КІБЕРНЕТИКА

УДК 519.712, 004.4′24

П.І. АНДОН
Інститут програмних систем НАН України, Київ, Україна, e-mail: *andon@isofts.kiev.ua.*

А.Ю. ДОРОШЕНКО
Інститут програмних систем НАН України, Київ, Україна; Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна, e-mail: *anatoliy.doroshenko@gmail.com*.

П.А. ІВАНЕНКО
Інститут програмних систем НАН України, Київ, Україна, e-mail: *paiv@ukr.net*.

О.А. ЯЦЕНКО
Інститут програмних систем НАН України, Київ, Україна, e-mail: *oayat@ukr.net*.

АЛГОРИТМІЧНІ АЛГЕБРИ ГЛУШКОВА ТА АВТОМАТИЗАЦІЯ ПРОЄКТУВАННЯ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

Анотація. Викладено огляд результатів, отриманих із застосуванням алгебри алгоритміки та засобів автоматизації розроблення програм для мультипроцесорних платформ. Алгоритміка ґрунтується на теорії алгебр алгоритмів та орієнтована на розв'язання широкого кола прикладних задач і розроблення інструментарію для автоматизованого проєктування та синтезу класів алгоритмів і програм. Загальність алгоритміки базується на різноманітності інтерпретацій схем алгоритмів і забезпечує можливості застосування алгоритміки та її засобів для розв'язування задач з різних предметних галузей. Поєднання алгоритміки та техніки правил переписування дало змогу розробити методи і засоби, орієнтовані на автоматизоване проєктування, перетворення, синтез та налаштовування програм для різноманітних платформ (багатоядерні процесори, графічні прискорювачі, програмовані логічні інтегральні схеми).

Ключові слова: алгебра алгоритмів, паралельні обчислення, проєктування та синтез програм, автоматичне налаштовування програм.

ВСТУП

Натепер паралельні обчислення на мультипроцесорних системах є основним джерелом забезпечення високої продуктивності обчислень під час розв'язання складних науково-технічних проблем. За останні роки архітектура мультипроцесорних систем помітно ускладнилась через урізноманітнення їхніх складових, що включають багатоядерні центральні мікропроцесори, графічні прискорювачі, мультипроцесорні кластери та інші структурні елементи. Одним із перспективних напрямів розроблення і дослідження систем паралельних і розподілених обчислень є побудова програмних абстракцій у вигляді формальних мов і моделей, зокрема, з використанням алгебро-алгоритмічного підходу.

Алгебро-алгоритмічне програмування — напрям української алгебро-кібернетичної школи автоматизації програмування і проєктування, що ґрунтується на фундаментальних ідеях і результатах академіка Віктора Михайловича Глушкова, отриманих у 60-ті роки минулого сторіччя, з теорії автоматів та формальних перетворень програм для оптимізації обчислень [1, 2]. У подаль-

шому цей напрям розвивали його колеги, учні та послідовники, зокрема, в працях колективу Інституту кібернетики ім. В.М. Глушкова НАН України, очолюваного К.Л. Ющенко і Г.О. Цейтліним [3–5], із розроблення систем алгоритмічних алгебр (САА) і синтезатора програм «Мультипроцесист», а також у працях колективу цього ж інституту, очолюваного О.А. Летичевським та Ю.В. Капітоновою [6–8], з автоматизації проєктування обчислювальних машин і алгебраїчного програмування. Останніми десятиліттями під керівництвом І.В. Сергієнка, О.М. Хіміча та В.Г. Тульчинського в інституті було створено потужну науково-технічну базу з дослідження інформаційних технологій та застосування високопродуктивних обчислень, зокрема, і їхніх алгебраїчних аспектів [9–13]. Дослідження з алгебро-алгоритмічного програмування проводилися також в Інституті програмних систем НАН України колективом, очолюваним П.І. Андоном та А.Ю. Дорошенком, у напряму розроблення теорії, методів та інструментальних засобів автоматизації проєктування паралельних програм [14, 15].

Специфічною рисою алгебри алгоритміки (АА) [16] є формалізація процесів проєктування та синтезу алгоритмів і програм. Ці об'єкти проєктуються в термінах регулярних схем — алгебраїчних подань у САА. Останні використовують для формалізованого проєктування алгоритмів, програм та об'єктів (абстрактних типів даних і пам'яті) у термінах неінтерпретованих і частково інтерпретованих схем, що називаються стратегіями оброблення. Схемне подання об'єктів формує знання про засоби їхнього проєктування та генерації. Поряд із синтезом знань в АА наявні засоби прогнозування та декомпозиції, а також моделювання і перевірки ефективності поведінки моделей.

У [14, 15] запропоновано теорію, методологію та інструментарій для автоматизованого проєктування, генерації та перетворення послідовних та паралельних програм, що ґрунтуються на засобах алгебр алгоритмів та переписувальних правил. Метою цієї роботи є огляд результатів, отриманих за останнє десятиріччя в рамках алгебри алгоритміки та відповідних інструментальних засобів програмування для мультипроцесорних обчислювальних систем і мереж.

1. АЛГЕБРИ АЛГОРИТМІВ ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ ПРОЄКТУВАННЯ ВИСОКОПРОДУКТИВНИХ ПРОГРАМ

Подібно до алгебраїчних специфікацій взагалі САА Глушкова орієнтовані на аналітичну форму подання алгоритмів і формалізовану трансформацію цих подань, зокрема, для оптимізації алгоритмів за заданими критеріями. Отже, САА Глушкова (алгебра Глушкова) є двоосновною алгеброю $GA = \langle \{Pr, Op\};\ \Omega_{GA} \rangle$, де основи Pr і Op — множини логічних умов і операторів, визначені на інформаційній множині IS; Ω_{GA} — сигнатура операцій. Оператори є відображеннями (можливо, частковими) інформаційної множини в себе, логічні умови — предикати на множині IS, які набувають значень тризначної логіки $E_3 = \{0, 1, \mu\}$, де 0 — хибність, 1 — істина, μ — невизначеність. Сигнатура $\Omega_{GA} = \Omega_1 \cup \Omega_2$ складається з системи Ω_1 логічних операцій, що набувають значень у множині Pr, і системи Ω_2 операцій, що набувають значень у множині операторів Op.

До логічних операцій системи $\Omega_1 \subset \Omega_{GA}$ належать узагальнені Булеві операції: диз'юнкція $u \vee u'$, кон'юнкція $u \wedge u'$, заперечення $\overline{u}$, а також операція прогнозування $A \cdot u$ (лівого множення умови u на оператор A), яка полягає в перевірці умови u після виконання оператора A.

Основними операціями, що входять у систему $\Omega_2 \subset \Omega_{GA}$ і породжують оператори з основи Op, є такі:

— композиція $A * B$ — бінарна операція на множині Op, що полягає в послідовному застосуванні операторів $A \in Op$, $B \in Op$;

— альтернатива (розгалуження) $([u]A, B)$ — тернарна операція, що залежить від умови $u \in Pr$ і операторів $A \in Op$, $B \in Op$. Ця операція породжує оператор $C = ([u]A, B)$ такий, що

$$C(m) = \begin{cases} A, & \text{якщо } u(m) = 1, \\ B, & \text{якщо } u(m) = 0, \\ W, & \text{якщо } u(m) = \mu, \end{cases}$$

для кожного $m \in IS$, де W — невизначений оператор;

— цикл $\{[u]A\}$ — бінарна операція, що залежить від умови $u \in Pr$ і оператора $A \in Op$. Сутність цієї операції полягає в циклічному застосуванні оператора A за хибного u доти, поки умова u не стане істинною.

У САА фіксується система твірних, яка є скінченною функціонально повною сукупністю логічних умов і операторів. За допомогою цієї сукупності та суперпозицій операцій, що входять в Ω_{GA}, породжуються довільні оператори і логічні умови з множин Pr та Op. Зафіксуємо базис САА: $\mathfrak{A} \subset Pr \cup Op$.

Регулярною схемою називається суперпозиція операцій, що входять у сигнатуру Ω_{GA} і елементів базису $\mathfrak{A}$.

Паралельні обчислення в САА формалізують за допомогою операцій асинхронної диз'юнкції, контрольних точок та синхронізаторів [3, 14].

Засоби САА покладені в основу мови САА/1 [3–5, 14, 15], призначеної для багаторівневого структурного проєктуванняя, документування послідовних і паралельних алгоритмів та програм. Перевагами мови САА/1 є можливість описувати алгоритми у стилі, близькому до природної мови, а також наявність автоматизованого перекладу у довільну мову програмування.

Основними операторними конструкціями мови є такі:

— композиція "*operator*1", "*operator*2";

— альтернатива IF '*condition*' THEN "*operator*1" ELSE "*operator*2" END IF;

— цикл FOR (*counter* FROM *start* TO *fin*) "*operator*" END OF LOOP.

Подання алгоритмів мовою САА/1 називають САА-схемами.

Зазначимо концептуальну близькість АА до алгебраїчної алгоритміки (АлгА) [18], породжувального програмування (ПП) [19], а також до формальних методів розроблення програм [20–22].

Можна віднести АлгА до методів спадного проєктування алгоритмів і програм, тоді як ПП — до їхнього висхідного проєктування. Зазначена АлгА є формалізованим підходом до опису методів оброблення математичних (алгебраїчних) об'єктів. Вона поєднує різні алгебри та алгоритми оброблення даних, що використовують під час доведення основних теорем у відповідних алгебрах.

Призначення ПП полягає у створенні різних предметних областей (ПрО) та їхньої інтеграції з використанням абстракцій, біології та екології програмування. Абстракція є аналітичним поданням знань для вибраної ПрО (наприклад, формули у відповідних алгебрах). Біологія програмування в ПП полягає в наявності «генетичних» зв'язків між близькими ПрО. Екологія є засобом автоматизації побудови різних ПрО та їхньої інтеграції.

Як і обидва згадані напрями, АА призначена для розв'язання тих самих проблем, тобто для формалізації знань про ПрО алгебраїчними засобами [16, 17]. Біологічний аспект в АА відображений в теорії клонів (сімей алгебр), а екологічний — у системі інструментальних засобів автоматизації проєктування та синтезу програмного забезпечення. Однак, на відміну від АлгА і ПП, в АА розглядають три взаємозалежні форми подання алгоритмічних знань [14, 16]: аналітичну — у вигляді алгебраїчної формули (регулярної схеми); природно-лінгвістичну — текст, близький до природної мови; візуальну — за допомогою графу (граф-схеми).

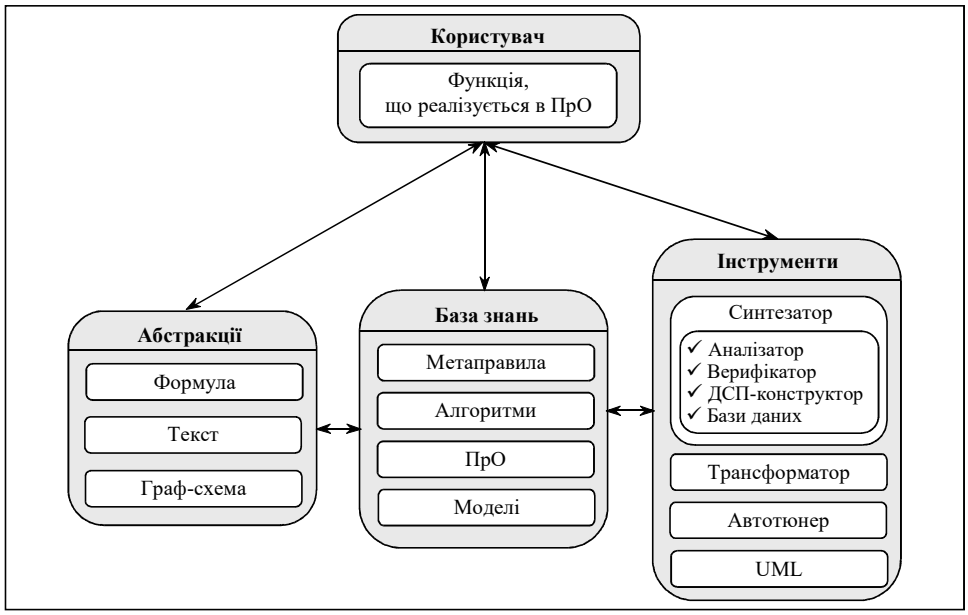

Рис. 1. Засоби проєктування та синтезу програм, що використовуються в рамках АА

Відзначаючи близькість АА до АлгА і ПП, варто звернути увагу на концептуальну цілісність її парадигми, яка багато в чому визначає переваги цієї парадигми порівняно зі згаданими напрямами.

Інструментальні засоби підтримки АА містять діалогові синтаксично-правильні конструктори (ДСП-конструктори), синтаксичні аналізатори, синтезатори об'єктів (рис. 1). Проблеми аналізу та синтезу були розвинуті у класичній теорії автоматів. Зазначимо потребу в декомпозиції (аналізі) об'єктів для можливого наступного збирання (синтезу) нового знання (шаблони-схеми, стратегії оброблення, інтерпретації тощо). На відміну від традиційних синтаксичних аналізаторів, ДСП-метод орієнтований не на пошук і виправлення синтаксичних помилок, а на унеможливлення їхньої появи в процесі побудови алгоритму. Основна ідея методу полягає в конструюванні схем за рівнями згори донизу з використанням суперпозиції мовних конструкцій САА. На кожному кроці конструювання система надає користувачу на вибір лише ті конструкції, підстановка яких у текст проєктованого алгоритму не порушує синтаксичної правильності схеми. На основі побудованої схеми алгоритму виконується автоматична генерація тексту програми цільовою мовою програмування.

Як відомо, процес синтезу, складається з двох основних етапів:

— фіксація контейнерів — базових понять у вибраній ПрО (семантичні ідентифікатори, інтерпретації, їхні реалізації тощо);

— генерація стратегій та алгоритмів оброблення за допомогою мов проєктування та синтезу програм, що ґрунтуються на відповідних алгебрах алгоритмів.

Схеми — форми подання проєктованих об'єктів, що ґрунтуються на застосуванні певних метаправил: згортки (абстрагування), розгортки (деталізації), переінтерпретації (згортки–розгортки) і трансформації (застосування рівностей — тотожностей, квазітотожностей та співвідношень, що характеризують властивості операцій, які входять у сигнатуру САА).

Коректність модифікацій випливає із функціональної еквівалентності схем, отриманих у результаті модифікацій на основі використання згаданих метаправил, що й визначає їхню семантичну правильність.

Залежно від сигнатури операцій алгебра, покладена в основу вибраних засобів проєктування, може бути орієнтована на послідовне або паралельне функціонування проєктованих об'єктів. Мультиоброблення тісно пов'язане із розв'язанням на рівні схем проблеми тупиків (клінчів), виникненням подій — замкнутих умов проходження відповідних контрольних точок. Таким чином, поставлена проблема клінчів виявляється пов'язаною із задачею виявлення фіктивних ітераційних конструкцій. Розв'язання обох задач вимагає розпізнавання відповідного розташування контрольних точок і синхронізаторів. Інакше кажучи, потрібно забезпечити перевірку виконання подій і засобів затримки процесів обчислень у взаємозалежних паралельних гілках. Організація паралелізму на рівні схем залежить також від використання засобів, розвинених у теорії замкнутих і локально-замкнутих логічних умов (або монотонних операторів та їхніх узагальнень).

Зазначимо, що вже створені і далі розвиваються інструментальні засоби проєктування об'єктів, які належать різним ПрО. При цьому об'єкти, що проєктуються, мають загальну структуру, задану трансформаційними перетвореннями, а також згадані раніше інтегровані форми подання: аналітичну, текстову і графсхемну. Ці форми відображають різні взаємодоповнювальні та інструментально підтримувані аспекти проєктування об'єктів. Зокрема, розроблено інтегрований інструментарій проєктування та синтезу програм IDS (Integrated toolkit for Designing and Synthesis of programs) [14]. Інструментальні засоби призначені для проєктування послідовних та паралельних схем алгоритмів в САА та генерації відповідних програм у цільових мовах програмування C++, Java та ін.

Однією з важливих проблем у рамках алгебро-алгоритмічного підходу є підвищення адаптивності програм до конкретних умов їхнього використання. Зокрема, вона розв'язується застосуванням параметрично-керованої генерації алгоритмів на основі специфікацій більш високого рівня — гіперсхем [14, 15], а також засобів автоматичного налаштовування (автотюнінгу) програм на цільове середовище виконання [23].

Процес проєктування об'єктів передбачає інтеграцію інструментарію АА з іншими засобами проєктування (мова UML, Rational Rose [24, 25], бібліотеки об'єктів). Зауважимо взаємодоповнюваність інтегрованих інструментальних засобів з погляду їхнього незалежного і спільного використання, зокрема, для розроблення спеціальних бібліотек, орієнтованих на підтримку синтезу об'єктів різних ПрО.

У середині 90-х років минулого сторіччя у рамках подальшого розвинення АА на основі відомих методів і технологій програмування було побудовано та досліджено метаалгебри (клони) [14, 16], що охоплюють сім'ї алгебр алгоритмів (структурних, неструктурних та ін.). З кожною із алгебр, що входить у той або інший клон, можуть бути пов'язані свої інструментальні засоби, що відповідають вибраним ПрО і методам проєктування. Перші ПрО, створені з використанням підходу, що розвивається, були пов'язані з алгоритмізацією низки задач символьного оброблення (сортування, пошук, мовне процесування). Потім було розроблено згаданий інтегрований інструментарій проєктування та синтезу програм, який забезпечує автоматизоване конструювання паралельних багатопотокових та розподілених програм у різних ПрО (зокрема, метеорологічне прогнозування).

Нещодавні публікації із застосування САА включають також генерування предметних унітермів формул алгебри алгоритмів [26], граматичний аналіз символьних обчислень виразів логіки висловлювань [27], застосування алгоритму Джонсона для пошуку найкоротших шляхів між усіма парами вершин графу [28].

Схеми алгоритмів та інтерпретації утворюють базу знань (БЗ), яка відображає сутність вибраної ПрО. До БЗ належать також співвідношення, тотожності та

квазітотожності, які використовуються в процесі перетворення схем. У [29] наведено результати експерименту з використання онтологій для подання бази алгебро-алгоритмічних знань. За допомогою онтології описуються основні об'єкти розроблюваної програми з вибраної ПрО — дані, функції та взаємозв'язки між функціями.

Проєктування та синтез об'єктів за допомогою алгебраїчних схем сприяє розв'язанню низки проблем, що виникають і в ПП:

— формування спеціалізованих бібліотек базових понять вибраної ПрО відповідає концепції родового програмування, яке забезпечує інтерпретацію змінних, що входять у проєктовані схеми;

— предметно-орієнтовані розширення — похідні мовні конструкції, які забезпечують фіксацію предметно-орієнтованих абстракцій, можна трактувати як створення відповідних бібліотек-розділів БЗ, що складаються із часто використовуваних у цій ПрО схемних проєктів (суперпозицій сигнатурних операцій заданої алгебри схем), тобто аналоги бібліотек розширень у ПП;

— зазначені бібліотеки розширень містять, зокрема, ефективні для вибраної ПрО оптимізувальні перетворення, методи тестування й налагодження, редагування тощо;

— синтаксичний аналіз схем забезпечується, по-перше, наявністю аналізаторів об'єктів у відповідних мовних процесорах проєктування і синтезу, а по-друге, наявністю діалогових конструкторів синтаксично правильних схем або дерев їхнього граматичного розбору. Зазначимо, що забезпечення синтаксичної правильності поширюється на всі взаємозалежні форми подання об'єктів (аналітичну, текстову і граф-схемну);

— створення синтезаторів об'єктів та бібліотек розширень (абстракцій) орієнтоване на застосування в різних ПрО. Бібліотеки схем (абстракцій) істотно розширюють можливості мовних процесорів, полегшують розуміння програмного коду. Крім того, алгебраїчний (схемний) підхід формалізує методи і технології, є інструментом інтеграції середовищ, сприяє створенню і нагромадженню різних бібліотек, що входять до складу БЗ.

Таким чином, алгебраїчний підхід до проєктування і синтезу об'єктів відповідає основним принципам ПП та водночас має низку істотних переваг, а саме:

— алгебра є не тільки зручною, зрозумілою, точною і компактною мовою опису об'єктів, а й орієнтованою на формалізовані перетворення об'єктів для їхнього якісного поліпшення за вибраними критеріями (пам'ять, швидкодія, апаратні ресурси тощо);

— алгебраїчний формалізм та інструменти, що ґрунтуються на ньому, орієнтовані на конструювання об'єктів (включаючи не тільки схеми, але й вхідні та цільові мови, а також розмаїтість різних ПрО);

— розроблено алгебраїчну теорію клонів [14, 16], у рамках якої відповідно до відомих методів проєктування об'єктів побудовано сім'ї алгебр різних типів і предметної орієнтації, покладених в основу мов специфікацій надвисокого рівня. У перспективі це означає, що прикладні програмісти матимуть можливість створювати власні зручні мови проєктування та інструменти синтезу об'єктів, які відповідають вибраній ПрО та цільовому середовищу реалізації.

2. ЗАСТОСУВАННЯ ІНСТРУМЕНТАРІЮ АЛГЕБР АЛГОРИТМІВ

Інструментальні засоби автоматизованого проєктування, генерації та оптимізації алгоритмів і програм, що ґрунтуються на використанні САА, було застосовано для розроблення програмного забезпечення для багатоядерних про-

цесорів загального призначення [23], хмарних платформ [29], графічних прискорювачів [30], програмованих логічних інтегральних схем (ПЛІС) [31].

2.1. Проєктування та автоматичне налаштовування програм для багатоядерних процесорів. У [23] запропоновано метод, що поєднує засоби алгебри алгоритмів та автотюнінгу програм і спрямований на максимізацію ефективності паралельних програм за часом виконання. Застосування підходу продемонстровано на прикладі проєктування та оптимізації паралельної програми сортування для виконання на багатоядерному процесорі.

Автотюнінг [32] — це автоматичне налаштовування програми на середовище виконання, що використовує емпіричний підхід для отримання якісної оцінки оптимізовуваної програми (під якістю зазвичай розуміють швидкодію і точність результату). Він автоматизує пошук оптимального варіанта програми з множини передбачених можливостей, виконуючи кожний варіант та вимірюючи його продуктивність на заданій паралельній архітектурі. Основною перевагою цього підходу є високий рівень абстракції — програма оптимізується без знання деталей апаратної реалізації, таких як кількість процесорних ядер, розмір кешу та швидкість доступу до пам'яті. Натомість, автотюнінг оперує високорівневими поняттями з ПрО, такими як кількість і розмірність незалежних задач, або особливостями алгоритму розв'язання задачі.

Розроблена система автотюнінгу програм TuningGenie [23] сприймає на вході вихідний код програми, відмічений прагмами (рис. 2). Прагми описують конфігурації і трансформації програми, що впливають на її продуктивність. Прагми задають вручну розробники програм із використанням коментарів мови Java спеціального вигляду. Спочатку інформація з усіх прагм у коді вхідної програми збирається синтаксичним аналізатором і на її основі генеруються усі можливі конфігурації застосунку. Далі для кожної конфігурації генерується відповідна варіація програми і вимірюється швидкодія. На основі отриманих результатів визначається оптимальна конфігурація і генерується оптимальний варіант застосунку. Для трансформацій вихідного коду використовується система TermWare, що ґрунтується на техніці переписувальних правил [33] і обробляє тексти програм, подані у вигляді термів.

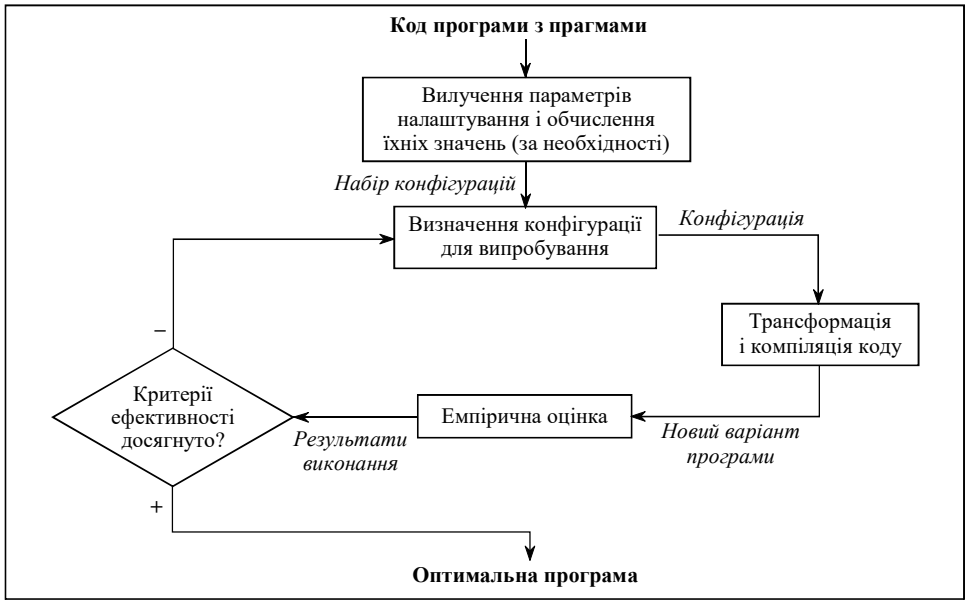

Рис. 2. Послідовність дій автотюнінгу в системі TuningGenie

Однією з прагм є tuneAbleParam, яка задає область пошуку для оптимального значення числової змінної. Наприклад, для змінної threadCount межі вибору значень [1…16] із кроком 1 задаються так:

“Comment (tunableParam name=threadCount start=1 stop=16 step=1)”
“Declare a variable (threadCount) of type (int) with initial value (1)”.

Ця прагма застосовується насамперед до алгоритмів, що використовують геометричний паралелізм, оскільки дає змогу легко підібрати оптимальну декомпозицію області обчислень, тобто «зернистість» алгоритму. Також за допомогою прагми tuneAbleParam можна підібрати деяке граничне значення для зміни схеми обчислень програми.

Далі наведено приклад САА-схеми гібридного паралельного алгоритму сортування, що підлягає подальшому налаштуванню за допомогою TuningGenie. Алгоритм застосовує сортування злиттям або вставками залежно від розміру блока (insertionSortThreshold) вхідного числового масиву. Схема містить дві прагми tuneAbleParam, які вказують область пошуку для оптимальних значень змінних insertionSortThreshold і mergeSortBucketSize. На основі схеми виконується генерація коду мовою Java, який далі оптимізується в TuningGenie.

```
“Parallel Hybrid Sorting (arr)”
====“Comment(tuneAbleParam name=insertionSortThreshold start=10 stop=200
            step=10)”;
    “Declare a variable (insertionSortThreshold) of type (int) with
    initial value (100)”;
    “Comment(tuneAbleParam name=mergeSortBucketSize start=5000
            stop=1000000 step=5000)”;
    “Declare a variable (mergeSortBucketSize) of type (int) with
    initial value (5000)”;
    IF ‘Length of the array (arr) is less or equal to (insertionSortThreshold)’
    THEN “insertionSort(arr)”
    ELSE IF ‘Length of the array (arr) is less or equal to (mergeSortBucketSize)’
      THEN “sequentialMergeSort(arr)”
      ELSE “concurrentMergeSort(arr)”
      END IF
    END IF
```

Як зазначалось раніше, підвищення адаптивності алгоритмів до конкретних умов їхнього використання може здійснюватись також за допомогою засобів гіперсхем, які є параметризованими алгоритмами для розв’язання певного класу задач. Установлення значень параметрів і подальша інтерпретація гіперсхем дає змогу одержати алгоритми, адаптовані до конкретних умов застосування [14, 15].

2.2. Засоби автоматизації програмування для графічних прискорювачів. У [30] розроблено програмний засіб для оптимізації обчислень (LoopRipper), що дає змогу в напівавтоматичному режимі здійснювати паралелізацію циклічних операторів програми для виконання обчислень на графічних прискорювачах. Здійснено буферизацію даних, синхронізовану із виконанням основного циклу, та побудований за допомогою системи переписувальних правил TermWare засіб інтегровано з інструментарієм проєктування та синтезу програм IDS.

Інструментарій розпаралелює вхідний послідовний цикл вигляду

```
“SEQUENTIAL LOOP”
==== FOR (i0 FROM 0 TO I0)
       FOR (i1 FROM 0 TO I1)
```

…
FOR (i_n FROM 0 TO I_N)
“$F(\bar{i}, p^{in}(\bar{i}), p^{out}(\bar{i}))$”
END OF LOOP,

де $I_0, I_1, \ldots, I_N$ — множини значень індексів $i_0, i_1, \ldots, i_N$; $N+1$ — глибина вкладеності циклів; $\bar{i} = \{i_j \mid j = 0, \ldots, N\}$; $p^*(\bar{i}) = P^* = \{p_j^*(\bar{i}) \mid j = 0 \ldots \# P^*\}$, $* \in \{in, out\}$ — множини вхідних та вихідних змінних, що описують оброблювані в циклі дані; F — функція, що обробляє дані; ітерації циклу незалежні.

Нехай T — кількість потоків, що будуть використані для виконання функції-ядра графічного прискорювача. Цикл, отриманий у результаті розпаралелювання, є таким:

“PARALLEL LOOP”
==== FOR (e FROM 0 TO L)
“*fill*(*inBuf*)”;
“*push*(*inBuf*)”;
“*kernel*(*e*, *inBuf*, *outBuf*)”;
“*pull*(*outBuf*)”;
“*unpack*(*outBuf*)”
END OF LOOP,

де L — кількість викликів ядра, яке вибирається найменшим можливим і таким, що $L \cdot T \geq G$ (тут G — загальна кількість ітерацій у початковому циклі); *fill* — функція заповнення буфера початкових даних *inBuf*; *push* — переміщення буфера початкових даних у пам’ять пристрою; *kernel* — виклик функції-ядра (остання містить виклик тіла початкового циклу “$F(\bar{i}, inbuf_{id}(\bar{i}), outbuf_{id}(\bar{i}))$”, де *id* — номер потоку); *pull* — функція, що переміщує оброблені дані з пам’яті пристрою до буфера оброблених даних *outBuf*; *unpack* — копіювання даних з буфера оброблених даних у відповідні змінні.

На рис. 3 зображено процес розпаралелювання програми за допомогою інструментарію LoopRipper.

Маючи списки вхідних та вихідних змінних, LoopRipper генерує функції kernel, fill та unpack, з яких паралельна програма компонується за допомогою інструментарію проєктування та синтезу програм. Генерація функцій здійснюється заміною параметрів вхідним та вихідним буферами даних і переобчисленням ітераторів у заданому початковому циклі. Таким чином, від користувача вимагається надати цикл та списки параметрів.

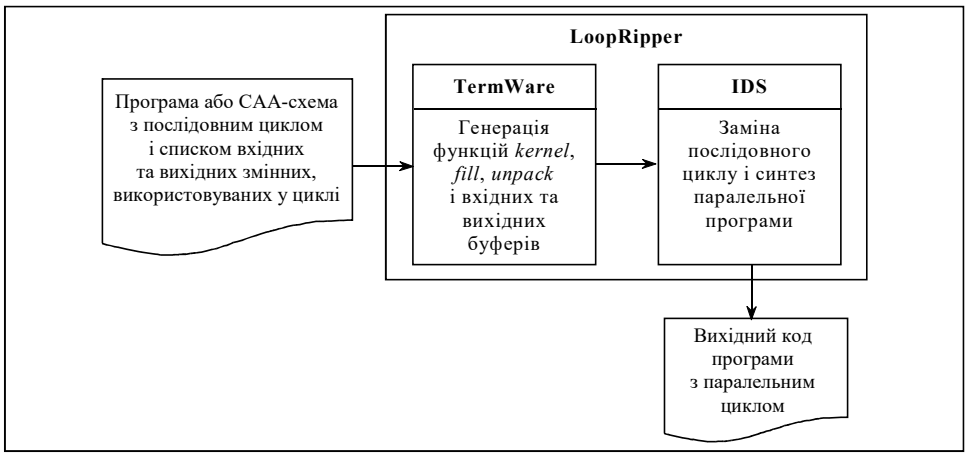

Рис. 3. Розпаралелювання програми за допомогою інструментарію LoopRipper

Запропонований метод застосовано для розпаралелювання послідовного циклу з глибиною гнізда, яка дорівніє чотирьом і входить до складу програми чисельного прогнозування погоди, а саме циклу інтерполяції початкових даних у вузлах локальної сітки. Проведено випробування розробленої системи на гетерогенному багатоядерному кластері.

2.3. Засоби автоматизації паралельного програмування для ПЛІС. У [31] запропоновано засоби автоматизованого проєктування та генерації програм для ПЛІС, що ґрунтуються на алгебрі алгоритмів, та продемонстровано їхнє застосування для розроблення генетичного алгоритму та штучного нейрона. Технологія розроблення застосунків для ПЛІС ґрунтується на поданні алгоритму мовою опису апаратури, наприклад VHDL [34], і автоматичному перекладі цього опису в специфікацію на рівні логічних таблиць та інших функціональних компонентів ПЛІС. Мова VHDL забезпечує високорівневу абстракцію для опису апаратних засобів завдяки наявності набору попередньо визначених типів даних і можливості створювати визначені користувачем ієрархічно організовані типи даних на базі основних, вбудованих у мову. Елементарні функції в ПЛІС зазвичай реалізуються як окремі проєкти або модулі, що містять інформацію про швидкість передавання даних і внутрішню структуру системи.

Паралельні обчислення, властиві для нейронних мереж, ефективно реалізуються із використанням мови VHDL. Кожний процес мовою VHDL має перелік сигналів, зміна яких спричиняє запуск процесу. На основі цього нейрони (окремі процеси) природно об'єднуються у мережу: виходи попереднього шару мережі запускають процеси, що відповідають наступному шару. Кожен нейрон представлений у вигляді окремого блока, що складається з кількох паралельних процесів, а нейронна мережа є багатопроцесорною системою.

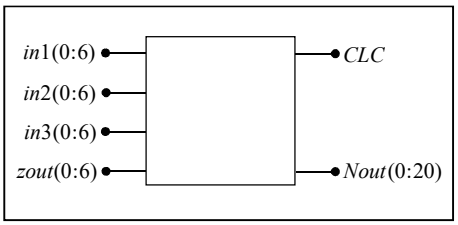

Рис. 4. Приклад блока генетичного алгоритму

Як приклад розглянемо блок генетичного алгоритму (рис. 4). Він складається з таких портів: *in*1, *in*2, *in*3, що отримують вхідні сигнали; *zout*, що зберігає значення, отримуване в результаті навчання; *Nout* (вихідний порт), що показує поточне значення виходу після кожної ітерації навчання. Сигнал *CLC* здійснює затримку в одну пікосекунду. Опис портів цього блока представлено такою САА-схемою:

ENTITY *genVHDL* is
"Signal (*in*1) direction (*in*) of type (*integer*) and range (–100 to 100) with initial value (63)";
"Signal (*in*2) direction (*in*) of type (*integer*) and range (–100 to 100) with initial value (– 82)";
"Signal (*in*3) direction (*in*) of type (*integer*) and range (–100 to 100) with initial value (70)";
"Signal (*zout*) direction (*in*) of type (*integer*) and range (0 to 100) with initial value (77)";
"Signal (*CLC*) direction (*inout*) of type (*bit*) with initial value ('1')";
"Signal (*Nout*) direction (*inout*) of type (*integer*) and range (–1048575 to 1048576);
END OF ENTITY

У блоці реалізовано оператори кросинговеру, мутації та селекції наступного покоління хромосом. Процес, що реалізує нейрон, спроєктований у вигляді такої САА-схеми:

```
PROCESS neuron (start) is
  DECLARATIONS (
    "Variable (LocalOut) of type (integer) and range (0 to 1023)";
    "Variable (lin) of type (integer)");
  (LocalOut := 100);
  (lin := (in1 * W(1) + in2 * W(2) + in3 * W(3)) / 16);
  FOR (a FROM 0 TO 49)
    IF (abs (lin) >= x(a)) AND (abs (lin) < x(a + 1))
    THEN (LocalOut := 50 + a); EXIT LOOP;
    END IF
  END OF LOOP;
  IF (lin < 0) THEN (LocalOut := 100 – LocalOut);
  END IF;
  IF (NOT (finishTeaching)) THEN
    (lout(i) := LocalOut);
    NFinish <= NOT NFinish;
  END IF;
  Nout <= LocalOut;
END OF PROCESS.
```

Тут *LocalOut* — поточний вихід нейрона, який в подальшому порівнюється з очікуваним результатом змінної *zout*; $W(1)$, $W(2)$, $W(3)$ — синаптичні ваги нейрона; i — номер хромосоми, на основі якої було сформовано вагові коефіцієнти перед запуском процесу, що формує вихід нейронної мережі.

На основі побудованих САА-схем в інтегрованому інструментарії виконано генерацію коду мовою VHDL.

ВИСНОВКИ

У статті викладено огляд результатів, отриманих у рамках наукового напряму алгебри алгоритміки та засобів автоматизації проєктування високопродуктивних програм для мультипроцесорних платформ, започаткованого академіком В.М. Глушковим. Алгоритміка базується на теорії алгебр алгоритмів і орієнтована на розв'язання широкого кола прикладних задач і розроблення програмних засобів для автоматизованого проєктування та синтезу класів алгоритмів і програм. Загальність алгоритміки базується на різноманітності інтерпретацій схем алгоритмів і забезпечує можливості застосування алгоритміки та її засобів для розв'язування задач у різних предметних галузях. Ефективність такого застосування багато в чому залежить від глибини розуміння семантики розроблюваних алгоритмів. Лише за цієї умови можлива адекватна інтерпретація засобів алгоритміки та її ефективність під час проєктування алгоритмів і програм. Поєднання алгоритміки та техніки переписувальних правил надало змогу розробити методи та засоби, орієнтовані на автоматизоване проєктування, перетворення, синтез та налаштовування програм для різноманітних платформ (багатоядерні процесори, графічні прискорювачі, програмовані логічні інтегральні схеми).

СПИСОК ЛІТЕРАТУРИ

1. Глушков В.М. Синтез цифровых автоматов. Москва: Физматгиз, 1962. 476 с.
2. Глушков В.М. Теория автоматов и формальные преобразования микропрограмм. *Кибернетика*. 1965. № 5. С. 1–9.
3. Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л. Методы символьной мультиобработки. Киев: Наук. думка, 1980. 252 с.

4. Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л. Алгебра. Языки. Программирование. 3-е изд. Киев: Наук. думка, 1989. 376 с.

5. Ющенко Е.Л., Цейтлин Г.Е., Грицай В.П., Терзян Т.К. Многоуровневое структурное проектирование программ: теоретические основы, инструментарий. Москва: Финансы и статистика, 1989. 208 с.

6. Глушков В.М., Капитонова Ю.В., Летичевский А.А. Автоматизация проектирования вычислительных машин. Киев: Наук. думка, 1975. 228 с.

7. Капитонова Ю.В., Летичевский А.А. Математическая теория проектирования вычислительных систем. Москва: Наука, 1988. 296 с.

8. Letichevsky A.A., Kapitonova Yu.V., Konozenko S.V. Computations in APS. *Theoretical Computer Science.* 1993. Vol. 119. P. 145–171.

9. Сергієнко І.В., Кривий С.Л., Провотар О.І. Алгебраїчні аспекти інформаційних технологій. Київ: Наук. думка. 2011. 400 с.

10. Сергієнко І.В. Наукові ідеї В.М. Глушкова та розвиток актуальних напрямів інформатики. Київ: Наук. думка, 2013. 287 с.

11. Петрик М.Р., Хіміч О.М., Бойко І.В. Високопродуктивні методи моделювання та ідентифікації складних процесів та об'єктів у багатокомпонентних неоднорідних середовищах. Київ: Ін-т кібернетики ім. В.М. Глушкова НАН України, 2020. 204 с.

12. Хіміч О.М., Мова В.І., Ніколайчук О.О., Попов О.В., Чистякова Т.В., Тульчинський В.Г. Інтелектуальний паралельний комп'ютер на процесорах Intel Xeon Phi нового покоління. *Наука та інновації.* 2018. Т. 14, № 6. С. 66–79.

13. Головинський А.Л., Маленко А.Л., Сергієнко І.В., Тульчинський В.Г. Енергоефективний суперкомп'ютер СКІТ-4. *Вісник НАН України*. 2013. № 2. С. 50–59.

14. Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А. Алгеброалгоритмические модели и методы параллельного программирования. Київ: Академперіодика, 2007. 634 с.

15. Andon P.I., Doroshenko A.Yu., Zhereb K.A., Yatsenko O.A. Algebra-algorithmic models and methods of parallel programming. Kyiv: Akademperiodyka, 2018. 192 p.

16. Цейтлин Г.Е. Введение в алгоритмику. Киев: Сфера, 1998. 310 с.

17. Захарія Л.М. Алгебро-алгоритмічні підходи до опису предметних областей та синтезу програмних середовищ для них. *Вісник Національного університету «Львівська політехніка». Сер. Інформаційні системи та мережі*. 2015. № 832. С. 376–384.

18. Naudin P., Quitté C. Algorithmique algébrique avec exercices corrigés. Paris: Masson, 1992. 469 p.

19. Czarnecki K., Eisenecker U. Generative programming: Methods, tools, and applications. Boston: Addison-Wesley, 2000. 864 p.

20. Roggenbach M., Cerone A., Schlingloff B.-H., Schneider G., Shaikh S.A. Formal methods for software engineering: Languages, methods, application domains. Cham: Springer, 2022. 524 p.

21. Wang J. Formal methods in computer science. New York: Chapman and Hall/CRC, 2019. 350 p.

22. Sannella D., Tarlecki A. Foundations of algebraic specification and formal software development. Berlin: Springer, 2012. 584 p.

23. Doroshenko A., Ivanenko P., Novak O., Yatsenko O. A mixed method of parallel software auto-tuning using statistical modeling and machine learning. Information and Communication Technologies in Education, Research, and Industrial Applications. ICTERI 2018. *Communications in Computer and Information Science*. 2019. Vol. 1007. P. 102–123. https://doi.org/10.1007/978-3-030-13929-2_6/.

24. Sundaramoorthy S. UML diagramming: A case study approach. Boca Raton: Auerbach Publications, 2022. 430 p.

25. Boggs W., Boggs M. Mastering UML with Rational Rose. Alameda: Sybex, 2002. 848 p.

26. Василюк А., Басюк Т. Система синтезу формул алгебри алгоритмів. *Інформаційні системи та мережі.* 2021. № 9. С. 11–22. https://doi.org/10.23939/sisn2021.09.011.

27. Литвин В.В., Бобик І.О., Висоцька В.А. Застосування системи алгоритмічних алгебр для граматичного аналізу символьних обчислень виразів логіки висловлювань. *Радіоелектроніка, інформатика, управління.* 2016. № 4. С. 77–89.

28. Погорілий С.Д., Слинько М.С. Створення і дослідження паралельних схем алгоритму Джонсона в технології GPGPU. *Проблеми програмування*. 2016. № 2–3. С. 105–112.

29. Doroshenko A.Yu., Yatsenko O.A., Ovdii O.M. Ontological and algebra-algorithmic tools for automated design of parallel programs for cloud platforms. *Cybernetics and Systems Analysis.* 2017. Vol. 53, N 2. P. 323–332. https://doi.org/10.1007/s10559-017-9932-8.

30. Дорошенко А.Ю., Яценко О.А., Бекетов О.Г. Алгоритм автоматизованого розпаралелювання циклічних операторів для графічних прискорювачів. *Проблеми програмування*. 2017. № 4. С. 28–36.

31. Doroshenko A., Shymkovych V., Yatsenko O., Mamedov T. Automated software design for FPGAs on an example of developing a genetic algorithm. *Proc. 17th Int. Conf. "ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer", ICTERI 2021* (28 Sept. – 2 Oct., 2021). CEUR-WS, 2021. P. 74–85.

32. Durillo J., Fahringer T. From single- to multi-objective auto-tuning of programs: Advantages and implications. *Scientific Programming*. 2014. Vol. 22, N 4. P. 285–297. https://doi.org/10.3233/SPR-140394.

33. Шевченко Р.С. Числення контекстних термів для систем переписування. *Проблеми програмування*. 2018. № 2–3. С. 21–30.

34. Godse A.P., Godse D.A. VHDL programming: Concepts, modeling styles and programming. Seattle: Amazon Digital Services LLC, 2020. 206 p.

P.I. Andon, A.Yu. Doroshenko, P.A. Ivanenko, O.A. Yatsenko

GLUSHKOV'S ALGORITHMIC ALGEBRAS AND AUTOMATED PARALLEL COMPUTING DESIGN

Abstract. An overview of the results obtained within the algebra of algorithms and tools for the automated development of programs for multiprocessor platforms is presented. Algorithmics is based on the theory of algorithm algebras and is focused on solving a wide range of applied problems and developing software tools for automated design and synthesis of classes of algorithms and programs. The generality of algorithms is based on the variety of interpretations of algorithm schemes and provides the possibility of applying algorithms and their tools for solving problems related to various subject domains. The combination of algorithmics and rewriting rules technique made it possible to develop methods and tools aimed at automated design, transformation, synthesis, and tuning of programs for various platforms (multicore processors, graphics processing units, field-programmable gate arrays).

Keywords: algebra of algorithms, parallel computation, program design and synthesis, software auto-tuning.

Надійшла до редакції 17.03.2023