https://doi.org/10.31891/2219-9365-2022-72-4-12
УДК 004.43:004.8

Оксана ЯШИНА
Хмельницький національний університет
https://orcid.org/0000-0001-7816-1662
Оксана ОНИШКО
Хмельницький національний університет
https://orcid.org/0000-0002-2125-4160

ПАРАДИГМИ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ В КОНТЕКСТІ ФУНКЦІОНАЛЬНОГО ПРОГРАМУВАННЯ

У статті здійснено аналіз низки парадигм паралельних обчислень, що підтримуються у різноманітних мовах програмування із описом вимог до мовної підтримки паралельного програмування. Також розглядаються основні принципи та обмеження, що виявляються при перенесенні техніки функціонального програмування на паралельні обчислення. Наприкінці описані вимоги до мовної підтримки.

Ключові слова: функціональне програмування, паралельні обчислення, мови програмування, парадигми програмування, мультипарадигмальність

Oksana YASHYNA, Oksana ONYSHKO
Khmelnytskyi National University

COMPARATIVE ANALYSIS OF PROGRAMMING PARADIGMS IN THE DEVELOPMENT OF SOFTWARE SYSTEMS BASED ON ARTIFICIAL INTELLIGENCE

The article analyzes a number of parallel computing paradigms supported in various programming languages with a description of the requirements for language support for parallel programming. Also, the main principles and limitations revealed when transferring the FP technique to parallel computing are considered. At the end, the requirements for language support are described.

For each of the difficult-to-solve problems of parallel computing, a separate convenient paradigm for its solution has already been formed, and a number of programming languages have been created that support such a paradigm, which confirms its status as a paradigm. The difference between the paradigms is manifested in the ordering of the importance of the means and methods used in solving individual tasks, individual tasks require a different ordering. At each moment of program development, one paradigm is usually used. Accordingly, a basic paradigm is distinguished in various programming languages. The requirements for solving rather complex tasks of parallel computing are associated with a number of difficult tasks, which entails the need to use different paradigms at different stages of their creation and phases of their life cycle. When switching to parallel programming technology, it is important to guarantee a practical result, which requires support for the full range of paradigms used at various stages of program development. The complexity of using different paradigms when solving one task is usually minimized by creating multilingual systems that allow, if necessary, the possibility of switching from one paradigm to another without the costs of mastering different interfaces. This leads to the expediency of creating a multi-paradigm parallel computing language that simultaneously supports all the main paradigms of parallelism and different levels of abstraction - from lower than is accepted in high-level languages to ultra-high-level languages.

Keywords: functional programming, parallel computing, programming languages, programming paradigms, multiparadigmality.

Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

Функціональне програмування – одна із парадигм, націлених не просто на отримання ефективної реалізації заздалегідь підготовлених алгоритмів, але й на вирішення нових завдань інформаційної обробки, які мають дослідницький компонент [1, 2]. Якщо кількість мов програмування стрімко зростає, то кількість парадигм не настільки велика. Вона допускає досить повне порівняння та вибір характеристик для чіткого поділу парадигм, з'ясування причин їхньої різноманітності та сфери їх успішного застосування [3]. Розглядаючи результати аналізу проблем, засобів та методів організації паралельних обчислень, а також підготовки багатопотокових програм для багатопроцесорних комплексів та розподілених систем, можна звернути увагу на явно широкий спектр постановок завдань та відповідних їм пріоритетів у прийнятті рішень на різних етапах розробки та налагодження програм. В результаті методика парадигмального аналізу дозволяє зводити постановки завдань до комплексів підзадач, що розвиваються автономно, оцінювати їх схожість та відмінності, облік яких необхідний при прогнозуванні складності процесів застосування програмованих рішень, починаючи з планування, вивчення та організації розроблення програм вирішення таких підзадач для довгоіснуючих програм (якими часто виявляються програми паралельних обчислень) [3].

Аналіз останніх досліджень та публікацій

Підгрунтям даного дослідження є праці як вітчизняних так і зарубіжних дослідників. Зокрема, Ю. М Тирчака, О. В. Корочкіна, В. В. Демчика, В. М. Коцовського, П. Купмана та інших.

Формулювання цілей статті

Метою роботи є: аналіз сучасних тенденцій функціонального програмування, що розглядається як метапарадигма рішення проблем організації паралельних обчислень та багатопотокових програм для багатопроцесорних комплексів та розподілених систем.

Виклад основного матеріалу

Розглядаючи ідеї функціонального програмування та практику їх застосування при аналізі проблем, засобів та методів організації паралельних обчислень, слід звернути увагу, що вже створено значну кількість мов і систем паралельного програмування. Використовуючи систематизацію парадигм програмування на основі відмінностей у пріоритетах прийняття програмованих рішень, можна зробити висновок, що однією з причин складності розробки програм паралельних обчислень є їхня прихована мультипарадигмальність [6]. Для розробки паралельної програми багато що потрібно продумувати по-різному одночасно в різних парадигмах, зручних для вирішення окремих підзадач без традиції та навичок рішення повного комплексу підзадач у єдиних умовах. Найбільш очевидно парадигмальні відмінності видно при вирішенні завдань масштабування обчислень на різні багатопроцесорні комплекси, синхронізації взаємодій над локальною та загальною пам'яттю в багатопотокових програмах, а також поданні природного асинхронного паралелізму рівня постановок завдань та досягнення високої продуктивності програм з урахуванням критерію повноти завантаження доступних багатопроцесорних комплексів або розподілених систем. Окремо має місце освітня проблема - ознайомлення з феноменами паралелізму, що дає фахівцям інтуїтивне розуміння методів вирішення особливо складних завдань. Нові роботи з функціонального програмування істотно націлені на пошук більш продуктивного розв'язання проблем паралельного програмування [1]. Створено помітну кількість мов та систем програмування, що дозволяють вирішувати окремі з цих завдань у межах конкретних парадигм, підтримка яких представлена різними мовами програмування.

Таким чином, для кожної із задач паралельних обчислень, що важко вирішуються, вже сформовано окрему зручну парадигму її вирішення та створено низку мов програмування, які підтримують таку парадигму, що підтверджує її статус як парадигми. Відмінність між парадигмами проявляється у впорядкуванні важливості засобів та методів, що використовуються при вирішенні окремих завдань, окремі завдання вимагають іншого впорядкування. У кожний момент розробки програми зазвичай використовується одна парадигма. Відповідно й у різних мовах програмування виділяється основна парадигма. Вимоги до вирішення досить складних завдань паралельних обчислень пов'язані із цілим рядом важких завдань, що тягне за собою необхідність використання різних парадигм на різних етапах їх створення та фазах їх життєвого циклу. При переході до технології паралельного програмування важлива гарантія отримання практичного результату, що вимагає підтримки повного спектру парадигм, які використовуються на різних етапах розробки програм. Трудомісткість використання різних парадигм при вирішенні одного завдання зазвичай мінімізується створенням багатомовних систем, що допускають при необхідності можливість переходу від однієї парадигми до іншої без витрат на освоєння різних інтерфейсів. Це призводить до доцільності створення мультипарадигмальної мови паралельних обчислень, що підтримує одночасно всі основні парадигми паралелізму та різні рівні абстрагування - від нижчого, ніж прийнято в мовах високого рівня до мов надвисокого рівня [1].

Як показує досвід декотрих мов програмування, зокрема Haskell, у таких випадках зручно декомпозувати визначення мов програмування на окремі підмови, що підтримують основні парадигми, націлені на конкретні моделі підготовки та представлення програм. Важливо на кожному етапі розробки програми локалізувати використання однієї парадигми, що характеризується відносно невеликим набором засобів та методів у рамках одного способу мислення. Кожна парадигма має своє наповнення семантичних категорій систем та своє впорядкування їхньої ролі в процесі програмування [9, 10]. Засоби багатопроцесорного програмування зазвичай спираються на дані та характеристики доступної архітектури, включаючи основний багатопроцесорний комплекс та взаємозв'язок між його елементами. Оперування комплексом дозволяє ініціювати процеси функціонування окремих процесорів, їх блокування, відновлення та скасування процесів. В ході процесів можливі обміни даними щодо певних протоколів, результатів яких можна досягнути. Прийняття рішень починається з визначення простору можливих багатопроцесорних комплексів, що можна розглядати як особливий різновид пам'яті зі своєю дисципліною функціонування та взаємодії елементів. Далі відбувається вибір відповідних конфігурацій та структур простору ітерування процесів, призначених до виконання на окремих процесорах. Потім отримана схема управління процесами наповнюється власне діями, що виконують обчислення.

Зазвичай надають перевагу процедурам без рекурсії, звільненим від ряду складнощів, управлінню обчисленнями та побічним ефектам над загальною пам'яттю багатопроцесорної програми, що допускає в динаміці реконфігурацію багатопроцесорного комплексу.

При синхронному (синхронізуючому) багатопотоковому програмуванні виділяються досить чіткі схеми управління обчисленнями і поділяються на регулярні ділянки та типові моделі програми, зручні для розпаралелювання. Головна відмінність від багатопроцесорного програмування зводиться до абстрагування від реальних конкретних структур комплексів. Потоки формуються ближче до рівня постановки задачі. Зазвичай виділяються фрагменти, вільні від побічних ефектів у пам'яті, і допускається неімперативне управління часом виконання потоків програми з урахуванням ієрархії схеми управління програмою та деяких часових відносин. Прийняття рішень починається з вибору стандартних схем управління, використовується поняття «простір ітерування», яке можна структурувати залежно від розподілу даних та методів їх зберігання, що може впливати на ефективність та дисципліну обробки багаторівневої пам'яті. Управління ітеруванням може використовувати довільні предикати. Схема управління над простором ітерування потоків наповнюється порівняно простими для налагодження фрагментами, можливо налагодженими заздалегідь або програмованими автономно. Для виведення результатів програми виділяються спеціальні засоби формування результатів обчислень, отриманих на рівноправних потоках багатопотокової програми. Таким чином, отримується багатопотокова програма із динамічно змінюваним простором потоків над локальною пам'яттю з можливістю епізодичної синхронізації їх окремих фрагментів.

Асинхронне програмування націлене на граничне вираження незалежних елементів програми, обумовлених природою розв'язуваної задачі, що може бути базою для максимального розпаралелювання за умови подання спеціальних схем організації обчислень, можливо з урахуванням специфіки доступного обладнання. Починається прийняття рішень з вибору загальних схем управління діями та подання умов їх спрацювання. Дії можуть використовувати ту чи іншу дисципліну обробки пам'яті. Підтримується неявна та/або програмована різноманітність дисциплін доступу до пам'яті, включаючи ієрархію неоднорідної пам'яті, та схем обчислень фрагментів, що наповнюють загальну схему програми процедурами чи бібліотечними модулями.

Для високопродуктивного програмування необхідний перехід від окремого прогону програми до врахування перспектив її багаторазового застосування та поліпшення. З'являється можливість використовувати недовантажені потужності багатопроцесорних комплексів виконанням фрагментів програми в розрахунку на майбутні прогони з даними, можливо затребуваними в подальших прецедентах її налагодження та застосування. Приблизно так організують програми, призначені для надшвидкого реагування на небезпечні події, наприклад, при прогнозуванні цунамі. Прийняття рішень починається зі схем і моделей обчислень, можливо, над загальною пам'яттю, але із пріоритетом локальної пам'яті.

Управління обчисленнями враховує особливості багаторазового виконання програми при її налагодженні та застосуванні, включаючи можливість успадкування результатів між сеансами та порівняння вимірних характеристик продуктивності версій програми. Отримується ряд покращених версій програми, вибір однієї з яких може враховувати особливості поточних умов застосування, включаючи конфігурацію багатопроцесорного комплексу та вимоги до вимірних характеристик продуктивності програм.

Навчальне програмування покликане компенсувати недоліки загальноприйнятих освітніх орієнтирів на безпомилковість, єдиність, послідовність, імперативність і однозначність рішень при вирішенні будь-яких завдань. Такі орієнтири ускладнюють вивчання методів програмування взагалі, і паралельних обчислень зокрмеа. Навчальне програмування може давати старт всім необхідним парадигмам паралельних обчислень на базі досвіду застосування та конструювання ігрових програм типу візуалізації роботів. Це достатня причина для мультипарадигмальності навчальних мов програмування, які зазвичай підтримують деякі засоби подання паралельних обчислень.

Довгоживучі мови програмування, як і нові мови програмування, зазвичай мультипарадигмальні. Є підстави робити висновки про те, що успішна практика паралельного програмування вимагає мультипарадигмальної підтримки повного спектра парадигм паралельних обчислень.

Необхідно допускати їх розвиток, поповнення та застосування в міру необхідності з можливістю переходу до чергової парадигми без зміни загальномовної та системної обстановки.

Сучасне практичне програмування використовує поняття «мова програмування» як «вхідна мова або розширена підмножина мов програмування типовою системою програмування, що функціонує на базі певної конфігурації обладнання». Відмінність полягає в тому, що система програмування зазвичай супроводжує реалізацію мови програмування розширюваним комплектом бібліотечних модулів і може не підтримувати окремі складності семантики мови програмування. В результаті відбувається згладжування видимих на практиці різниці між різними мовами програмування. Крім того, прямі вимірювання трудомісткості програмування та продуктивності програм майже не відображають залежності результату від прийнятих програмістом рішень і вибору конструкцій мов програмування. Хоча програмовані рішення надаються в термінах мов програмування, їх вплив незначний у дуже складному комплексі, що успадковує продуктивність системи програмування та обладнання з великим домінуванням характеристик елементної

бази та апаратури. Таким чином, існує проблема створення методики, що дозволяє виявляти такі залежності суміщенням прямих вимірювань з результатами експертних оцінок особливостей системи програмування, можливо, від оцінок мови програмування.

Успіх функціонального програмування на практиці істотно залежить від вибору постановок задач, розв'язання яких надаються системами функцій, порівняно простих, не надто трудомістких та зручних при налагодженні [2, 3]. За ступенем вивченості істотно розрізняються такі категорії постановок завдань, що впливають на вибір методів вирішення задач та трудомісткість їх програмування:

- ✓ нові;
- ✓ дослідні;
- ✓ практичні;
- ✓ точні.

Нові постановки задач характеризуються відсутністю доступного прецеденту практичного розв'язання задачі, новизною використовуваних засобів або недоліком досвіду виконавців. Завдання паралельних обчислень поставлені ще в докомп'ютерну епоху і постановки багатьох таких завдань мають математичну точність [5]. Проте, частину завдань паралельного програмування та їх рішень доводиться розглядати як нові через стрімкий розвиток інформаційних технологій, а також елементної бази. Тобто, теоретично, будь-яка проблема, яка не отримала адекватного та вірного рішення, залишається в статусі нового завдання незалежно від часу її постановки.

В сучасних реаліях відмічаються тенденції функціонального підходу та вивчення схем структурування простору ітерування процесів, що дозволяє оптимізувати обробку пам'яті. Функціональний підхід можна розглядати як методику зведення рішень складних завдань до композицій із планарних проекцій, досить зручних для розуміння, аналізу та обробки.

Практичні постановки математичних завдань паралельних обчислень, націлені на актуальність і зручність застосування, переважно використовують потужність доступних інформаційних технологій для відтворення математичних моделей, раніше обмежених низькою ефективністю обладнання, а тепер отримали перспективу стати новою інформаційною революцією.

Точні постановки більшості завдань паралельного програмування склалися на базі послідовних алгоритмів і включають випробування можливостей використовуваних засобів, пов'язаних з мірою організованості раніше створеної імперативної програми. Деякі складнощі пов'язані з використанням однопроцесорних конфігурацій як вихідної моделі паралельних багатопотокових програм. Не виключено, що для граничного випадку зручніше розглядати двопроцесорні конфігурації. Поява власне паралельних алгоритмів зустрічається не так часто.

Проблемою є перехід від теорії паралельних обчислень до практики реалізації високопродуктивних програм, що задовольняють особливо складним критеріям надійності та безпеки. Різниця у вимогах, націлених на розуміння сенсу розв'язуваних завдань, розвиток галузі застосування рішень, визначення межі застосування отриманих результатів та досягнення системності при узагальненні створених інструментальних засобів, важливих для вирішення сучасних ІТ- завдань.

Характерною рисою системного підходу як провідного методу програмування є перехід до класів задач при змістовному аналізі постановок задач. Межі класу встановлюються вибором процесу розв'язання задач. Перехід до експериментів на суперкомп'ютерах показав, що саме системні рішення можуть дати вагомий внесок у продуктивність паралельних обчислень, причому такий вклад може перевищувати теоретичні прогнози [1]. Це можна розглядати як обґрунтування необхідності більш фундаментального підходу до програмування, особливо до системного програмування та його математичних основ, що дає шлях до створення високопродуктивних програм. При створенні, формуванні та дослідженні математичних моделей як фундаментальної бази для вирішення особливо важких проблем ефективності, надійності та безпеки програмного забезпечення важливу роль відіграє розвиток моделей, пов'язаних з часом і ресурсами[1].

Екстенсивний розвиток ІТ помітно випереджає здатності людини оперативно освоювати нові можливості апаратури та системних засобів ІТ, що виходять за межі рівня користувача, що підтримується постачальниками інструментарію та програмних продуктів. Місія системного програмування – створення інструментарію, що дозволяє підвищувати якість інформаційних систем, включаючи пошук нових рішень щодо забезпечення надійності та безпеки інформаційних технологій [1, 3].

Зазвичай функціональне програмування володіє підтримкою низки семантичних та прагматичних принципів, що сприяють створенню функціональних моделей на етапі дослідницького комп'ютерного експерименту, а також природного при вирішенні нових та складних завдань. Серед них функціональне програмування підтримує зовнішні семантичні принципи представлення алгоритмів, такі, як універсальність, саме застосування, рівноправність параметрів, і системно-реалізаційні внутрішні прагматичні принципи підтримки інформаційної обробки, такі як гнучкість обмежень, незмінність даних, строгість результату. Семантичним принципам програміст слід під час підготовки програми. Прагматичні принципи забезпечує система програмування, звільняючи програміста від технічно складних

непринципових рішень, які залежать від природи завдання. Багато з цих принципів були закладені Дж. Маккарті у перших реалізаціях мови Lisp [1].

До таких принципів відносять: універсальність чи загальність, рівноправність, самозастосованість. Поняття «функція» та «значення» володіють тими самими засобами, як і будь-які дані для комп'ютерної обробки. Історично близьке поняття – «принцип програми, що зберігається».

Цей принцип дозволяє будувати уявлення функцій з їх частин і обчислювати частини в міру надходження та обробки даних. Немає принципових обмежень на маніпулювання засобами мови, функціями визначення семантики мови, конструкціями реалізації мови в системі програмування та виразами в програмі. Все, що знадобилося при реалізації мови програмування, може стати в нагоді при його застосуванні. Оскільки не існує перешкод для обробки уявлень функцій тими ж засобами, якими обробляються дані, остільки уявлення функцій можна будувати з їх частин – знаків. Їх навіть можна формувати по ходу процесу обчислень, надходження та обробки інформації про них.

Принцип самозастосованості полягає у поданні функцій прямо або опосередковано, що можуть використовувати самі себе. Це дозволяє будувати ясні лаконічні рекурсивні символьні форми.

Приклади самозастосовності дають багато математичних функцій, особливо рекурсивні, такі, як факторіал, числа Фібоначчі, підсумовування рядів та багато інших, визначення яких використовує математичну індукцію.

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

Отже, стаття присвячена результатам аналізу сучасних тенденцій функціонального програмування, що розглядається як метапарадигма рішення проблем організації паралельних обчислень та багатопотокових програм для багатопроцесорних комплексів та розподілених систем. Враховуючи мультипарадигмальність мов паралельного програмування, тут використано парадигмальний аналіз, що дозволив виявити низку особливостей та прогнозувати хід процесів застосування програм, а також їх вивчення та розробки. Є підстави розраховувати, що функціональне програмування здатне підвищувати продуктивність програм за допомогою попередньої підготовки їхніх прототипів.

Наведено опис семантичних (універсальність, самозастосовність, рівноправність) параметрів) і прагматичних (гнучкість обмежень, незмінність даних, строгість результату) принципів функціонального програмування, а також наслідків із цих принципів, націлених на зниження трудомісткості та підвищення технологічності створення прототипів, модулів, що автономно розвиваються, і налагодження програм (конструктивність, верифікація, факторизація, безперервність поцесів, оборотність дій, унарні функції). Відзначено роль парадигмальної декомпозиції програм у технології розробки довготривалих програм, якими нерідко бувають програми паралельних обчислень. Особливо підкреслено перспективу функціонального програмування як допоміжної універсальної метапарадигми вирішення складних і важливих завдань, обтяжених важко засвідченими та слабо сумісними вимогами ефективності, надійності, безпеки та покращення. Показано різноманітність парадигмальних характеристик, властивих підготовці та налагодженні довготривалих програм паралельних обчислень.

Література

1. Pieter Koopman, Steffen Michels, and Rinus Plasmeijer Dynamic Editors for Well-Typed Expressions. Trends in Functional programming/ 22nd internationa Symposium, TFP 2021, February 17-19, 2021/ Springer, LNCS 12834, pp. 44-66.

2. Weizenbaum J. ELIZA – A Computer Program For the Study of Natural Language Communication Between Man And Machine / J. Weizenbaum // Communications of the ACM. – 1966. – Volume 9(1). – P. 36–45.

3. Schwartz Jacob T. "Set Theory as a Language for Program Specification and Programming". Courant Institute of Mathematical Sciences, New York University, 1970.

References

1. Pieter Koopman, Steffen Michels, and Rinus Plasmeijer Dynamic Editors for Well-Typed Expressions. Trends in Functional programming/ 22nd internationa Symposium, TFP 2021, February 17-19, 2021/ Springer, LNCS 12834, pp. 44-66.

2. Weizenbaum J. ELIZA – A Computer Program For the Study of Natural Language Communication Between Man And Machine / J. Weizenbaum // Communications of the ACM. – 1966. – Volume 9(1). – P. 36–45.

3. Schwartz Jacob T. "Set Theory as a Language for Program Specification and Programming". Courant Institute of Mathematical Sciences, New York University, 1970.