).

Наведемо приклади сильно- і слабкозв'язаних багатопроцесорних систем.

У сильнозв’язаній системі зв'язок між процесорами даних і приладами пам'яті даних здійснюється через динамічний комутатор типу n × n, які реалізують функціонально загальну пам'ять даних зв'язку між обчислювальними процесами за допомогою загальних глобальних змінних. Процесори даних можуть мати власну пам'ять. Пам'ять команд може бути і загальною (BBN Butterfly) (рис. 3.6).

Типова слабкозв'язана багатопроцесорна система подана на рис. 3.7.

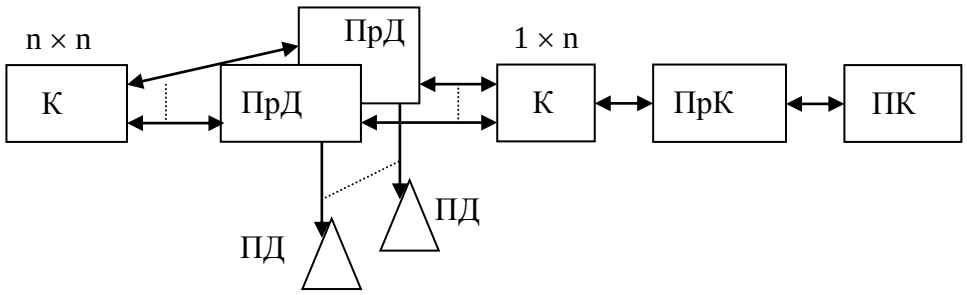

Рис. 3.4. Абстрактна матрична машина типу 1

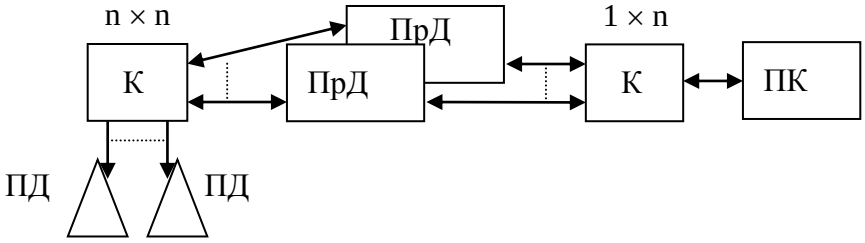

Рис. 3.5. Абстрактна матрична машина типу 2

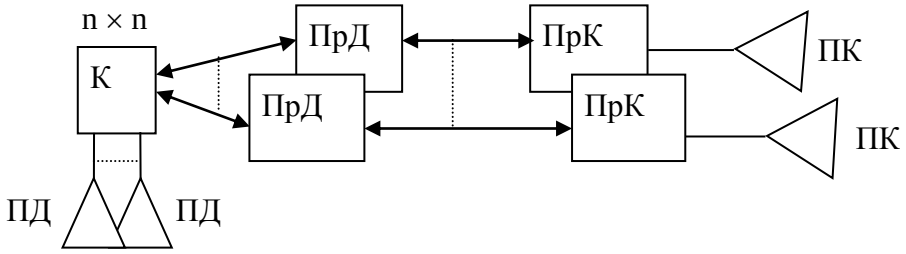

Рис. 3.6. Абстрактна сильнозв'язана багатопроцесорна машина

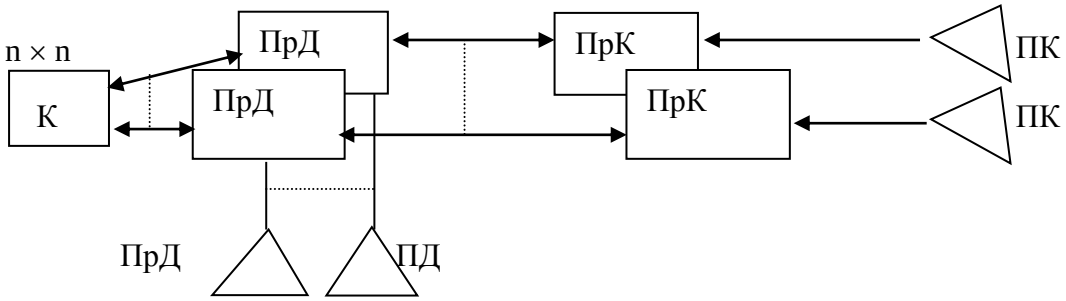

Рис. 3.7. Абстрактна слабкозв'язана багатопроцесорна машина

Зв'язок між процесорами даних і приладами пам'яті даних здійснюється через комутатор типу n - n, а між процесорами – через комутатор типу n × n за допомогою запитів одного процесора до іншого, шляхом передачі повідомлень або ж зовнішнього виклику процедур (AMETEK 14 і 2010).

Машини на базі потоку даних не мають механізму для упорядкування проходження команд, оскільки у них паралельний порядок виконання операцій визначається залежностями за даними. Вони не мають процесорів і пам'яті команд як таких. Набір процесорів даних може бути організований двома способами.

Кожний ПрД має власну пам'ять, а обмін даними здійснюється через комутатор типу n × n (рис. 3.8).

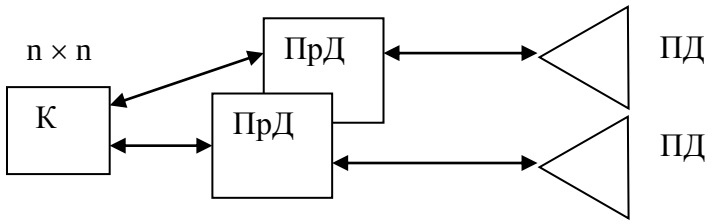

Рис. 3.8. Абстрактна машина типу 1

У граничному випадку кожний процесор використовує оператори тільки одного типу і ми приходимо до абстрактної машини типу 2 (рис. 3.9).

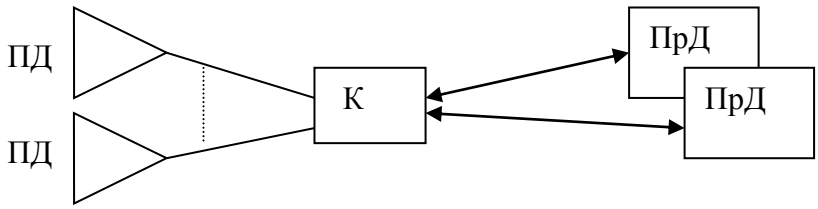

Рис. 3.9. Абстрактна машина типу 2

У цій абстрактній машині на базі потоку даних є загальні прилади пам'яті, з якими через комутатори (n × n) зв'язані процесори даних. Цей підхід більш розповсюджений у машинних реалізаціях.

Розглянуті типи абстрактних машин дають змогу побудувати формалізовану класифікацію архітектури.

Виділяють класи:

1 - 5 – машини для потоку даних;

6 – однопроцесорні машини на базі потоку управління ОКОД;

11 - 12 – машини типу МКОД;

13 - 28 – багатопроцесорні машини класу МКМД (починаючи з 21 до 28 процесори команд зв'язані з процесорами даних через комутатор типу n × n).

3.1.6. Шляхи подальшого розвитку паралельних обчислювальних систем для автоматизації процесу прийняття рішень

В автоматизованих системах управління для виконання найбільш складних розрахунків при розв’язанні задач управління створюються суперпроцесори різноманітних орієнтацій: матричні, мікроконвеєрні, поточновекторні та ін., до яких висуваються вимоги, які значною мірою відрізняються від вимог, що висуваються до звичайної обчислювальної техніки. Для комп'ютеризації цієї галузі необхідне створення ефективних і надійних засобів контролю й управління процесами, що швидко змінюються, збільшення інтелекту в приладах зв'язку, зменшення габаритів і потужності, що споживається. У перспективі ставиться завдання створення на базі кібернетичних систем засобів, що якнайповніше заміщають людину. Необхідність здійснювати управління процесами, що швидко змінюються, при управлінні залізничним транспортом, зростання швидкості руху поїздів і обсягів перевезень призвело до розширення діапазонів сигналів, що надходять від датчиків, і зменшення часу, що відпускається на їх обробку. Управління такими об'єктами і процесами висуває до технічних засобів настільки високі вимоги з оперативності, продуктивності й ефективності, що вони не

забезпечуються в рамках традиційних засобів перетворення та обробки інформації і принципів побудови універсальних ЕОМ [16].

Створення ефективних засобів високої оперативності і продуктивності, що забезпечують перетворення та обробку інформації в реальному масштабі часу, потребує розроблення теоретичних основ їх побудови, у тому числі розгляду процесів перетворення обробки інформації з погляду суміщення їх у часі, питань спеціалізації або проблемної орієнтації технічних засобів, розпаралелювання обчислювальних процесів, використання засобів апаратної реалізації алгоритмів.

Аналіз основних тенденцій розвитку паралельних обчислювальних систем показує, що тут можна виділити такі напрямки розвитку ПОС [6]:

1) створення паралельних обчислювальних систем для вирішення завдань первинної обробки на нижньому рівні систем, що зводяться до масштабування, помноження сигналів, функціональних перетворень, диференціювання, інтегрування, згладжування, фільтрації;

2) розроблення проблемно-орієнтованих паралельних обчислювальних систем для розв’язання задач лінійної алгебри, а також диференціальних рівнянь у частинних похідних (найбільших успіхів у їх створенні як у теоретичному, так і в практичному плані досягли фахівці інституту кібернетики імені В. М. Глушкова АН України);

3) розроблення паралельних систем для розв’язання задач комбінаторної оптимізації і теорії графів. Цей напрямок є найбільш складним і найменш розробленим на сьогодні, а також є найбільш важливим з погляду створення засобів автоматизації процесу прийняття рішень, оскільки задачі цілочислового програмування, оптимізації на графах, а також варіаційного числення є основою формальних моделей систем прийняття рішень.

Особливе місце посідає булеве програмування, що є формальною моделлю широкого класу задач теорії прийняття рішень, пов'язаних з виділенням неупорядкованої підмножини «кращих об'єктів».

Найпростіша модель такого типу має вигляд [66]

$$L = \sum_j \alpha_j x_j \to \max \tag{3.1}$$

при обмеженнях

$$\sum_j \beta_j x_j \le b_j \quad i = \overline{(1,m)}; \quad j = \overline{(1,n)} \quad x_j \in \{0,1\}. \tag{3.2}$$

У багатьох практичних ситуаціях у процесі прийняття рішень потрібно не тільки відібрати об'єкти, але й вибрати для кожного відібраного об'єкта деякий варіант реалізації, що характеризується індивідуальними ефективністю і ресурсними обмеженнями. Це приводить до моделі сходового типу [15]

$$\sum_{i=1}^{n}\sum_{j=1}^{m} \alpha_{ij} x_{ij} \to \max \tag{3.3}$$

при обмеженнях

$$\begin{aligned} &\sum_{i=1}^{n}\sum_{j=1}^{m} \beta_{ij} x_{ij} \le b; \quad \sum_{j=1}^{m} x_{ij} \le 1; \quad j = \overline{(1,m)}; \\ &x_{ij} \in \{0,1\}; \quad \alpha_{ij} \ge 0; \quad \beta_{ij} \ge 0; \quad b \ge 0; \quad i = \overline{(1,n)} \end{aligned} \tag{3.4}$$

або до моделі блокового типу, у якій друге обмеження (3.4) має вигляд

$$\sum_{j=1}^{l_i} x_{ij} = 1.$$

Динамічне програмування, розроблене Р. Белманом, основане на принципі оптимальності, що використовується на практиці як спосіб послідовного аналізу варіантів В. С. Михалевича [14] і локальних варіацій Н. Н. Моісєєва. Задачі цілочислового динамічного програмування в загальному випадку можна подати в такому вигляді:

$$f(\bar{x}) = f_1(x_1) T f_2(x_2) T ... T f_n(x_n) \to \max \qquad (3.5)$$

при обмеженнях

$$\sum_{j=1}^{n} \varphi_j(x_j) \le b; \quad x_j \in D_j; \quad j = \overline{(1,n)}, \qquad (3.6)$$

де b – ціле число;

T – символ додавання, при якому функція f(x) сепарабельно-адитивна, або символ множення, при якому функція f(x) сепарабельно-мультиплікативна;

$\varphi_j(x_j)\ (j = \overline{1,n})$ – функції, що збережуть цілочисельність;

D_j – деяка множина цілих чисел.

Практично будь-яка задача динамічного програмування може бути зведена до задачі визначення найкоротших шляхів з вершини s до t на графі G(V, E) [19]. Остання може бути подана в такому вигляді:

$$\sum_i \sum_j c_{ij} f_{ij} \to \min; \qquad (3.7)$$

$$\sum_{j \in V} f_{ij} \le 1; \quad f_{ij} - \text{одиничний потік по } (i, j); \ f_{ij} \ge 0; \ (i, j) \in E; \qquad (3.8)$$

$$\sum_{j \in V} f_{ij} \ge 1; \qquad (3.9)$$

$$c_{ij} - \text{витрати } (i, j); \qquad (3.10)$$

$$\sum_j f_{ij} - \sum_j {}_{ij} = 0; \ i \ne s; \ j \ne t. \qquad (3.11)$$

Серед варіаційних задач для управління технікою важливе значення має задача оптимального управління [24]. Оптимальне управління може розглядатися як узагальнене варіаційне числення. Усі задачі варіаційного числення пов'язані з максимізацією або мінімізацією інтегралів.

$$P = \int_{x_0}^{x_k} G[y_1(x), y_2(x) \ldots y_n(x); \quad y_1(x), y_2(x) \ldots y_n(x)]dx. \qquad (3.12)$$

При відповідних обмеженнях і граничних умовах ці задачі можуть бути сформульовані як задачі оптимального управління, якщо зробити заміну

$$x \equiv t, \qquad x_0 \equiv t_0, \qquad x_k \equiv t_k;$$

$$y_i(x) = x_i(t); \qquad \frac{dx_i}{dt} = u_i(t) \equiv y_i(t); \qquad i = \overline{1,n}. \qquad (3.13)$$

Підстановка співвідношень (3.13) у функціонал (3.12) зводить задачу варіаційного числення до задачі оптимального управління з функціоналом

$$P(t) = \int_{t_0}^{t_k} G\,(x_1, x_2, \ldots, x_n \;;\; u_1, u_2, \ldots, u_n)dt, \qquad (3.14)$$

де

$$\frac{dx_i}{dt} = u_1 \quad (i = \overline{1,n}). \qquad (3..15)$$

Варіаційні задачі ефективно модулюються на ґратках. Моделювання складається з двох етапів: на першому етапі здійснюється перетворення задачі (3.14)–(3.15) в задачу (3.7)–(3.11), вага ребер у яких заздалегідь обчислюється на ЕОМ; на другому етапі розв’язується задача (3.8)–(3.11).

Тому для забезпечення автоматизації процесу прийняття рішень у масштабі реального часу потрібне розв’язання такого комплексу задач:

1) розроблення методології побудови обчислювальних систем, що дають змогу забезпечити розв'язання задач прийняття рішень на управління складними системами в масштабі реального часу, яка містить у собі:

- розроблення архітектури адаптивних паралельних обчислювальних систем циклічного типу;

- розроблення паралельних алгоритмів для реалізації основних формальних моделей теорії прийняття рішення, а саме: алгоритмів розв'язання задач визначення найкоротших шляхів, оптимального управління, динамічного програмування, лінійного програмування з булевими змінними і задач оптимізації на графах;

2) розроблення спеціалізованих обчислювальних систем для реалізації запропонованих паралельних алгоритмів розв'язання задач теорії прийняття рішень;

3) аналіз можливості застосування розроблених алгоритмів і обчислювальних структур для оптимального планування;

4) розроблення паралельної архітектури проблемної обчислювальної системи, що орієнтована на забезпечення автоматизації процесу прийняття рішень у реальному масштабі часу;

5) проведення порівняльного аналізу розроблених паралельних алгоритмів і обчислювальних систем для їх реалізації з відомими.

3.2. Архітектура паралельних обчислювальних систем циклічного типу для систем штучного інтелекту

3.2.1. Показники ефективності паралельних алгоритмів і паралельних обчислювальних систем

Для коректного порівняння алгоритмів, що розробляються, і ПОС з відомими необхідно визначити основні показники їх ефективності, що дають змогу проводити порівняльний аналіз. Найбільш широко застосовуваним показником ефективності роботи алгоритму є відрізок часу, який витрачає алгоритм для розв'язання поставленої задачі. Однак такий підхід неоднозначно визначає ефективність алгоритмів, оскільки цей показник залежить від типу ЕОМ, на якій виконується алгоритм. При аналізі алгоритмів у цій роботі час роботи алгоритмів буде визначатися в термінах кількості елементарних кроків (арифметичних операцій, порівнянь, передачі тощо), необхідних для виконання цього алгоритму на гіпотетичній обчислювальній машині. Іншими словами, виконання будь-якої операції потребує одну одиницю часу.

Однак кількість кроків алгоритмів, що виконуються, неоднакова для різного розміру входу вхідних даних. Для врахування таких неоднозначностей у поведінці алгоритму при переході від одного входу до іншого будемо розглядати всі входи цього розміру n разом і визначимо складність алгоритму для цього розміру входу як кількість кроків алгоритму в гіршому випадку по всіх цих входах. У комбінаторних задачах оптимізації входом є комбінаторний об'єкт: граф, множина цілих чисел (можливо упорядкованих у вигляді векторів або матриць), сімейство кінцевих множин тощо. Щоб ввести цей вхід в обчислювальну систему для розв’язання, необхідно якимось чином закодувати або подати його у вигляді послідовності символів над деяким фіксованим алфавітом, таким як подвійний алфавіт. Ми не будемо визначати, як кодуються комбінаторні об'єкти послідовності символів. Ці кодування можна виконувати будь-яким з багатьох відомих способів [19]. Оскільки ми домовилися, що вхід алгоритму подається у вигляді послідовності символів, визначимо розмір входу як довжину цієї послідовності, тобто кількість символів у ній. При аналізі алгоритмів звичайно цікавить швидкість зростання складності алгоритму, яку описують за допомогою нижченаведених формалізмів [19].

Визначення. Нехай f(n) і g(n) – функції, визначені на множині цілих позитивних чисел і які набувають додатні дійсні значення:

$$f(n) = O(g(n)), \tag{3.16}$$

якщо існує така константа $C > 0$, що $f(n) \leq Cg(n)$ для достатньо великих n.

$$f(n) = O(g(n)), \tag{3.17}$$

якщо існує така константа $C > 0$, що $f(n) \geq Cg(n)$ для достатньо великих n.

$$f(n) = \theta(g(n)), \tag{3.18}$$

якщо існують такі константи C' > 0, що C g(n)≤f(n)≤C'g(n) для достатньо великих n. Замість виразу (3.18) можна записати f (n) (g (n)).

Відношення (3.18) є відношенням еквівалентності. Клас еквівалентності відносно цього співвідношення, що містить f(n), тобто множина всіх функцій g(n), таких, що f(n) = θ (g(n)), називається швидкістю зростання f(n) [12].

Завдяки введеному поняттю швидкість зростання складності алгоритму, як показано в роботі [24], можна оцінити зверху, використовуючи вираз типу «вимагає часу $O(n^3)$».

Таким чином, часова складність алгоритму відображає витрати часу, які потрібні для нього. Це функція, у якій кожній вхідній довжині n відповідає максимальний час, який витрачає алгоритм на розв’язання індивідуальних задач цієї довжини (за всіма індивідуальними задачами довжини n). Природно, що ця функція не буде повністю визначена доти, доки не зафіксована схема кодування, вибрано обчислювальний пристрій (або його модель), потрібний час роботи. Один і той самий алгоритм А можна виконувати за різний час на різних структурах. При цьому при переході від однієї структури до другої часто використовуються поняття ємності складності обчислювальної системи [19], що характеризує зростання апаратних витрат при такому переході. Апаратні витрати можуть задаватися кількістю елементарних процесорних елементів, кількість зв'язків у системі, комірок пам'яті тощо. При цьому використовується вираз "потрібно $O(n^2)$ комірок пам'яті" [19].

При аналізі складності алгоритмів оцінка в найгіршому випадку не завжди об'єктивно несе інформацію про час, який витрачає алгоритм на розв’язання задачі, оскільки гірші випадки при розв’язанні конкретної задачі цим алгоритмом, можуть бути рідкісними, тому в роботі для кожного алгоритму теоретично визначена складність у найгіршому випадку, а при експериментальному моделюванні роботи алгоритмів наводиться оцінка складності алгоритмів у середньому, що дає змогу більш об'єктивно оцінити можливості алгоритму. Оскільки при аналізі алгоритмів, що розробляються, і обчислювальних структур у роботі широко застосовуються графи, то зупинимося більш докладно на питанні: що таке розмір графа.

Граф можна подати багатьма способами. Наприклад, будь-який граф G (V, E) можна зіставити з його матрицею суміжностей

AG = a_{ij} розміру $|V \times V|$, у якій $a_{ij} = 1$, якщо $(\nu_i, \nu_j) \in E$, і $a_{ij} = 0$ в іншому випадку. Одним з корисних способів подання є списки суміжності. При цьому для кожної вершини $\nu \in V$ виписується множина $A(\nu) \subseteq V$ вершин, суміжних з ν. Для розпізнання вершин графа використовуються індекси. Оскільки в нас $|V|$ вершин, то для цього нам потрібно приблизно $\theta(\log|V|)$ подвійних (або десяткових) розрядів. Отже, для подання графа G(V, E) потрібно $\theta(|E|\log|V|)$ символів, і $|E|$ є розумною апроксимацією розміру графа, і аналіз складності алгоритмів на графах з використанням $|E|$ як параметра практично прийнятний.

Природно, $|E|$ і $|V|$ зв'язані нерівністю

$$|E| \leq |V|\,|V-1|/2. \qquad (3.19)$$

Слід мати на увазі, що $|E|$ може дуже сильно змінюватись, роблячи граф насиченим (коли $|E| = \theta(|V|^2)$) або розрідженим (коли $|E|$ набагато менше, ніж його максимальне значення). Отже, алгоритм з оцінкою $O(|E|^3)$ може бути кращим, ніж алгоритм з оцінкою $O(|V|^3 \times |E|)$, коли граф розріджений, тоді як для насичених графів виправданий протилежний вибір. При побудові паралельних швидкодійсних обчислювальних систем іноді за час потрібно платити певним апаратним переповненням. При цьому необхідно виходити з того, що спроектована обчислювальна структура повинна задовольняти обмеження можливості її технічної реалізації при існуючих технологіях і елементній базі.

3.2.2. Поняття циклічної паралельної обчислювальної системи та організація в ній обчислень

Математичний аналіз принципів побудови супер-ЕОМ та інших високопродуктивних систем значною мірою ускладнюється великою їх різноманітністю, численними погано введеними поняттями і відсутністю формалізованих принципів дослідження. Незважаючи на велику кількість досліджень у цьому напрямку, картина, пов'язана з організацією паралельних обчислень, архітектурою паралельних обчислювальних систем для різноманітних типів задач, не прояснилася. Найбільш цікавими у

цьому напрямку є роботи, виконані під керівництвом Г. І. Марчука, а також публікації В. В. Васильєва і Є. А. Ралдугіна [24]. Слід виділити монографію В. В. Воєводина [18], у якій зроблено спробу об'єднати дослідження обчислювальних засобів та архітектури обчислювальних систем у єдиний процес і розглянути питання відображення проблем обчислювальної математики й архітектури обчислювальних систем. У роботі [18] зроблено песимістичний висновок про те, що немає жодних підстав сподіватися на отримання загальної задачі відображення проблем математики на архітектуру обчислювальних систем за допомогою деякого універсального способу. Висновок базується на тому, що алгоритми можна подавати як ациклічні орієнтовані графи, а ряд задач, пов'язаних з аналізом можливості ефективної реалізації довільного алгоритму A_1, якому відповідає граф G_1, на обчислювальній системі заданої графом G_2, належать до NP-повних задач. У цій роботі зроблено спробу розглянути різноманітні концепції побудови паралельних обчислювальних систем, як проблемних, так і універсальних, на базі циклічних паралельних орієнтованих обчислень.

Алгоритм можна подати у вигляді графів, при цьому природно припустити, що алгоритм і правило, що його описує, дають змогу визначити:

- множину змінних, у перетворенні яких полягає реалізація алгоритму;
- множину операцій, які виконуються в процесі реалізації алгоритму;
- відповідність, яка показує, що результати виконання попередніх операцій, є аргументами для кожної наступної операції.

Нехай задано деякий алгоритм А. Його можна подати у вигляді підмножини алгоритмів $\{A_j\}$, $(j = 1, n)$, які необхідно виконати в певній послідовності, що задана графом G(V, E). В останньому кожна вершина відповідає алгоритму A_j, а кожне ребро між вершинами і та j, направлене від вершини і до j, існує лише в тому випадку, якщо алгоритм A_j можна виконати тільки після реалізації алгоритму A_i. Наприклад, на рис. 3.10, а наведена структурна схема деякого алгоритму А, задана у вигляді графа G(V, E), з якого видно, що для його реалізації спочатку потрібно виконати алгоритм A_1. Результати роботи цього алгоритму є вхідними даними для виконання алгоритмів A_2, A_3, A_4. Після їх виконання реалізується алгоритм A_5, алгоритм A_6

відпрацьовується після алгоритмів A_3 і A_4, а A_7 – після A_5 і A_6. Очевидно, що кількість різноманітних подань графів алгоритму А визначається множиною способів, що використовуються для розв'язання задачі. У принципі, граф G (рис. 3.10, а) можна розглядати як структуру програмного виробу, що реалізує алгоритм А. Тоді $\{A_j\}$ – множина програмних модулів, з яких складається програма реалізації алгоритму А. Очевидно, що якщо в кожну вершину графа G (рис. 3.10, а) помістити процесор, що орієнтується на виконання алгоритму A_j, то ми отримаємо обчислювальну систему, що дасть змогу реалізувати алгоритм А.

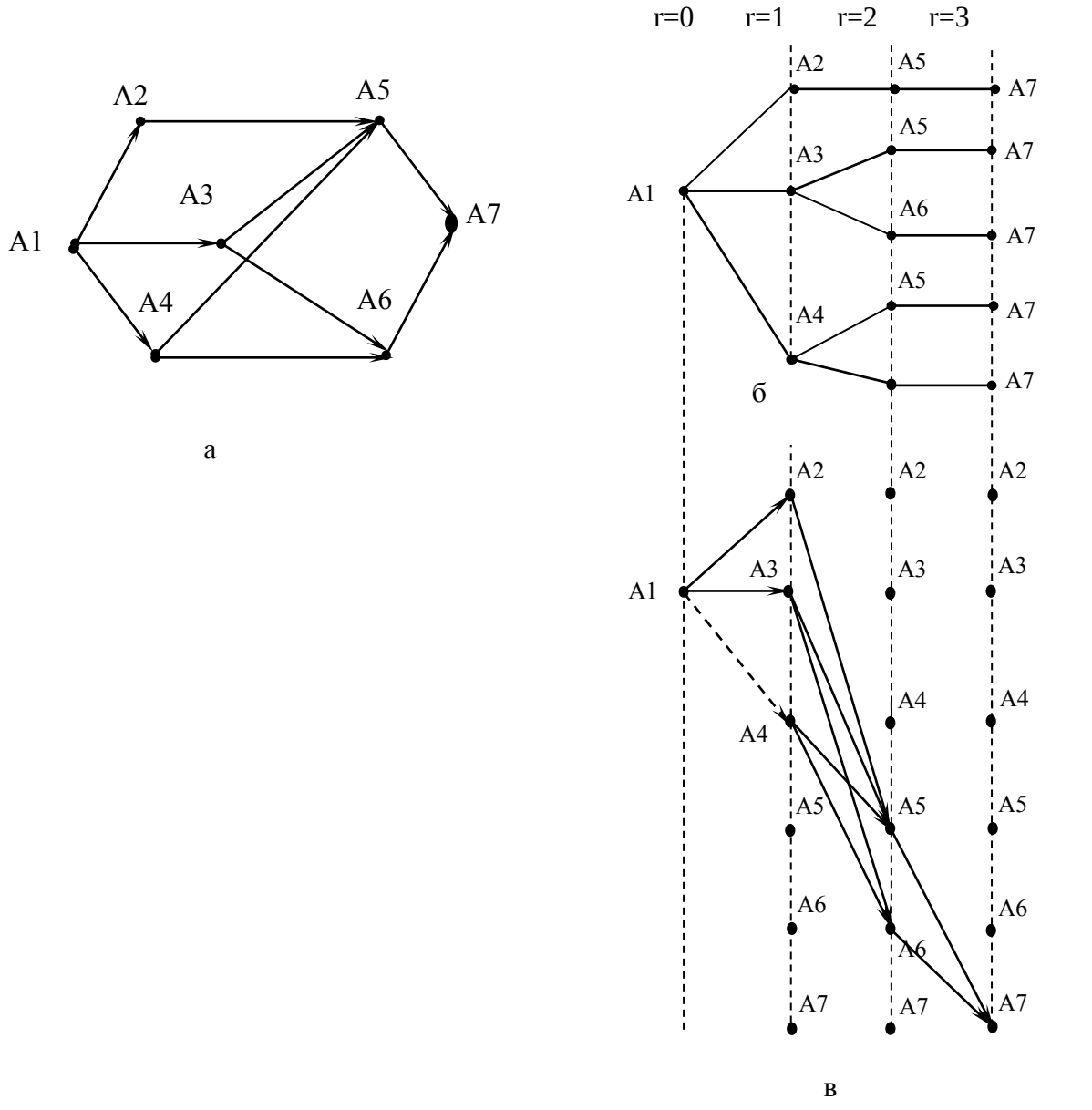

Рис. 3.10. Структурна схема алгоритму А

Припустимо, що кожний процесор $П_j$ виконує алгоритм A_j відразу, як тільки на його вхід надійдуть усі необхідні дані. Позначимо t_j – час реалізації кожного з алгоритмів A_j на відповідному процесорі $П_j$. Тоді, якщо присвоїти всім вершинам графа G(V, E) ваги t_j, мінімальний час реалізації алгоритму A не може перевищити довжину критичного (найдовшого) шляху M_{17} у графі G. Очевидно, що така обчислювальна структура ідеальна для цього подання графа алгоритму A, оскільки вона забезпечує розв'язання задачі за мінімальний час при максимальному розпаралелюванні процесів обчислень. Ранг критичного шляху (кількість ребер у шляху) в графі G(V, E), що відповідає деякому алгоритму A, будемо називати глибиною алгоритму A, а максимальну кількість алгоритмів $A_j \in A$, які можна водночас виконати на довільному r-му кроці реалізації алгоритму A, – шириною алгоритму. У подальшому глибину і ширину довільного алгоритму позначимо літерами a і h. Припустимо, що кожний з алгоритмів $\{A_j\} \in A$ реалізується на будь-якому процесорі $П_j$.

Нехай задана структурна схема алгоритму A графом G(V, E). Виникає задача визначення оптимальної стратегії розподілу алгоритмів $\{A_j\}$ між процесорами $\{П_j\}$, при якій час розв'язання задачі відрізняється від довжини критичного шляху в графі G(V, E) на мінімально можливу величину, вибрана стратегія їх використання і повинна визначити архітектуру обчислювальної системи. Якщо ширина алгоритму дорівнює h, то природно спробувати обмежитися h процесорами для реалізації алгоритму A.

Для цього подамо граф G(V, E) алгоритму A у вигляді стягнутого дерева всіх шляхів. Дерево всіх шляхів D графа G(V, E) (рис. 3.10, а) побудовано від першої вершини, як показано на рис. 3.10, б. Воно будується за матрицею суміжності $B = \| b_{ij} \|$ від деякої вершини S таким чином:

- позначимо вершину S і беремо S-й рядок матриці B;
- з цього рядка виберемо номери вершин j_1, для яких $b_{sj} \neq 0$, і утворимо множини вершин, що мають від вершини S шлях рангу r = 1 (вершини першого ярусу);
- вершини r-го ярусу зі шляхами рангу r від вершини S перебувають під черговим переглядом рядків вершин j_{r-1} в (r-1) ярусі, що не зустрічалися у шляху від S до j_{r-1}.

Побудова дерева триває або до отримання шляхів максимального можливого рангу, або доти, поки для вершин jr-1 не виявиться, що всі вершини вже увійшли в шлях μsjr-1.

У загальному випадку кількість гілок у такому дереві (N-1)!, тобто для цього утворення потрібно (N-1)! регістрів пам'яті. Щоб уникнути таких труднощів, перейдемо від дерева D всіх шляхів до стягнутого дерева всіх шляхів (рис. 3.10, в). Воно може бути отримане стягуванням однойменних вершин дерева D на кожному ярусі. При цьому вершини з однаковими номерами на кожному ярусі упорядковуються в горизонтальні лінійки. Як видно з рис. 3.10, в, множина шляхів у стягнутому дереві шляхів така ж, як і в дереві D (рис. 3.10, б), але для читання шляхів на кожній горизонтальній лінійці дозволяється перебувати лише один раз.

Архітектура обчислювальної системи, що реалізує побудову дерева всіх шляхів довільного графа, наведена в роботі [11]. Розглянемо стратегію розподілу процесорів за алгоритмами A_i в припущенні, що в нас є N процесорів і кількість вершин у графі алгоритму так само дорівнює N. Стратегія полягає в тому, щоб задіяти процесори відповідно до стягнутого дерева шляхів графа алгоритму А. Інакше кажучи, на першому кроці працює процесор $П_1$, який реалізує алгоритм A_1, розташований на першому ярусі (r = 1). Далі за матрицею суміжності B встановлюється зв'язок між процесорами $П_1$, $П_2$, $П_3$ і $П_4$. Після закінчення роботи всіх процесорів $П_2$, $П_3$, $П_4$ згідно з матрицею суміжності B встановлюється зв'язок процесорів $П_2$, $П_3$, $П_4$ з процесорами $П_5$ і $П_6$. Результати роботи процесорів $П_2$, $П_3$, $П_4$ передаються на процесори $П_5$ і $П_6$. Далі процесори $П_5$ і $П_6$ реалізують алгоритми A_5 і A_6. Після закінчення їх виконання за матрицею суміжності встановлюється зв'язок між процесорами $П_5$ та $П_6$ і процесором $П_7$, і йому передається вся необхідна інформація для роботи. Таким чином, розв’язання задачі здійснюється в процесі побудови стягнутого дерева шляхів. На кожному етапі обчислень взаємодіють процесори, що відповідають вершинам у стягнутому дереві шляхів, розташованим на n-му та r+1-му ярусі.

Ми розглянули приклад (рис. 3.10), коли структурна схема алгоритму складена так, що алгоритми A_j на кожному ярусі одержують необхідні вхідні дані для їх реалізації. У загальному випадку при довільному заданні структурної схеми алгоритму ця

умова може не дотримуватися, наприклад, якщо структурна схема алгоритму A задана графом G (V, E), наведеним на рис. 3.11, а. На рис. 3.11, б наведено матрицю суміжності графа G і значень ступеня d^- – виходу кожної вершини і d^+ – ступеня виходу. Зрозуміло, що ступінь вершини графа G дорівнює $d = d^- + d^+$. У цьому випадку побудова стягнутого дерева шляхів (рис. 3.11, б) відрізняється лише вершинами і зв'язок з вершинами наступного ярусу встановлюється лише тоді, коли ступінь виходу вершин досяг величини, що визначається матрицею суміжності B.

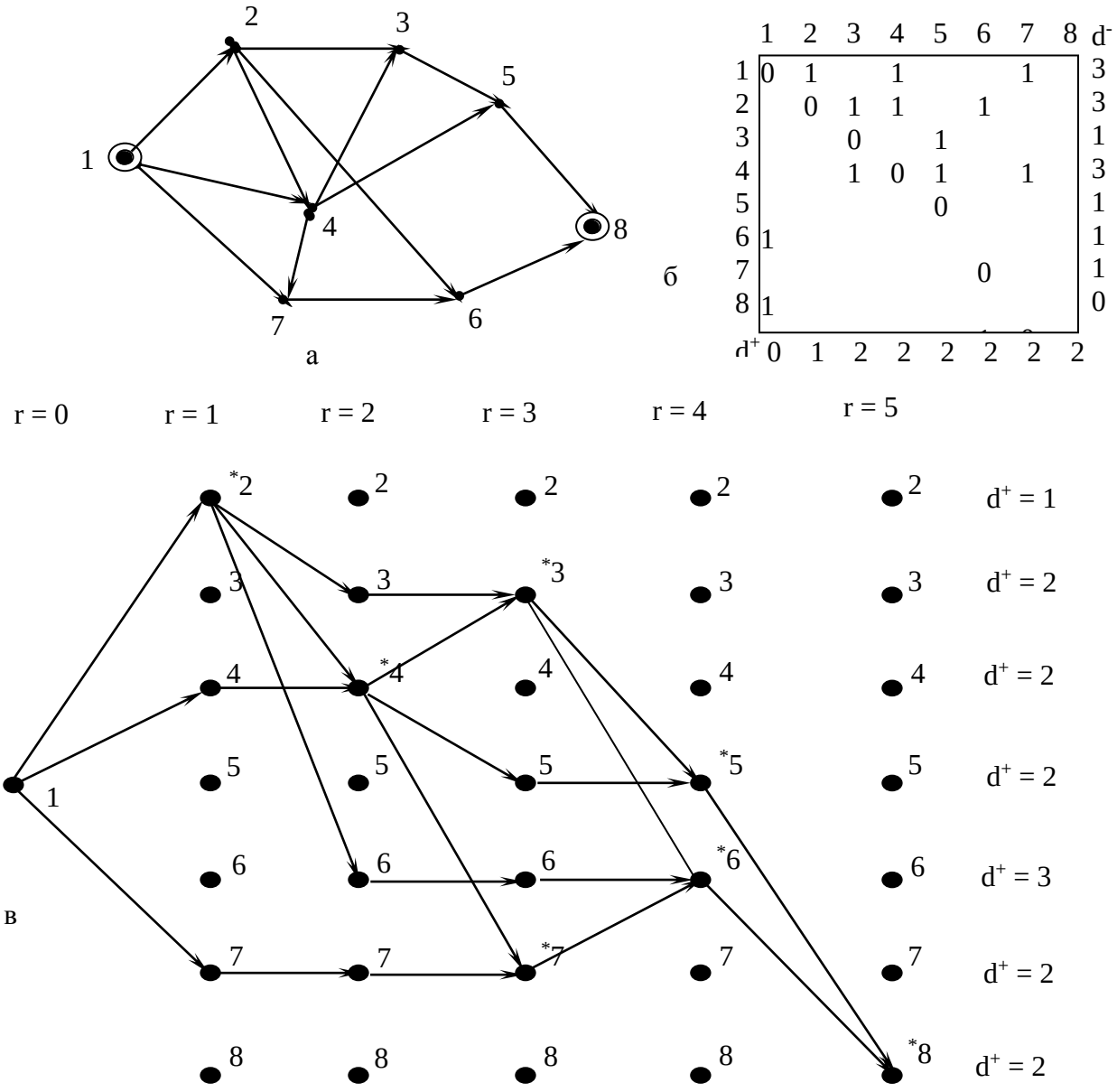

Рис. 3.11. Схема роботи алгоритму A

Фактично це означає, що процесор з номером j займається з першої подачі на нього інформації, необхідної для реалізації алгоритму А, і він залишається зайнятим доти, доки кількість звернень до нього не досягне ступеня виходу d_j^+. Це означає, що вся вхідна інформація для виконання алгоритму A_j отримана і процесор j здатний його реалізувати. Згідно зі стягнутим деревом шляхів (рис. 3.11, в) виконання алгоритму А, заданого графом G (рис. 3.11, а), здійснюється таким чином. Процесор $П_1$ виконує алгоритм A_1, далі відповідно до В встановлюються зв'язки з процесорами $П_2$, $П_4$ та $П_7$, і на них пересилаються результати виконання A_1. При цьому процесор $П_2$ одержує всю необхідну інформацію і відпрацьовує алгоритм A_2. Процесори $П_4$ і $П_7$ перебувають у режимі очікування. Далі по В встановлюються зв'язки з процесорами $П_3$, $П_4$ та $П_6$ і їм передається інформація, необхідна для їх роботи. Ступінь виходу вершини 4 дорівнюватиме $d^+ = 2$, що визначається матрицею В, і, отже, процесор $П_4$ може виконати алгоритм A_4. На рис. 3.11, в номери звільнених на кожному ярусі процесорів позначені зірочками (*). Для процесора $П_4$ визначаємо по В зв'язки з процесорами $П_3$, $П_5$ і $П_7$. При цьому процесори $П_3$ і $П_7$ виконують свої алгоритми і передають інформацію на процесори $П_5$ і $П_6$, що реалізують алгоритми A_5 і A_6, і передають вхідні дані на процесор $П_8$. Останній і завершить виконання алгоритму А.

Для здійснення такої організації обчислень пропонується використати обчислювальну структуру (рис. 3.12), яку ми і будемо називати циклічною паралельною обчислювальною системою.

Вона містить n процесорів (складається з пам'яті Мі і арифметичного логічного пристрою Аі), входи і виходи яких підключені до комутатора вимірності (n x n).

Комутатор управляється пристроєм управління введення і виведення (ПУВВ) інформації згідно з матрицею В. Фактично така структура обчислювальної системи і реалізує побудову стягнутого дерева шляхів графа, забезпечуючи зв'язок між процесорами при переході від ярусу до ярусу за допомогою ПУВВ і комутатора К, на основі матриці В. Обчислювальний процес у такій структурі можна подати у вигляді кілець, що паралельно обертаються, рух яких синхронний або асинхронний залежно від прийнятої організації синхронізації.

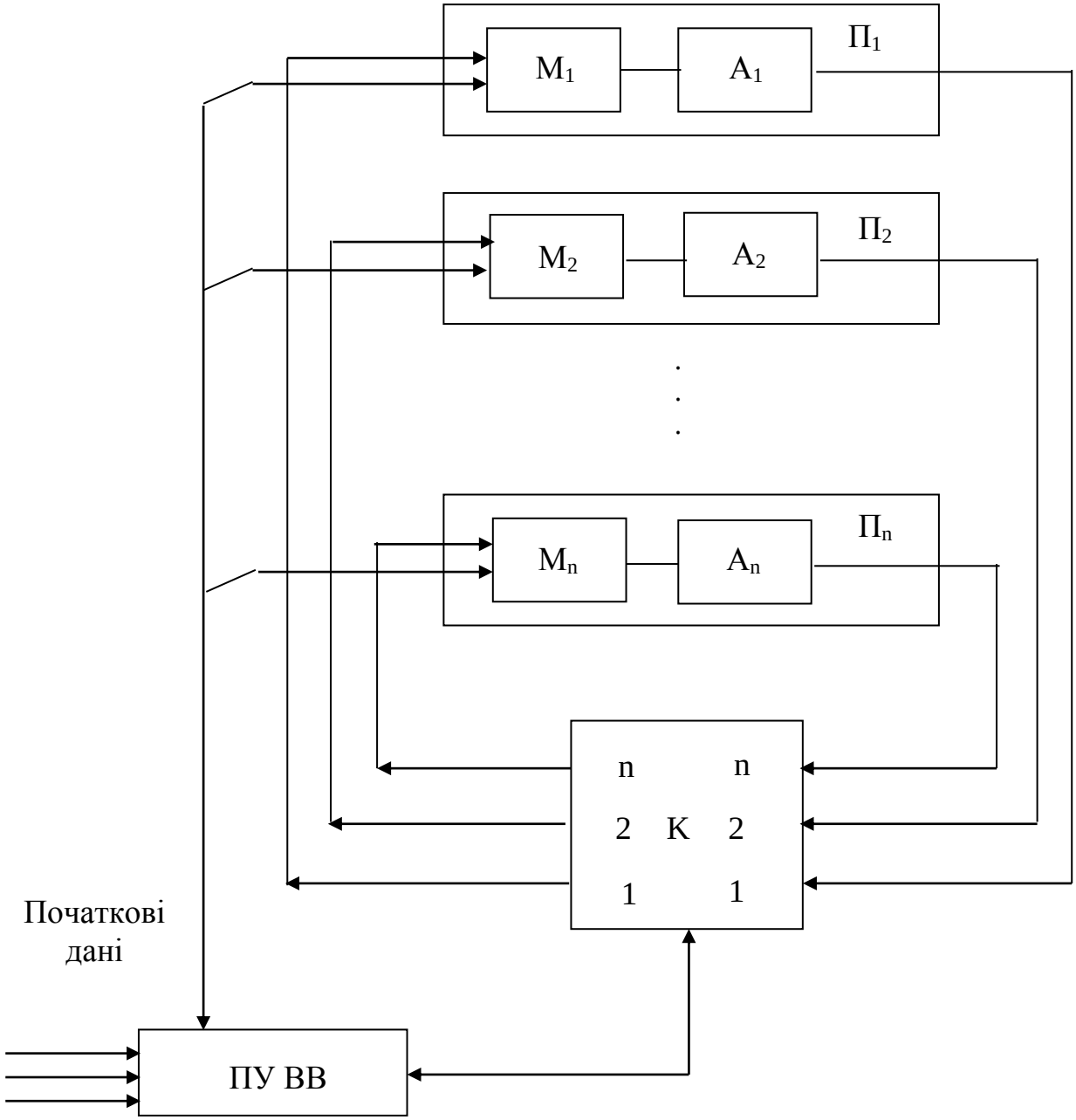

Рис. 3.12. Приклад ПОС

Слід зазначити, що якщо кількість процесорів дорівнює N (кількості вершин у структурній схемі алгоритму, заданого графом G), то на кожному кроці роботи процесорів частина з них простоює.

Якщо максимальна ширина алгоритму A дорівнює h, то водночас на довільному ярусі буде працювати не більше h процесорів. Такою кількістю процесорів можна обмежитися в кільцевій структурі, за рахунок чого зменшити кількість процесорів, що простоюють. Це можливо, якщо при переході від

ярусу до ярусу в дереві стягнутих шляхів будуть автоматично змінюватися номери комутованих процесорів згідно з матрицею суміжності B.

Архітектура кільцевої системи набуває вигляду, що показаний на рис. 3.13, де ПУП – пристрій управління перенумерацією комутатора К для застосовування процесорів, що відповідають наступному ярусу стягнутого дерева шляхів. При реалізації алгоритму А на кільцевій структурі час виконання алгоритму відрізняється від довжини критичного шляху тільки на величину At_k, де t_k – час опрацювання комутатора. При побудові комутатора на основі програмних логічних матриць t_k визначається часом перемикання логічного елемента І.

Якщо $t_k < \min \{t_j\}$, то і час виконання алгоритму А на циклічній обчислювальній системі буде точно дорівнювати довжині критичного шляху.

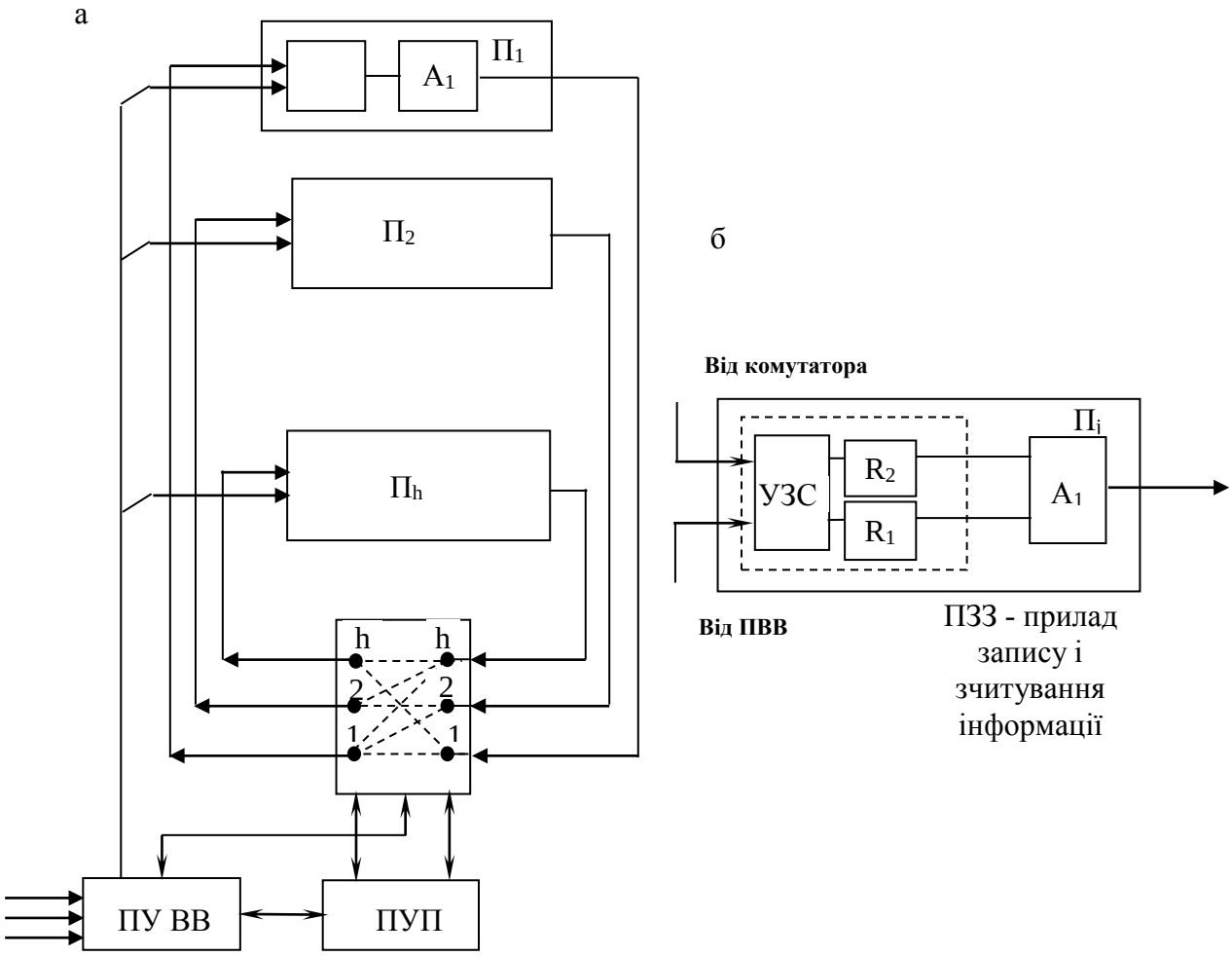

Рис. 3.13. Приклад кільцевої структури

3.2.3. Варіанти побудови циклічних паралельних обчислювальних систем

Процес реалізації алгоритму А розбивається на множину алгоритмів $\{A_j\}$, кожний з яких у свою чергу знов розбивається на підмножину алгоритмів $\{A_j\}'$ тощо, до тих пір, доки алгоритм буде являти собою просто одну з базових операцій $y_i \in Y$. Множина Y утворить функціонально повну систему. Тому процесори $П_i$ доцільно створювати з використанням трансп'ютерної технології побудови високопродуктивних мікропроцесорів, створених за принципом RISC-архітектури, що має локальну пам'ять, яка складається з двох регістрів і приладів запису і зчитування інформації. RISC-процесор має арифметичний пристрій A_i (рис. 3.13, б), що забезпечує виконання базових арифметичних операцій. Процесор у цьому випадку реалізує визначені операції над двома числами і, отже, структурна схема алгоритму А має бути зображена орієнтованим графом, у якому ступінь заходу вершини не більше двох, а ступінь виходу – довільний. У процесорах можна звільнитися від ПЗЗ (рис. 3.13, б), тобто розпаралелити процес запису вхідної інформації в регістрі пам'яті, що потребує додаткового комутатора К, який працює в паралель з основними. При цьому архітектура обчислювальної системи буде мати такий вигляд, як показано на рис. 3.14.

Для оцінки роботи алгоритмів на таких системах використаємо критерій – коефіцієнт прискорення [18]:

$$K_y = \frac{T_1}{T_n},$$

де T_1 – час реалізації алгоритму А на гіпотетичній однопроцесорній ЕОМ з такою самою швидкодією, як і $П_j$, і оперативною пам'яттю, що дорівнює сумарній пам'яті $П_j$, за наявності необхідної кількості зовнішніх приладів зі швидкостями обміну інформацією як і в циклічній обчислювальній системі;

T_n – час реалізації алгоритму А на циклічній обчислювальній системі, що містить n – процесорів.

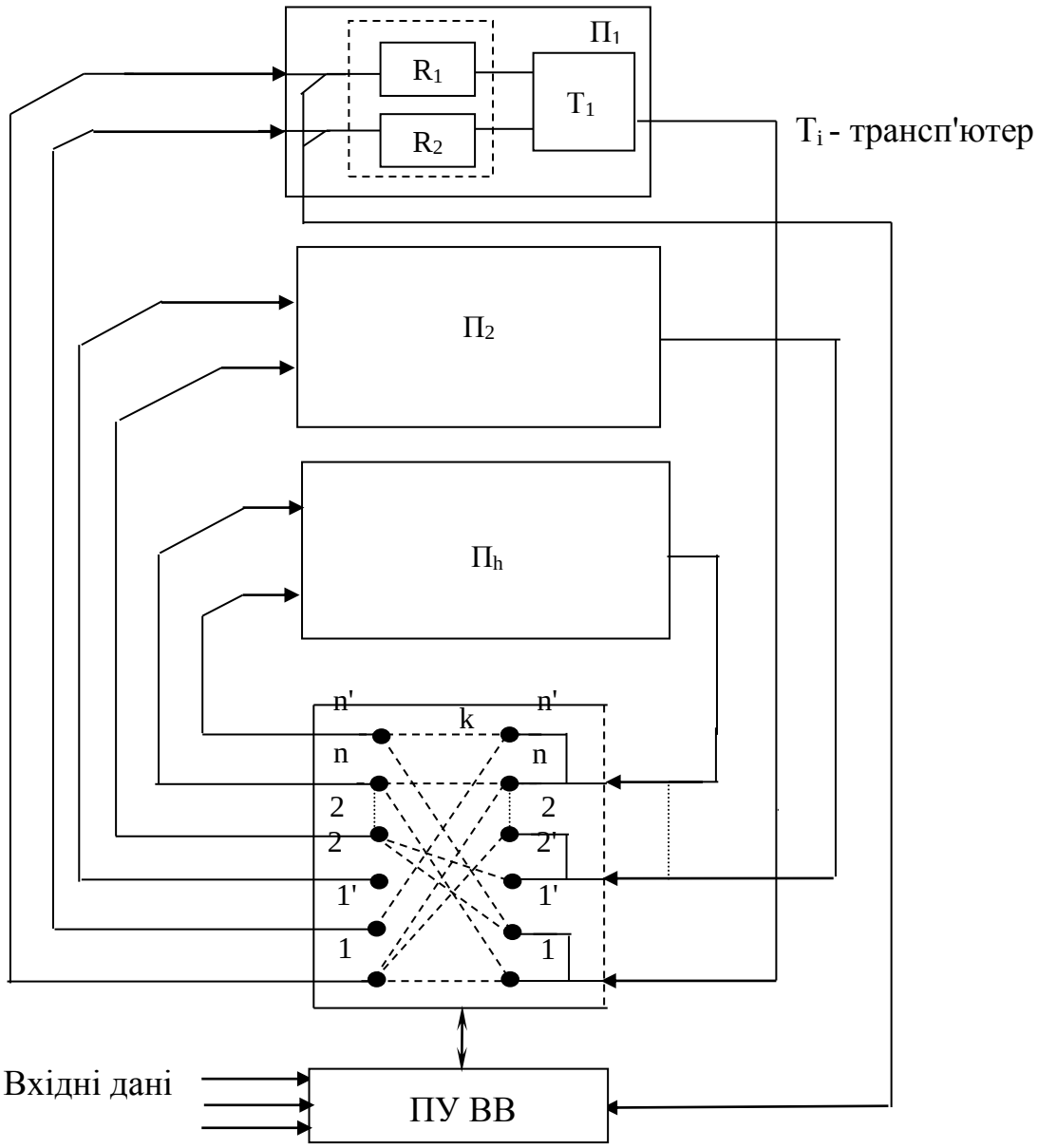

Рис. 3.14. Приклад ПОС

Якщо алгоритм А подано як структурну схему, яка задана графом G, що містить n – вершин, то

$$T_1 = \sum_{i=1}^{n} t_i + nt_0; \qquad (3.20)$$

де t_i – час виконання і-ї базової операції;

t_0 – час обміну.

Час роботи алгоритму А на структурах, наведених на рис. 3.14, можна визначити так:

$$T_n = \sum_{j=1}^{a} t_j^{max} + at_0 , \qquad (3.21)$$

де t_i^{max} – найбільший з часів базових операцій, що виконуються водночас декількома процесорами на j-му кроці;

a – глибина алгоритму A, заданого графом G.

Тоді коефіцієнт прискорення набуде вигляду

$$K_y = \frac{\sum_{i=1}^{n} t_i + nt_0}{\sum_{j}^{a} t_j^{max} + at_0} . \qquad (3.22)$$

Для спрощення аналізу припустимо, що $t_i = \overline{t_k}$ – середнє значення по i = (1, n), а $t_j^{max} = \overline{t_a}$ – середнє значення по j = (1, a). При цьому

$$K_y = \frac{n}{a}\alpha, \qquad (3.23)$$

де $\alpha = \frac{\overline{t_k} + t_0}{\overline{t_a} + t_0}$.

Як видно з формули (3.23), зі зменшенням глибини алгоритму A зростає коефіцієнт прискорення. У випадку, коли кожний процесор спеціалізується на виконанні алгоритмів деякої підмножини алгоритмів $\{A_j\}$, можлива організація обчислень циклічного типу на обчислювальній структурі, яка наведена на рис. 3.15.

У цій структурі пам'ять процесора $П_i$ містить:

- (n-1) регістр, закріплений за кожним з $П_i$ процесорів ($i = \overline{1,n}$);
- комутатор $К_i$;
- пристрій управління записом і зчитуванням;
- арифметико-логічні блоки (за кількістю регістрів);
- пристрій аналізу $У_{Ai}$, що за певним критерієм вибирає з результатів роботи A_j найкращий.

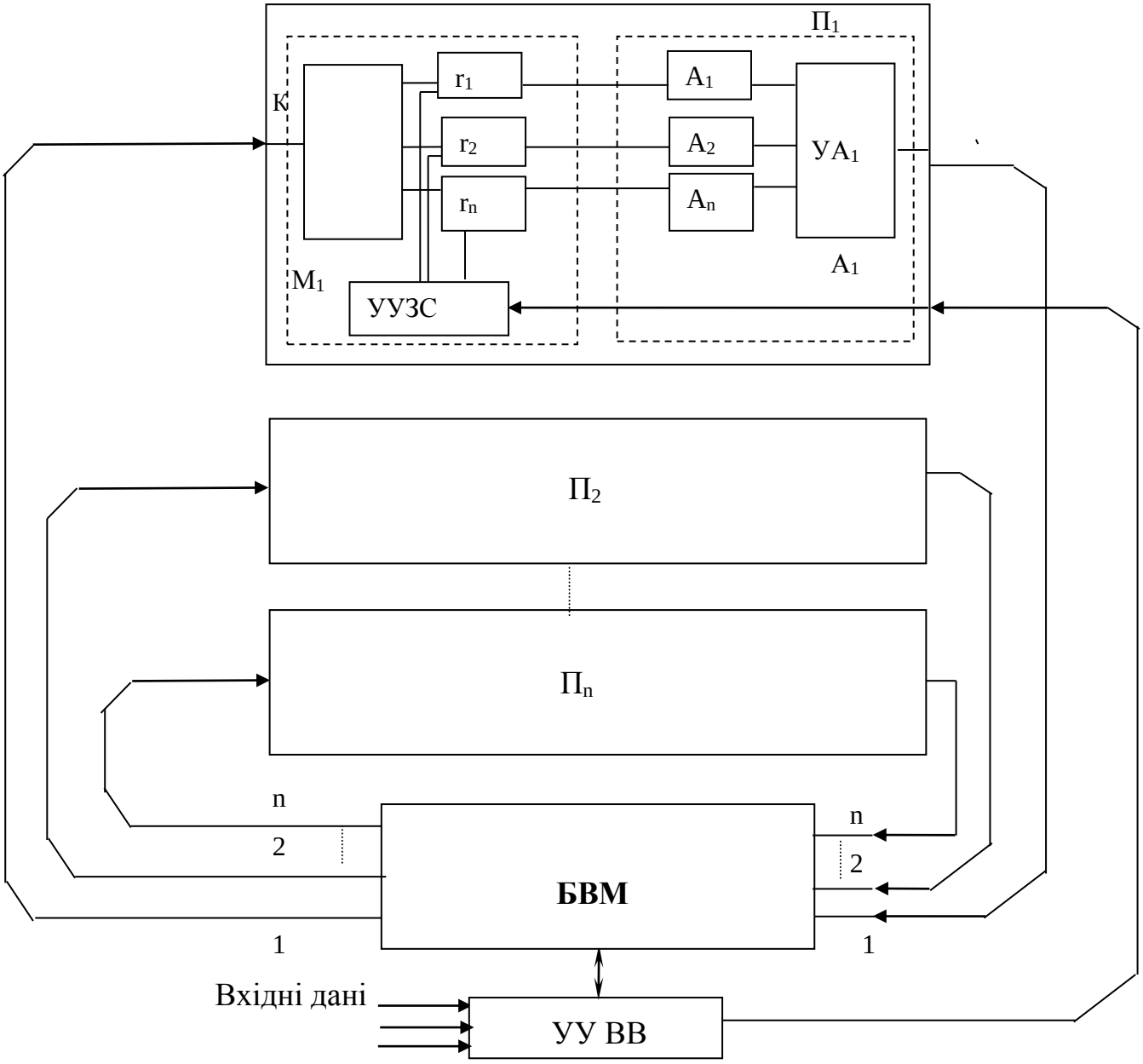

Рис. 3.15. Приклад паралельних обчислювальних систем

Кільця циркуляції інформації замикаються через блок вибору напрямку передачі інформації (БВН). Він послідовно здійснює опитування процесорів $П_i$. Інформація процесора I = 1 за допомогою комутатора $К_i$ записується в регістри пам'яті всіх процесорів $П_i$, закріплених за процесором з номером l. Після закінчення опитування процесорів вони виконують операції за отриманими даними, а результати їх роботи знов за допомогою БВН і комутаторів К заносяться в регістри пам'яті, яка закріплена за процесорами. Описані операції виконуються до повної

реалізації алгоритму А. Для цього варіанта коефіцієнт прискорення визначається як

$$K_y = \frac{\sum_{i=1}^{n} t_i + nt_0}{\sum_{j}^{a} t_j^{max} + at_0}. \qquad (3.24)$$

Щоб забезпечити прискорення, у цьому випадку повинна виконуватися нерівність

$$\sum_{i=1}^{n} t_i - \sum_{j=1}^{a} t_j^{max} > nt_0(a-1). \qquad (3.25)$$

При $t_i = t_k$ і $t_j^{max} = t_0$

$$\frac{n}{a} > \frac{t_k + nt_0}{t_a + t_0}. \qquad (3.26)$$

У реальних задачах час виконання алгоритму $A_j \in A$ різний. Тому, якщо $\Delta t = \left(t_1^{max} - t_2^{min}\right)$ – час між закінченням виконання найдовшого алгоритму $A_1 \in \{A_i\}$ і найкоротшого $A_2 \in \{A_j\}$, використати для опитування процесорів $П_i$, то прискорення, що випливає з формули (3.24), може зрости до величини, що визначається співвідношенням (3.23). Останнє еквівалентне пропозиції $n = 1$ у знаменникові дробу, що визначається співвідношенням (3.24). Таку структуру кільцевої мережі, як показано в роботі [11], ефективно прийняти для визначення найкоротших шляхів розв’язання задач динамічного програмування і варіаційного обчислювання. Задачі варіаційного обчислювання ефективно модулюються на основі підходів, викладених у роботі [24].

Реалізація кільцевих структур (рис. 3.13, 3.15) можлива і на базі багатоканальних систем зв'язку, що використовують частотне або часове ущільнення. Кількість каналів зв'язку буде визначатися кількістю регістрів пам'яті, наявних у процесорах $П_i$, тобто кожний процесор оснащений багатоканальним приймачем і передавачем (див. рис. 3.16, 3.17).

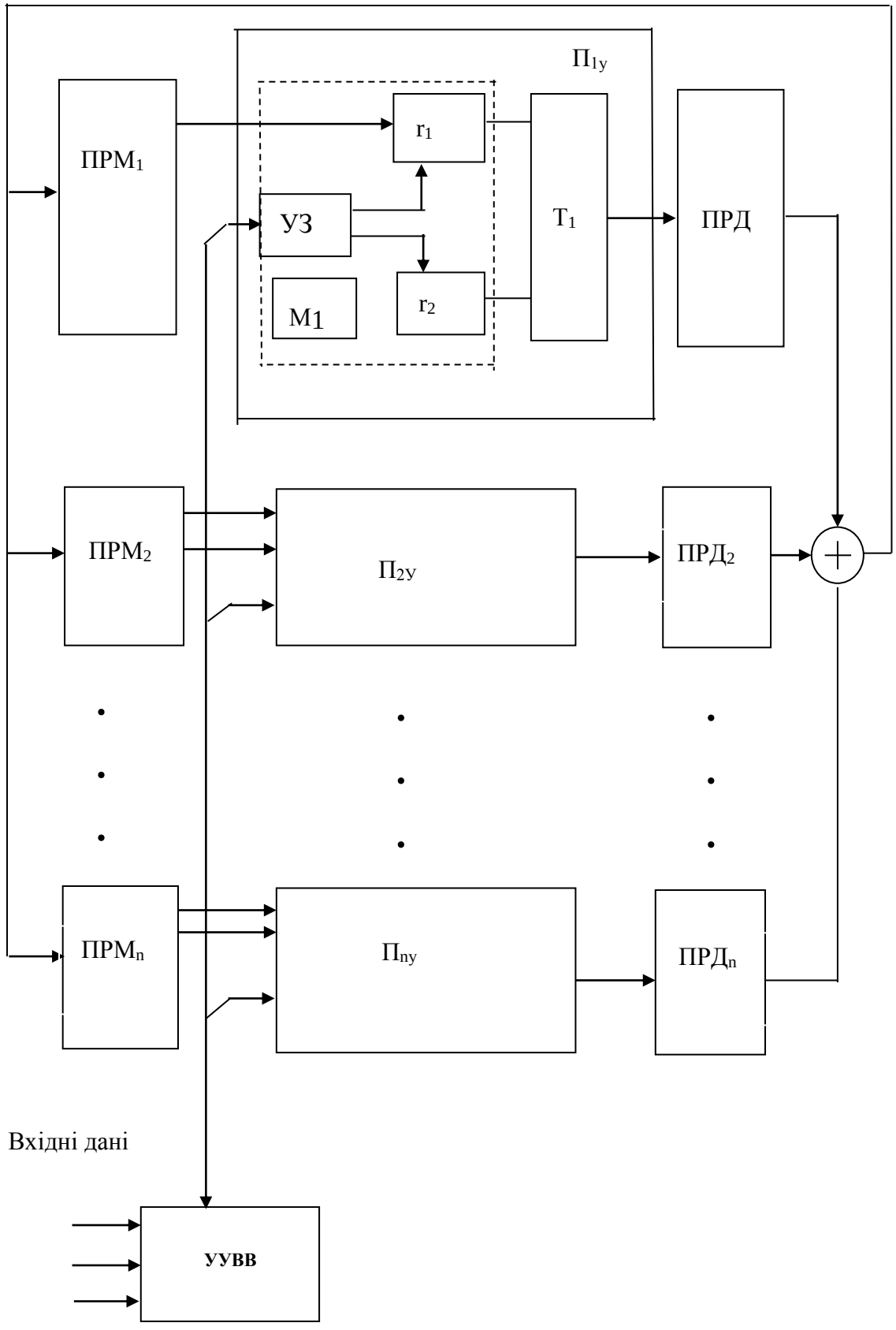

Рис. 3.16. Приклад багатоканального пристрою

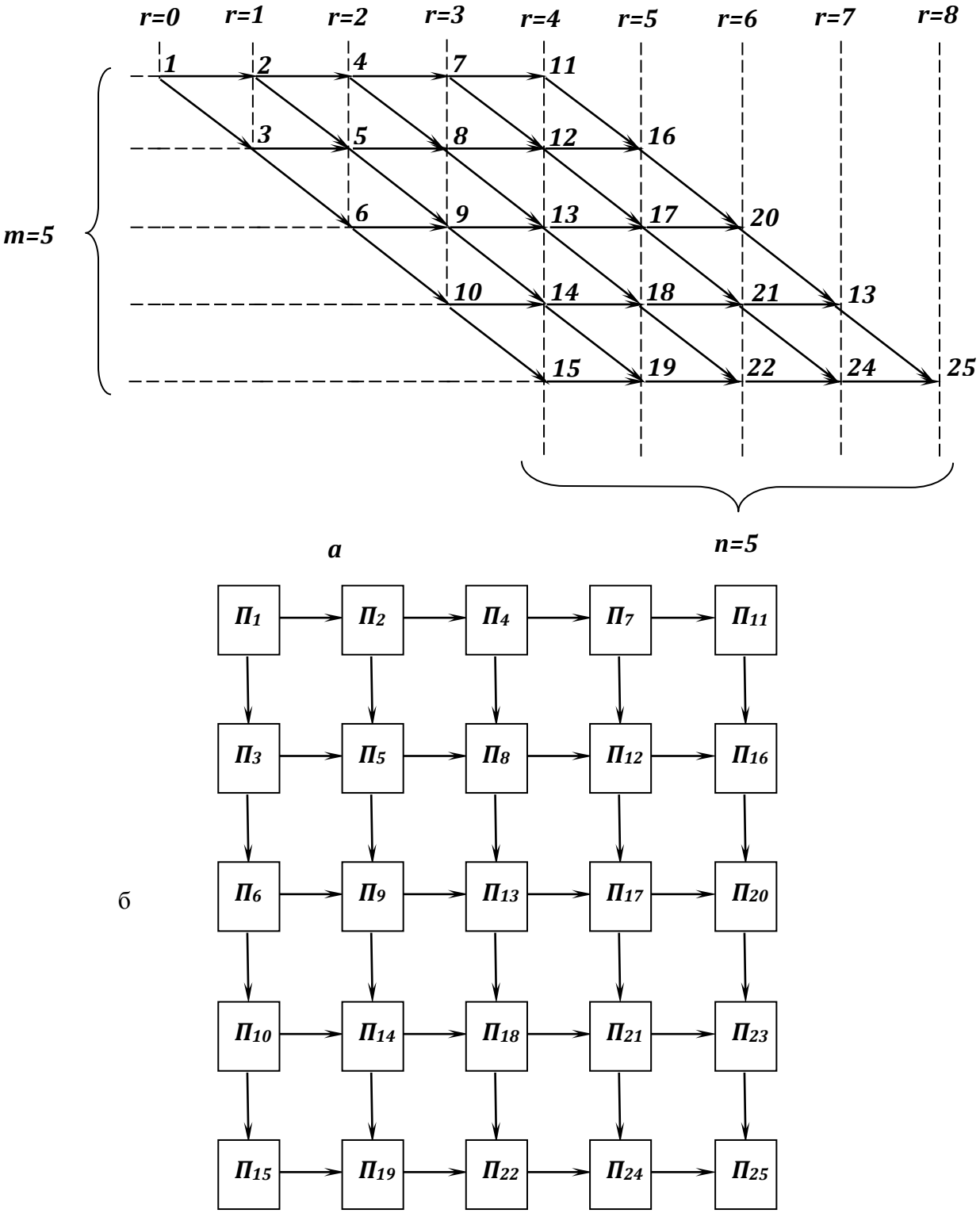

Рис. 3.17. Однорідне обчислювальне середовище

Розглянуті структури циклічного типу зможуть успішно конкурувати з конвеєрними обчислювачами, побудованими на базі однорідних обчислювальних систем. Продемонструємо цю властивість на прикладі.

Припустимо, що граф алгоритму має вигляд, показаний на рис. 3.17, а.

Такі графи властиві багатьом обчислювальним методам лінійної алгебри і математичної фізики [21]. У загальному випадку ґратка має вимірність m×n. Однорідне обчислювальне середовище, що реалізує цей алгоритм, наведено на рис. 3.17, б.

Його можна замінити на кільцеву обчислювальну структуру, що містить h – процесорів, наведену на рис. 3.13, а, де h – ширина алгоритму, що аналізується. Швидкодія залишається практично такою ж, а виграш у кількості процесорів буде в (m·n)/h разів.

У нашому конкретному випадку m = n = h = 5 і, отже, виграш в апаратурних витратах буде теж дорівнювати 5. Підвищення продуктивності циклічних паралельних обчислювальних систем залежить від їх вкладеності.

3.2.4. Вкладеність циклічних паралельних обчислювальних систем

Припустимо, що структурна схема алгоритму A задана у вигляді графа G(V,E). Кожний алгоритм A_j, який відповідає і-й вершині графа G, зобразимо як граф G`(V`, E`). Кожний алгоритм $A_i$ (граф G`) у свою чергу можна подати у вигляді графа G``(V``, E``) тощо, доти, доки вершині не буде відповідати одна з базових елементарних операцій. У цьому випадку будь-який процесор $П_i$ для виконання алгоритму A_j буде обчислювальною структурою кільцевого типу (рис. 3.12), де кожний процесор теж кільцева структура.

Ступінь вкладеності таких систем довільний. На мікрорівні він обмежується кількістю функцій процесора, що не може бути меншою від якоїсь кількості базових операцій, необхідних для розв’язання поставлених задач, а також зумовлених технологічними можливостями виготовлення процесорів.

У вкладеній системі процесор $П_i$ можна розглядати як «матрьошку», у якій містяться «процесори-матрьошки» меншої величини. У кожному процесорі $П_i$ пронумеруємо «процесори-матрьошки» від 1 до γ, починаючи з найменших.

Архітектура циклічної системи показана на рис. 3.18. Процес обчислювань у такій системі відбувається так. Водночас вмикаються процесори $П_1$ (1), $П_2$ (1) ,…, $П_n$ (1), результати своєї

роботи вони передають процесорам $П_1$ (2), $П_2$ (2) ,…, $П_n$ (2) тощо до тих пір, доки водночас не запрацюють процесори $П_1$, $П_2$,…, $П_n$. По закінченні їх роботи задача буде розв’язана.

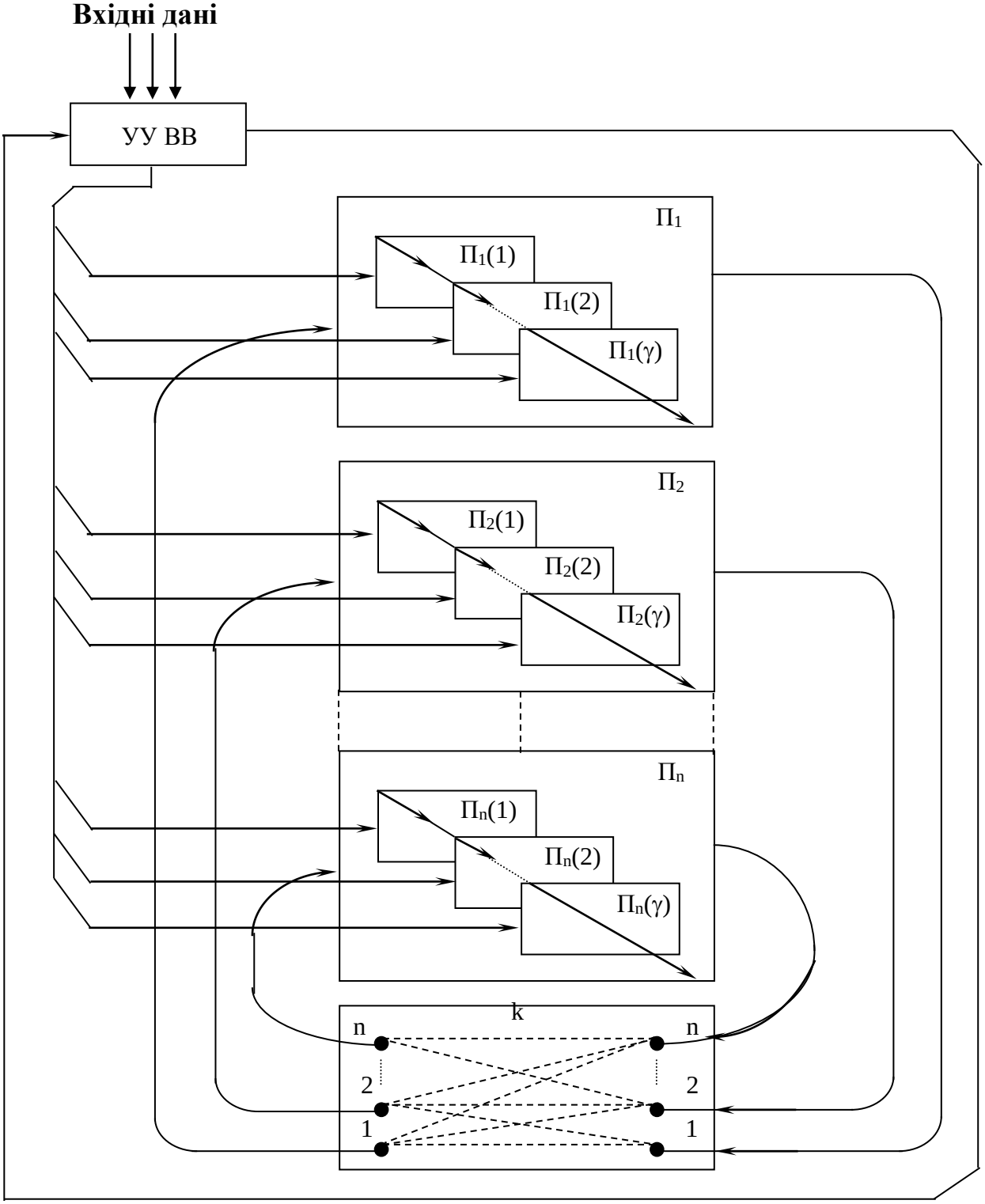

Рис. 3.18. Архітектура передбачувальної циклічної системи

На такій системі дуже ефективно розв'язуються однотипні задачі у випадку, коли їх кількість достатньо велика. Після того, як процесори $П_1$ (1), $П_2$ (1) ,…, $П_n$ (1) відпрацювали, вони звільняються, і на них можна буде розв'зувати наступну задачу.

Якщо почергово подавати задачі на розв'язання, то на процесорах $П_1$, $П_2$,…, $П_n$ буде закінчуватися розв'язуватися перша задача, на процесорах $П_1$ (1), $П_2$ (1) ,…, $П_n$ (1) – вже почнеться розв'язання задачі γ. Отже, якщо ступінь вкладеності кожного процесора $П_i$ дорівнює γ, то на ПОС можна водночас розв'язувати γ задач, причому всі «процесори-матрьошки» будуть весь час завантажені.

Для розв'язання різнотипних довільних задач доцільно використовувати структуру циклічного типу, наведену на рис. 3.13, де процесори реалізовані за трансп'ютерною технологією і кількість приладів комутації і перенумерації номерів комутатора (УКП) повинна бути такою, як і кількість типів задач, що розв'язуються.

Процесори підключаються до УКП шляхом зібрання «І», на які сигнали управління подаються з приладу розподілу вільних на цей момент часу процесорів на УКП (рис. 3.19). При такій організації обчислень водночас розв'язується q-різнотипних задач, що залежно від ширини алгоритмів їх розв'язання потребують на кожному кроці підключення до УКП певної кількості процесорів.

Тут можлива ситуація, коли на якомусь кроці деякі задачі потребують усіх процесорів, що є в наявності, і тоді розв'язання задач, що залишилися, буде затримуватися.

У більшості випадків комплекс задач, що розв'язуються, має деяку кінцеву мету і очевидно, що вона буде швидше реалізована, якщо сумарний час розв'язання цього комплексу задач буде мінімальним.

Розглянемо задачу визначення оптимальної стратегії розподілу процесорів $П_i$ по УКП ($i = 1 \div n$), при якій сумарний час T розв'язання всього комплексу задач мінімальний.

Позначимо через A_{jr} комплекс алгоритмів, які необхідно виконати на r-му кроці роботи для розв'язання j-ї задачі ($i = 1 \div n$). У кожному комплексі алгоритмів є алгоритм з найбільшим часом реалізацій. Такі алгоритми будемо називати критичними і позначати A_{jr} (кр.).

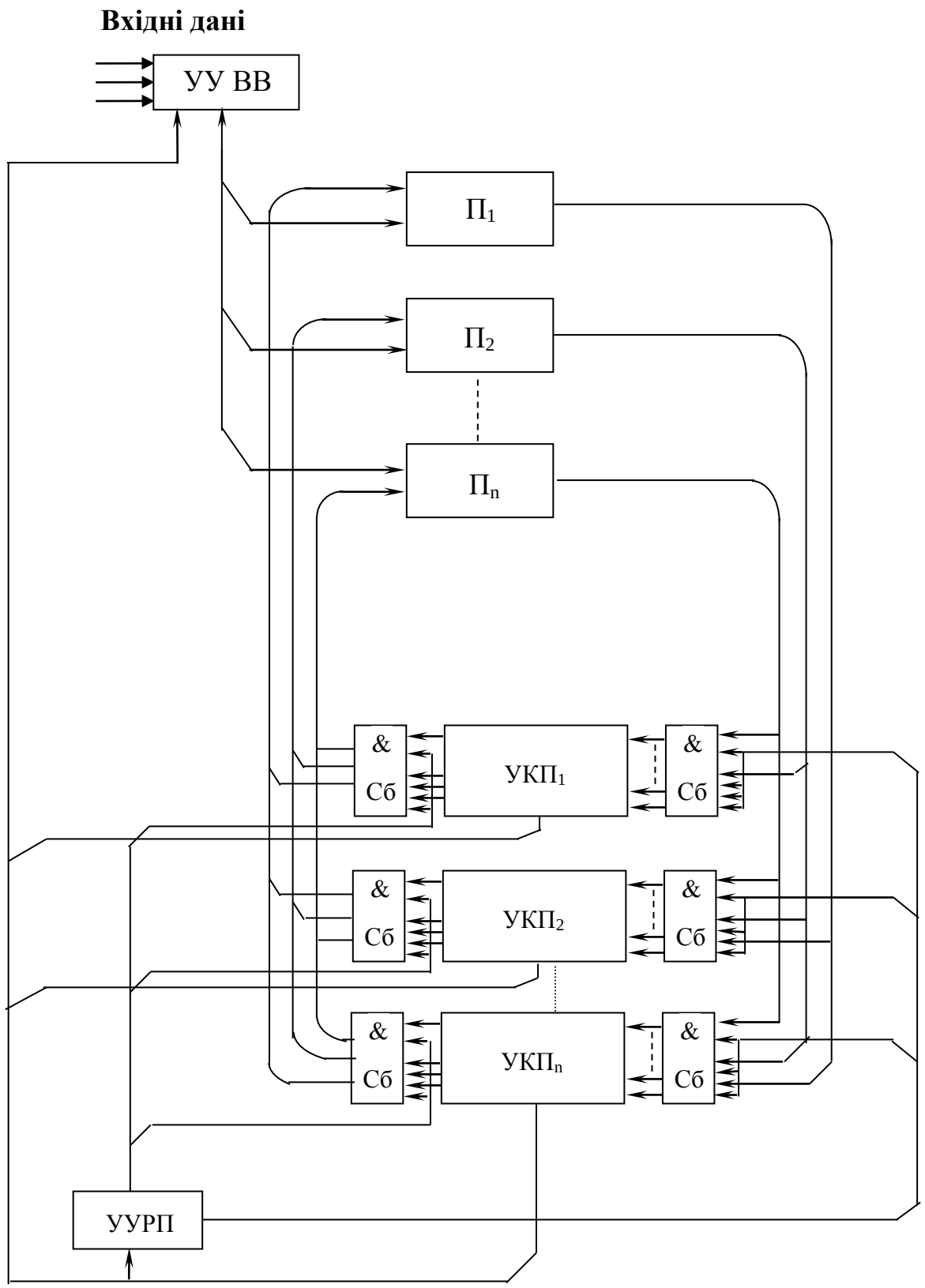

Рис. 3.19. ПОС із кількістю УКП залежно від типів задач

Тоді час виконання j-ї задачі дорівнює

$$t_{вj} = \sum_{r=1}^{a_j} t_r^j + \sum_{r=1}^{a_j} t_{ожг}^j (\Pi_r(t_i)), \qquad (3.27)$$

де

$$t_{ожг}^j(\Pi_r(t) = \begin{cases} 0 - \text{якщо передбачається виділення } x_r^j = x_{нг}^j ; \\ \Delta t_r^j, \text{ в іншому випадку;} \end{cases}$$

$\prod_r(t_i)$ – план розподілу процесорів Π_i за задачами в момент часу t_i;

Δt_r^j – час простою j-ї задачі на r-му кроці виконання при нестачі кількості процесорів X_i^rн;

t_r^j – час виконання A_{jr}.

Мінімальний час розв'язання всіх задач з урахуванням формули (3.27) визначиться таким чином:

$$\mathbf{T = \min\left\{ \max\left[\sum_{r=1}^{a_j} t_r^j + \sum_{r=1}^{a_j} t_{ожг}^j(\Pi_r(t_i)) \right] \right\};}$$

$$\sum_{j=1}^{q} b_j x_j \le b_c(t_i);$$

$$\sum_j b_j \le n;$$

$$x_j = \begin{cases} 1 - \text{якщо реалізується алгоритм } A_r^j; \\ 0 - \text{в іншому випадку,} \end{cases}$$

де b_j – кількість процесорів, необхідних у момент часу t_j для виконання A_r^j;

bc (t_i) – кількість вільних процесорів у момент часу t_i;

n – загальна кількість процесорів у системі (рис. 3.18).

Таким чином, оптимальний розподіл процесорів за задачами з метою мінімізації їх сумарного часу виконання T являє собою мінімаксну задачу, у якій необхідно знайти таку послідовність планів Π_2 (t_1), Π_r (t_2) ,…, Π_r (t_j), коли буде досягатися мінімум функціонала (3.27). Задачу знаходження оптимального розподілу

процесів за задачами в момент t_i можна сформулювати таким чином:

$$\max \sum_{j=1}^{q} x_j$$

при обмеженнях

$$\sum_{j=1}^{q} b_j x_j \le b_c(t_i);\ 0 \le b_j \le n;\ xj\{0\in,\ 1\}.$$

Вона являє собою задачу лінійного програмування з булевими змінними, відому як задачу про рюкзак [4]. У ній шукається максимальне число Arj, яке можна виконати в момент часу ti вільними процесорами. Цей процес потрібно повторювати багаторазово при переходах від одного циклу обчислень до іншого.

У роботі [1] показано, що ця задача зводиться до задачі визначення екстремальних шляхів з обмеженням за вагою і може бути ефективно розв'язана в масштабі реального часу в мікросекундному діапазоні на обчислювальних структурах циклічного типу. Для вдосконалення структури (рис. 3.18) необхідна спеціалізація процесорів на розв'язання різноманітних класів задач. Процесори можуть мати більш глибоку вкладеність, тобто кожний процесор системи (рис. 3.19) у свою чергу являє собою обчислювальну систему (рис. 3.18).

Контрольні питання

1. Як пов'язана структура алгоритму з архітектурою ПОС?
2. Які основні завдання вирішуються при створенні ПОС?
3. Що таке асимптотика і як її використовують для визначення складності алгоритмів?
4. Як класифікуються ПОС?
5. Які типи абстрактних машин виділяються у ПОС?
6. Які основні показники ефективності паралельних алгоритмів та ПОС?
7. Пояснити принцип циклічної обробки інформації у ПОС.
8. Які архітектури циклічних ПОС існують?
9. Що таке коефіцієнт прискорення?
10. Пояснити поняття вкладеності циклічних ПОС.

Розділ 4. Систолічні матричні процесори для систем прийняття рішень та штучного інтелекту

4.1. Паралельні обчислювальні системи на базі трансп'ютерних технологій у системах прийняття рішень та штучного інтелекту

Систолічні процесори [18, 21] – це новий клас конвеєрних матричних структур. Систолічна система являє собою мережу процесорів, які виконують ритмічні обчислення та передачу даних по системі. Наприклад, можна показати, що об'єднання процесорних елементів (ПЕ), виконуючи елементарну операцію «скалярне множення» локальними зв'язками, дасть змогу реалізувати цифрове фільтрування, множення матриць та інші операції. Систолічний масив має важливі властивості модульності, регулярності, локальної взаємодії, високий рівень конвеєрності і синхронізованої мультиобробки. Просування даних у ньому завжди описується за допомогою кадрів (миттєвих уявлень функціонування комірок масиву).

Систолічний процесор відрізняється від звичайної Неймановської машини високим рівнем конвеєрних обчислень, тобто як тільки елемент даних виходить з пам'яті, він може бути ефективно використаний у кожній комірці, у яку він влучає при «прокачуванні» на всьому масиві. Це викликає інтерес до широкого класу обчислювальних задач, коли множина операцій повторно виконується над кожним елементом даних. У цьому випадку усуваються проблеми обміну з пам'яттю, притаманні Неймановським машинам.

Концептуально обчислювальні задачі можна поділити на два сімейства – задачі, пов'язані з обчислюваннями, та задачі, пов'язані з введенням-виведенням. Якщо в обчислювальній задачі загальна кількість операцій більша від загальної кількості операцій введення-виведення, говорять, що ця задача пов'язана з обчисленням, інакше – з введенням-виведенням. Наприклад, звичайне множення матриць є задачею, пов'язаною з обчисленнями та введенням двох матриць – задачею, пов'язаною з введенням-виведенням. Для прискорення розв'язання задач,

пов’язаних з введенням-виведенням, необхідно збільшити доступ до пам'яті, що при існуючій технології вкрай важко. У той же час прискорити розв’язання задач, пов’язаних з обчисленнями, часто можна за рахунок використання систолічних масивів. Основна конфігурація систолічного масиву подана на рис. 4.1.

Замінюючи окремий процесор одно- або двовимірним масивом процесорів, можна досягнути збільшення продуктивності обчислення без збільшення пропускної спроможності пам'яті.

Рис. 4.1. Базова конфігурація систолічного масиву

4.1.1. Визначення систолічних масивів

У роботах [18, 20] дано декілька визначень систолічних масивів.

Систолічний масив – це обчислювальна мережа, що має такі властивості:

1) *синхронність.* Дані обчислюються ритмічно (при глобальному тактируванні) і пропускаються по мережі;

2) *модульність і регулярність.* Масив містить модульні процесорні елементи з однорідними зв'язками. Більш того, обчислювальна мережа може необмежено поширюватися;

3) *просторова локальність і часова локальність.* Масив характеризується локально зв'язаною структурою міжз’єднання, тобто просторовою локальністю. У структурі існує розподілена затримка (щонайменше одиничної тривалості), протягом якої можуть бути завершені приймання, обробка і видача сигналу від одного вузла до іншого, тобто часова локальність;

4) *конвеєризованість.* Масив виявляє конвеєризованість з лінійною швидкістю, тобто досягається прискорення обчислення O(M), де M – кількість ПЕ. Тут ефективність робіт масиву

визначається коефіцієнтом прискорення T_s / T_p, де T_s – час обробки на одному процесорі і T_p – час обробки на матричному процесорі.

4.1.2. Властивості систолічної архітектури

До основних факторів, які пояснюють інтерес до систолічних масивів для спеціалізованих систем обробки, відносять простоту і регулярність схеми, паралелізм і зв'язок, збалансованість обчислювань з введенням-виведенням.

Простота і регулярність схеми. В інтегральній технології вартість компонентів суттєво падає, однак вартість схеми зростає разом з її складністю. Використання простих і регулярних схем із застосуванням ПВІС-технології дає змогу отримати відчутний виграш. Крім того, прості і регулярні схеми передбачають модульність системи, що сприяє збільшенню продуктивності.

Паралелізм і зв'язок. Важливим фактором у швидкодії обчислювальних систем є використання паралелізму. Для спеціалізованих систем паралелізм визначається тими алгоритмами, котрі реалізуються в системі. При великій кількості одночасно працюючих процесорів важливого значення набуває зв'язок. У ПВІС-технології вартість трасування домінує над вимогами потужності, часу і площі, необхідних для реалізації обчислювання, тому регулярні і локальні реалізації обчислювань є вкрай корисними.

Збалансованість обчислювань з введенням-виведенням. Систолічний масив, як правило, використовується у вигляді приєднаного матричного процесора і проводить обмін даними і результатами за допомогою основної машини. Тому втрати на операції введення-виведення повинні враховуватись при оцінці загальної продуктивності. Кінцева мета реалізації матричної процесорної системи – швидкість обчислювань, погоджена з шириною смуги введення-виведення основної машини. При відносно малій пропускній спроможності сучасних приладів введення-виведення для збільшення швидкодії необхідно виконувати великий обсяг обчислювань на одне звернення до введення-виведення. Але повторне використання даних потребує їх зберігання у системі протягом достатньо тривалого часу.

Інакше кажучи, проблема введення-виведення пов'язана не лише з пропускною спроможністю, але і з вимогою до внутрішньої пам'яті. Тому є важливим при алгоритмічній обробці матричного процесора питання, як співвіднести обчислення з відносною структурою пам'яті і з пропускною спроможністю введення-виведення з тим, щоб час обчислень був збалансованим з часом введення-виведення.

Проблема введення-виведення стає особливо актуальною, коли розв’язання задачі великого розміру виконується на невеликому матричному процесорі. Неминуче виникає задача роздроблення, тобто частини повинні бути декомпозованими. На практиці це відбувається дуже часто, і тому виникає питання, як зробити роздроблення і як організувати буферну пам'ять з метою мінімізації введення-виведення, що є критичними для практичного розроблення матричної системи обробки.

4.1.3. Схеми розподілу тактових імпульсів

Згідно з роботою [18] синхронізація дій у великих синхронних системах (таких, як систолічний масив) здійснюється широким системним синхронізуючим сигналом. Синхронізуючий імпульс виконує дві функції – він виступає як еталон послідовності та як еталон часу. Як еталон послідовності тактові імпульси служать визначенню послідовних моментів часу, у які система може змінити свій стан. Як еталон часу тактові імпульси необхідні для визначення затримок спрацювання елементів і передачі даних з виходу на вхід синхронізованих елементів. Подвійна роль синхронізуючих імпульсів, з одного боку, вносить деяке спрощення в розроблення цифрових систем, але, з другого – об’єднання функцій установлення послідовності робіт і синхронізації є джерелом багатьох труднощів при розробленні, обслуговуванні, модифікації і забезпеченні надійності синхронних систем.

Оскільки синхронізуючі імпульси передбачають узгодження в усій системі, їх розподіл є критичним для систолічних масивів. Існує низка проблем, пов’язаних із синхронізацією великих масивів. Ці проблеми виникають, головним чином, у результаті розфазування синхронізуючих імпульсів, тобто ситуації, коли всі

ПЕ в масиві не можуть отримати тактові імпульси одночасно. Причини таких явищ – різні довжини сполучення генератора тактових імпульсів з кожним ПЕ та інші причини, наприклад, варіації параметрів сигналу в різних колах розповсюдження. З метою елімінації цих явищ у регулярних масивах для розповсюдження синхронізуючих імпульсів можна використовувати схеми Н-дерев, у яких кожний ПЕ розміщується на однаковому віддаленні від генератора. Розміщення Н-дерев для лінійних, квадратних та гексагональних масивів показані на рис. 4.2.

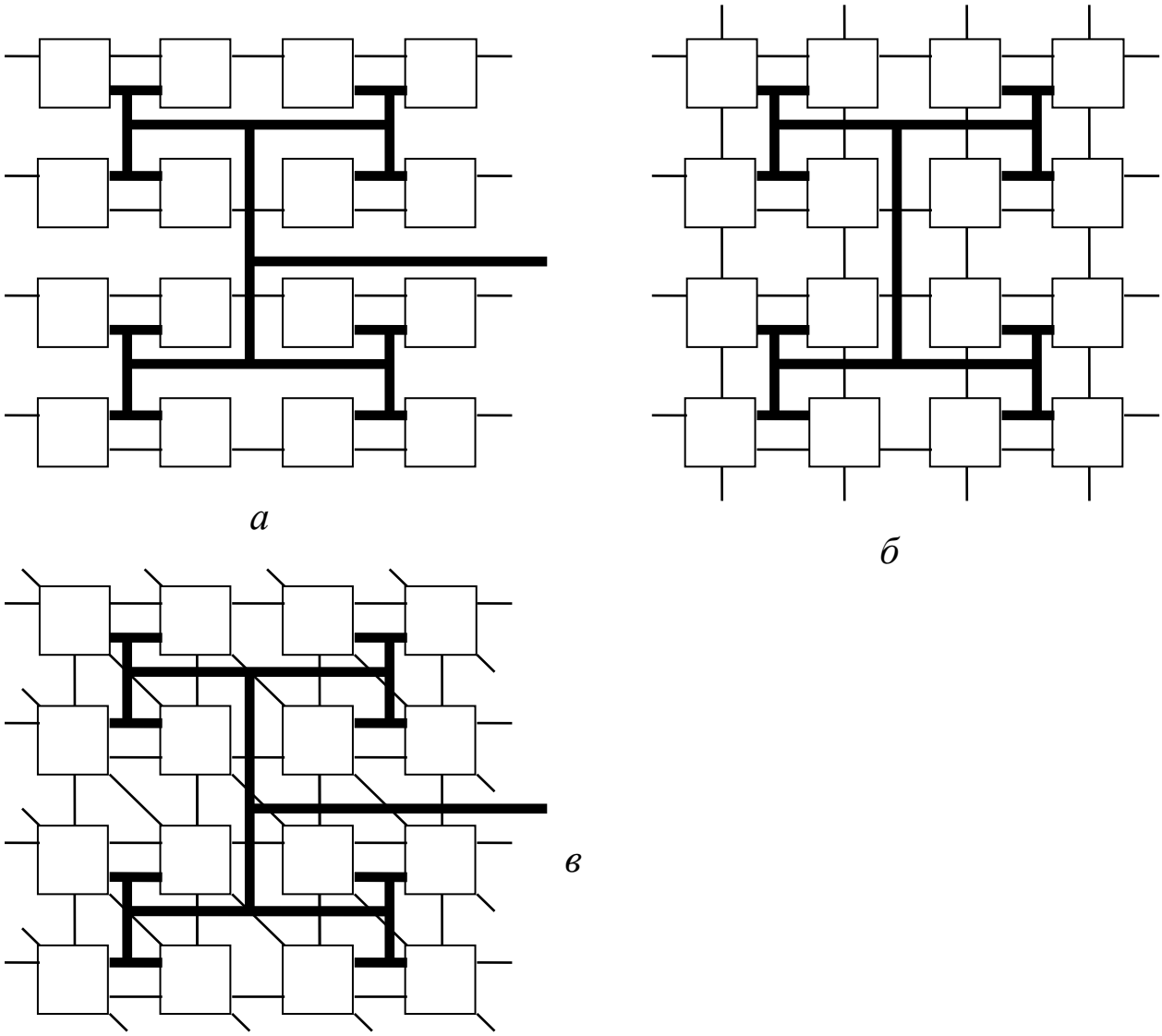

Рис. 4.2. Розміщення Н-дерев для синхронізації лінійного масиву (а), квадратного масиву (б) і гексагонального масиву (в)

Хоча Н-дерева і дають змогу вирішувати проблему рівновіддаленості для розповсюдження синхронізуючих імпульсів, завдання розфазування вирішується не повністю. Фішер показав, що досить великий лінійний систолічний масив може бути синхронізований глобальними імпульсами за допомогою їх конвеєрного розповсюдження.

Однак при синхронізуванні двовимірного масиву звично виникає розфазування, пропорційне розміру масиву. При елімінації цих труднощів були розроблені асинхронні хвильові матричні процесори, які розглядаються далі. В асинхронних системах проблема тактування може бути спрощена, оскільки необхідно підтримувати тільки коректну послідовність дій (а не синхронізацію).

4.1.4. Критерій оптимальності та основні формули

Існує безліч факторів, що визначають критерій оптимальності при розробленні систолічних масивів. Кінцевий вибір критеріїв оптимальності графа залежить від сфери застосування. До найбільш типових факторів слід віднести час обчислення, конвеєрний такт, блоковий конвеєрний такт, розмір масиву, канал введення-виведення. Уточнимо їх визначення.

Час обчислення T – інтервал часу між початком обчислення першого і закінченням останнього обчислення для однієї задачі, що розв'язується на процесорному масиві.

Конвеєрний такт α – інтервал часу між двома послідовними обчисленнями в процесорі. Інакше кажучи, процесор зайнятий один з кожних α інтервалів часу. (Зазначимо, що α є величиною, зворотною швидкості конвеєра.)

Блоковий конвеєрний такт β – інтервал часу між початком двох послідовних задач, що виконуються в процесорному масиві.

Розмір масиву – кількість процесорів у масиві. Розмір масиву, очевидно, визначає базову вартість апаратури.

Канали введення-виведення – кількість каналів введення-виведення, які зв'язують із зовнішнім середовищем (основною машиною). Кількість каналів введення-виведення безпосередньо впливає на вартість апаратури (залежить від кількості виводів

кристала ПВІС або шин введення-виведення на друкарській платі).

Інколи викликає інтерес комбінація двох або декількох з перелічених факторів. Наприклад, добуток розміру масиву на час обчислень є корисною мірою оцінки апаратури за критерієм «вартість-ефективність». Інший корисний критерій – використання процесора, що залежить від конвеєрного такту, блокового конвеєрного такту і схем введення-виведення даних. Нижче наводяться можливі варіанти, з яких видно, як у різних додатках природно виникають критерії, що визначають проектування.

1. *Розв'язання однієї задачі з кінцевим потоком даних.* У випадку, коли ГЗ закінчене, час обчислень, можливо, є більш важливим критерієм, ніж конвеєрний такт. У працях зарубіжних авторів показано, як мінімізувати час обчислень на систолічному масиві.

2. *Розв'язання однієї задачі з нескінченним потоком даних.* У багатьох застосуваннях ЦОС (таких як фільтрування) довжина вхідних даних є дуже великою або нескінченною, що призводить до ГЗ вузької форми. У цьому випадку мінімізація часу обчислень еквівалентна мінімізації конвеєрного такту. Наприклад, у випадку пакетів, якщо вхідна послідовність є нескінченною, то конвеєрний такт стає визначальним критерієм оптимізації. При малому конвеєрному такті малий і час обчислень.

3. *Розв'язання великої кількості задач.* При обробці на одному і тому самому систолічному масиві великої кількості задач варто звернутися до блокового конвеєрного такту β. Якщо повинні бути обчислені M задач, то і загальний час обробки становить $M \times \beta$. Якщо не враховувати впливу на роботу систолічного масиву схеми введення-виведення, то блоковий конвеєрний такт обчислюється досить просто. Наприклад, таблиця резервування для всіх ПЕ визначає інтервал часу їх зайнятості при розв'язанні однієї задачі. На підставі таблиці можна обчислити для кожного ПЕ проміжок часу між останнім і першим інтервалом зайнятості. Тепер легко обчислити блоковий конвеєрний такт – це просто найбільший проміжок часу за всіма ПЕ масивами.

Основний принцип систолічного розроблення полягає в досягненні масового паралелізму при мінімальних витратах на зв'язок. Властивості модульності, регулярності і локальності зв'язків роблять систолічні структури досяжними при реалізації на ПВІС. Однак їх застосування обмежене спеціальним класом алгоритмів, що використовують регулярні і локальні структури даних, відповідні систолічним структурам. Зараз велику увагу дослідників привертають прямі засоби відображення алгоритмів у систолічних масивах.

4.2. Хвильові матричні процесори в системах прийняття рішень та штучного інтелекту

Систолічні масиви добре пристосовані для ПВІС – реалізації обчислювальних алгоритмів обробки сигналів і зображень. Вони мають такі переваги, як модульність, регулярність, локальні міжз’єднання, високий ступінь конвеєрної мультиобробки та безперервний потік даних між ПЕ. Недолік систолічних масивів – глобальне управлінні їх функціонуванням за допомогою потактової синхронізації. В апаратній реалізації глобальна синхронізація призводить до проблем розфазування синхронізуючих імпульсів, відмовостійкості та пікової потужності. Просте вирішення цих проблем у матричних процесорах полягає в застосуванні принципу обчислення з управлінням потоку даних, що і є основою.

На відміну від Неймановської машини, традиційної ЕОМ зі збереженою програмою, що основана на потоці управління (тобто команди виконуються відповідно до потоку управління, яке задається програмою), ЕОМ з управлінням потоком даних для запуску команди на виконання використовують потік даних. Інакше кажучи, у машинах з управлінням потоком даних команда ставиться в чергу на виконання лише після того, як одержані всі операнди, які її активізують. Результат виконання команди у свою чергу просувається далі і використовується іншими командами. Такий підхід виключає необхідність у глобальному управлінні та глобальній синхронізації. Завдяки цьому для матричної обробки можна використовувати підхід, оснований на

управлінні даними та автосинхронізації. При цьому чітко прослідковується залежність даних (виконання команд залежить від доступності їх операндів). Крім того, цей підхід замінює вимогу коректної синхронізації вимогою коректного порядку стеження.

Робота універсальної ЕОМ, яка керується потоком даних, часто пов'язана з великим обсягом керованих даних та ресурсів, для чого потрібна потужна управляюча система. З цієї причини сучасні ЕОМ з управлінням потоком даних не досить ефективні. Великій кількості алгоритмів притаманні властивості модульності та локальності, котрі легко можуть бути використані. Відображення цих властивостей у мультипроцесорній системі з управлінням потоком даних може спростити проблеми обміну та конфліктів при звертанні до пам'яті в універсальних ЕОМ такого типу. Ця думка приводить до ідеї хвильового матричного процесора.

4.2.1. Визначення хвильових матричних процесорів

Визначення хвильового матричного процесора має вигляд обчислювальної мережі з такими властивостями:

1) автосинхронні обчислення, що керуються даними. Глобальна синхронізація не потрібна, бо обчислення – автосинхронне;

2) регулярність, модульність, локальність міжз'єднань. Матричний процесор повинен містити модульні процесорні елементи з регулярними та (просторово) локальними міжз'єднаннями. Крім того, обчислювальна мережа може бути необмежено розширеною;

3) можливість програмування хвильовою мовою чи шляхом задання графа потоку даних (ГПД). Хвильові матриці можуть бути реалізовані у вигляді спеціалізованих процесорів або тих, що програмуються. Однак останні є більш вагомими. Використання хвильових алгоритмів обчислень може спростити програмування матричних процесорів;

4) конвеєр з лінійним зростанням прискорення. Хвильовий процесор повинен забезпечувати лінійне зростання прискорення, тобто досягати прискорення O(M), де M – кількість ПЕ.

Зазначимо, що основна властивість, яка відрізняє хвильовий процесор від систолічного процесора, − властивість управління даними, тобто у хвильовому процесорі буде відсутня глобальна синхронізація. У хвильовій архітектурі передача інформації поміж ПЕ здійснюється на підставі взаємодії ПЕ з його безпосередніми сусідами. Кожного разу, коли дані, що передає ПЕ, доступні, інформується про це ПЕ, який приймає дані, що від нього вимагаються. Після цього ПЕ-приймач надсилає відправнику підтвердження про одержання даних. Ця схема може бути реалізована за допомогою простого протоколу підтвердження зв'язку, який гарантує, що обчислювальні хвильові фронти спрямовані в правильній послідовності і не призводять до руйнування інших фронтів.

Існує простий засіб порівняння хвильового процесора з його систолічним аналогом:

Хвильовий процесор = Систолічний процесор + Обчислення управління потоком даних.

Іншими словами, хвильова обробка використовує локальність потоку даних і потоку управління, притаманну багатьом алгоритмам обробки сигналів. Оскільки немає потреби синхронізації робіт усього процесора, хвильовий процесор являє собою архітектурну структуру, що розширюється. Хвильовий матричний процесор має більшість переваг систолічного процесора, а саме: простора конвеєризація та багатопроцесорна обробка, регулярність та модульність. Крім того, він має можливість асинхронної обробки, яка основана на управлінні даними та притаманна ЕОМ з управлінням потоком даних, а це дає змогу спростити важливу проблему невизначеності синхронізації у матричних процесорах на ПВІС.

4.2.2. Порівняння із систолічними процесорами

Систолічні та хвильові процесори мають багато спільного, зокрема використання спільної кількості модульних та локально зв’язаних процесорів, що забезпечують масову конвеєризацію та паралельну обробку. Відмінності полягають у схемному виконанні (наприклад, у приладах тактового генератора і буферів), можливостях архітектурного розширення, ефективності

конвеєра, програмування мовою високого рівня та спроможності справлятися з часовою невизначеністю у відмовостійких схемах.

Синхронізація. Схема синхронізації є критичним фактором для матричних систем великого обсягу, бо при глобальній синхронізації часто трапляється розфазування синхронізуючих імпульсів, що значною мірою залежить від розмірів систолічної матриці. Однак асинхронній моделі з керованими даними притаманні фіксована часова затримка і додаткові апаратні витрати на квитування та встановлення зв'язку.

Швидкість обробки та ступінь конвеєризації

Обчислення та управління даними у хвильовому процесорі можна прискорити за рахунок конвеєризації. Це особливо корисно в тому випадку, коли окремі ПЕ характеризуються різними рядками обробки. Оптимальна систолічна схема потребує середнього значення конвеєрного такту $\alpha = 3$. Унаслідок асинхронної обробки хвильових обчислень середній конвеєрний такт для тієї самої мережі буде становити $\alpha = 2{,}5$. Крім того, у багатьох задачах час обчислення залежить від надходження даних. У синхронному процесорі повинен бути вибраний «найгірший» такт, тоді як асинхронна обробка дає змогу виконати ті самі обчислення за менший час. Ця можливість дуже важлива, наприклад, у багатьох операціях з розрядженими матрицями. У середовищі обробки, залежно від даних, звичайне множення матриць потребує набагато менше часу, ніж множення на регулярній решітці. Тому для збільшення швидкості обчислень можуть бути застосовані способи хвильової обробки. Моделювання розв’язання задачі мінімізації рекурсивним способом найменших квадратів показало, що хвильовий процесор здатний працювати удвічі швидше, ніж глобально синхронізований систолічний процесор.

Програмованість. Програмування хвильових процесорів означає визначення послідовності операцій для кожного ПЕ. Кожна операція містить такі специфікації:

1) тип обчислення (додавання, множення, ділення тощо);

2) вхідний канал даних (наприклад, північ, південь, схід, захід або внутрішній регістр);

3) вихідний канал даних.

Зазначимо, що при програмуванні систолічних процесорів з фіксованими сполученнями потрібні додаткові специфікації планування операцій, оскільки необхідна коректна синхронізація. Це, однак, не потрібно при програмуванні хвильових процесорів завдяки принципу їх робіт з управлінням даними. У цьому розумінні програмування хвильових процесорів легше, ніж систолічних. Програмування хвильового процесора може легко здійснюватись на основі опису обчислювальних структур у вигляді графа потоку даних та використання схожих мов, а саме матрична мова потоку даних.

Відмовостійкість. У той час, як для лінійних систолічних процесорів відомо безліч відмовостійких схем, двовимірні систолічні процесори взагалі не придатні для реалізації відмовостійкості, бо вони потребують глобального переривання робіт усіх ПЕ у випадку появи будь-якої помилки. Відомо, що деякі прийоми забезпечення відмовостійкості (повернення, зупинення виконання тощо) легше здійснювати в архітектурі потоку даних, ніж в інших мультипроцесорах. Оскільки хвильові процесори використовують властивість керованості даними, вони забезпечують ту саму придатність і при елімінації часової невизначеності у відмовостійкому середовищі, де діючі шляхи зв'язку змінюються з метою обходу ПЕ, що відмовили. При виникненні відмови завдяки властивості керованості даними подальше розповсюдження хвильового фронту може бути автоматично зупинено.

Пікова потужність та якість джерела живлення

У великих матричних процесорних системах часто недоцільно синхронізувати пересилання даних між великим числом ПЕ, бо передача між цими ПЕ відбувається водночас. Це може призвести до проблем наведення перешкод у шинах живлення через великі струми перевантаження, викликаних одночасним запитом або модифікацією стану компонентів процесора (наприклад буферів). З цього погляду хвильові процесори більш придатні для реалізації великомасштабних систем.

Систолічний чи хвильовий процесор? Вибір між хвильовим і систолічним процесором залежить від декількох важливих факторів, а саме: (глобальна) синхронізація, програмування,

складність апаратури, можливість розширення відмовостійкості і тестування. Кінцевий вибір між двома матричними процесорами залежить від специфіки їх застосування. У загальному випадку систолічний процесор корисний тоді, коли ПЕ являє собою прості модулі, бо апаратура встановлення зв'язку у хвильовому матричному процесорі буде становити помітну частку для таких застосувань. З другого боку, хвильовий процесор краще застосовувати у тих випадках, коли модулі ПЕ є більш складними, коли синхронізація великого матричного процесора стає неефективною або коли суттєвою є надійність обчислювального середовища.

Контрольні питання

1. Що таке систолічний масив?
2. Як відбувається синхронізація в систолічних масивах?
3. Які основні критерії функціонування систолічних масивів?
4. Які основні особливості побудови хвильових матричних процесорів?
5. Порівняти хвильові та систолічні процесори за ступенем конвеєризації.
6. Порівняти хвильові та систолічні процесори за ступенем програмування та відмовостійкості.

ВИСНОВКИ

Знання основ автоматизації проектування та вміння працювати із засобами автоматизованого проектування потрібно практично будь-якому інженерові-розроблювачеві. Комп'ютерами насичені проектні підрозділи, конструкторські бюро та офіси. Сучасні програми автоматизованого проектування призначені не тільки для створення креслень – вони мають набагато ширші можливості та використовуються практично у всіх галузях техніки. З їх допомогою можна створювати моделі різних конструкцій і перевіряти властивості моделі на комп'ютері до початку виробництва, що істотно знижує кількість помилок проектування та прискорює появу виробу на ринку.

Сучасний рівень розвитку обчислювальної техніки і засобів передачі інформації відкриває широкі можливості з розроблення і впровадження технічної бази автоматизації. Існуючі ж математичні засоби дають змогу формалізувати основні завдання управління.

Аналіз сучасних розподілених систем показує, що зросла роль часових факторів. Час, що система витрачає на доведення інформації про стан керованого процесу до пунктів обробки і прийняття рішення на основі отриманої інформації і доведення прийнятого рішення до виконавчих органів повинен зменшуватися.

Сучасні складні системи характеризуються винятковою складністю умов, у яких здійснюється управління. До них можна віднести значний обсяг завдань, що раптово виникають, жорсткий ліміт часу, відведеного на прийняття (уточнення) рішення щодо їх виконання; велику інтенсивність інформаційних потоків між різними ланками управління; високий динамізм зміни обстановки; обмеженість ресурсу, призначеного для розв’язання задач. Існують об'єкти, у яких час доведення інформації про стан керованого процесу до пунктів становить одиниці секунд. У таких системах час є найважливішим параметром, бо дуже часто потребує формування керуючих дій у реальному часі.

Бібліографічний список

1. Listrovoy, S. V. On the class of NP-complete problems and approach Baptism of discrete optimization and graph theory [Text] / S.V. Listrovoy // LAP LAMBERT Academic Publishing GmbH & Co. KG. – 2014. – 100 p.

2. The comparative analysis of a multiprobe microwave multimeters with involvement of processing by the Kalman filtering and the least-squares methods with regard for Re-reflection of probes [Text] / M. A. Miroschnik, O. B. Zaichenko, I. I. Klyuchnik, R. I. Tzekhmistro // Telecommunications and radio engineering. – 2015. – Vol. 74. – № 74 (1). – P. 79-86.

3. Мирошник, М. А. Проектирование диагностической инфраструктуры вычислительных систем и устройств на ПЛИС [Текст] : монография / М. А. Мирошник. – Харьков : ХУПС, 2012. – 188 с.

4. Listrovoy, S. V. General Approach to Solving Optimization Problems in Distributed Cjmputing Systems and Theory of Intelligence Systems Construction. [Text] / S. V. Listrovoy, S. V. Minykhin // Journal of Automation and Information Science. – 2010. – Vol. 42, N. 3. – P. 30-45.

5. Listrovoy, S. V. Algoritm of Sub Exponential Complexity for the SAT [Text] / S. V. Listrovoy V. M. Butenko // International Jornal of Computer and Information Technology (ISSN: 2279-0764) – September 2013. – Vol. 2. – Issue 05. – P. 211-215.

6. Теорія графів у задачах розподілу ресурсів. Кн. 1 Алгоритми та методи обчислень [Текст] : підручник / Ф. О. Демченко, С. В. Лістровий, М. І. Луханін, Р. В. Семчук. – Харків : УкрДАЗТ, 2008. – 119 с.

7. Методы и модели планирования ресурсов в GRID-системах [Текст] : монография / С. В. Листровой, С. В. Минухин, В. С. Пономаренко, С. В. Знахур. – Харьков : ИНЖЭК, 2008. – 408 с.

8. Мирошник, М. А. Решение задач диспетчеризации в распределенных телекоммуникационных системах [Текст] / М. А. Мирошник, В. Г. Котух, С. Н. Селевко // Радиотехника: Всеукр. межвед. науч.-техн. сб. – 2011. – Вып. 169. – С. 139–152.

9. Мирошник, М. А. Развитие современных направлений цифровых телекоммуникационных систем и сетей [Текст] / В. Г. Котух, М. А. Мирошник, С. Н. Селевко // Радиотехника: Всеукр. межвед. науч.-техн. сб. – 2011. – Вып. 165. – С. 254–258.

10. Miroschnik, M. A. Application of software complex for query processing in the database management system with a view of dispatching problem solving in Grid systems [Text] / M. A. Miroschnik, V. G. Kotukh, S. N. Selevko // Telecommunications and radio engineering. – 2013. – Vol. 27, № 10. – P. 875-891.

11. Методы и модели планирования ресурсов в GRID-системах [Текст] : монография / С. В. Листровой, С. В. Минухин, В. С. Пономаренко, С. В. Знахур. – Харьков : ИНЖЭК, 2008. – 408 с.

12. Мирошник, М. А. Методы повышения отказоустойчивости телекоммуникационных систем. [Текст] / В. Г. Котух, М. А. Мирошник, С. Н. Селевко // Радиотехника: Всеукр. межвед. науч.-техн. сб. – 2011. – Вып. 166. – С. 259–268.

13. Листровой, С. В. Анализ сетевого трафика стека протоколов TCP/IP на основе динамической модели [Текст] / С. В. Листровой, С. В. Моцний // Інформаційно-керуючі системи на залізничному транспорті. – 2014. – № 5(108). – С. 10–14.

14. Подход к проектированию компьютерных систем с интеллектуальной диагностической инфраструктурой [Текст] / М. А. Мирошник С. Г. Карпенко, М. А. Ковалева, С. В. Панченко // Інформаційно-керуючі системи на залізничному транспорті. – 2011. – № 6. – С. 51–59.

15. Listrovoy, S. V. Znakhur Investigation of the Scheduler for Heterogeneous Distributed Computing Systems based on Minimal Cover Method [Text] / S. V. Listrovoy, S. V. Minykhin // Internation Journal of Computer Aplication (0975-8887). – August 2012. – Vol. 51, No. 19. – P. 123–127.

16. Мирошник, М. А. Синтез детерминированных проверяющих тестов для телекоммуникационных систем на одномерных сетях клеточных автоматов [Текст] / М. А. Мирошник, Я. Ю. Королева // Технология приборостроения. – 2012. – № 1. – С. 30–34.

17. Бережная, М. А Синдромно-сигнатурное тестирование микроэлектронных устройств [Текст] / М. А. Бережная,

М. Г. Рыжикова // Физические и компьютерные технологии: XI Междунар. науч.-техн. конф. (Харьков, 2-3 июня 2005 г.). – Харьков : ХНУРЭ, 2005. – С. 113.

18. Мирошник, М. А. Отказоустойчивость распределенных телекоммуникационных систем. [Текст] / М. А. Мирошник, В. Г. Котух, С. Н. Селевко // Радиотехника: Всеукр. межвед. науч.-техн. сб. – 2011. – Вып. 168. – С. 51–55.

19. Listrovoy, S. V. Jn Cjrrelation of P and NP-Classes [Text] / S. V. Listrovoy // I.J. Modern Education and Computer Science. 2012. – № 3. – P. 21-27.

20. Мирошник, М. А. Синтез проверяющих тестов для телекоммуникационных сетей на основе циклических отличительных последовательностей [Текст] / М. А. Мирошник, Я. Ю. Королева // Системи обробки інформації. Інформаційні проблеми акустичних, радіоелектронних та телекомунікаційних систем. – 2012. – Вип. 6 (104). – С. 108–113.

21. Model of influences of sensor reflections on the accuracy of microwave reflectometer [Text] / M. A. Miroschnik, I. Klyuchnyk, R. Tsekhmistro, Z. Warsza, O. Zaichenko // PAK (Pomiary Automatyka Kontrola). – 2014. – Vol. 60. – P. 223-225.

22. Pseudoexhaustive tpg based on nonlinear feedback shift registers [Text] / M. Berezhna, L. Derbunovsch, M. Ryzhskova, D. Tatarenko // Інформаційно-керуючі системи на залізничному транспорті. – Харків, 2005. – № 5. – С. 54-59.

23. Мирошник, М. А. Исследование методов диагностирования сложных систем [Текст] / М. А. Мирошник, Ю. Н. Салфетникова // Системи обробки інформації. Інформаційні проблеми акустичних, радіоелектронних та телекомунікаційних систем. – 2012. – Вип. 6 (104). – С. 70-75.

24. Решение задач диспетчеризации в распределенных телекоммуникационных системах [Текст] / М. А. Мирошник, В. Г. Котух, С. Н. Селевко, В. В. Васильєв, Є. А. Ралдугін // Радиотехника: Всеукр. межвед. науч.-техн. сб. – 2011. – Вып. 169. – С. 139–152.