МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

Факультет комп'ютерно-інформаційних технологій та автоматизації
Кафедра прикладної математики і інформатики

«До захисту допущено»
Завідувачка кафедри ПМІ
_____ Наталія МАСЛОВА
«___» _____ 2023р.

Випускна кваліфікаційна робота
магістра
(освітній ступень)

на тему: «Дослідження ефективності паралельних алгоритмів чисельного розв'язання жорстких задач Коші»

Виконав: студент 2 курсу, групи ІПЗм-22

Спеціальності 121 Інженерія програмного забезпечення

БЕРВІН М.С. _____
(прізвище та ініціали) (підпис)

Керівник доц. каф. ПМІ, к.т.н., доцент Ірина НАЗАРОВА _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Рецензент доц. каф. ВМ, к.ф.-м.н., доцент Наталія ГОГОЛЄВА _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Рецензент зав. каф. КІ, к.т.н., доцент Сергій КОВАЛЬОВ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій випускній кваліфікаційній роботі немає запозичень з праць інших авторів без відповідних посилань

Студент _____
(підпис)

Луцьк – 2023 р.

ДВНЗ «Донецький національний технічний університет»
Факультет комп'ютерно-інформаційних технологій та автоматизації_____
Кафедра прикладної математики та інформатики_____
Освітньо-кваліфікаційний рівень (освітній ступінь) магістр_____
Напрям підготовки (спеціальність) 121 Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Завідувачка кафедри ПМІ
Наталія МАСЛОВА
/_____/
"____" _____2023 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТУ

Бервіну Михайлу Сергійовичу

1. Тема проєкту (роботи) Дослідження ефективності паралельних алгоритмів чисельного розв'язання жорстких задач Коші
Керівник проєкту (роботи) Ірина НАЗАРОВА, к.т.н, доцент, доцент каф. ПМІ
(ім'я, прізвище, науковий ступінь, вчене звання, посада)
затверджені наказом від "13" листопада 2023 року № 562
2. Строк подання студентом проєкту (роботи)_____
3. Вихідні дані до проєкту (роботи) результати науково-дослідної роботи, матеріали переддипломної практики.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1) огляд предметної області;
2) аналіз технологій та методів;
3) розробка алгоритму вирішення поставлених завдань;
4) реалізація програмного продукту.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
робота містить рисунки, в кількості, достатній для демонстрації матеріалів випускної кваліфікаційної роботи

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Назарова І.А. доц. каф. ПМІ		

7. Дата видачі завдання 04.09.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Об'єктний аналіз предметної області і постановка завдання	10.09.2023	
2.	Огляд засобів для розробки	14.09.2023	
3.	Розробка прототипу програми	20.09.2023	
4.	Проєктування системи	10.10.2023	
5.	Реалізація механізму збереження розрахунків	15.10.2023	
6.	Проведення експериментів	18.10.2023	
7.	Збір та аналіз результатів	22.11.2023	
8.	Оформлення пояснювальної записки	10.12.2023	
9.	Підготовка слайдів презентації	15.12.2023	
10.	Підготовка демонстраційної версії розробленого програмного продукту	16.12.2023	
11.	Написання доповіді	20.12.2023	
12	Захист	26.12.2023	

Студент _____ Михайло БЕРВІН
(підпис) (ім’я, прізвище)

Керівник роботи _____ Ірина НАЗАРОВА
(підпис) (ім’я, прізвище)

АНОТАЦІЯ

Михайло БЕРВІН. Дослідження ефективності паралельних алгоритмів чисельного розв’язання жорстких задач Коші / Випускна кваліфікаційна робота на здобуття освітньо-кваліфікаційного рівня «магістр» за спеціальністю 121 Інженерія програмного забезпечення ДВНЗ ДонНТУ, Покровськ-Луцьк, 2023.

Об'єкт дослідження: ефективність паралельних методів розв’язання жорстких задач Коші.

Предмет дослідження: розробка та створення програмного інструменту для аналізу ефективності паралельних методів розв’язання жорстких задач Коші.

Мета роботи: дослідження та використання паралельних методів розв’язання жорстких задач Коші, порівняння їх ефективності.

Практичне значення: підвищенні ефективності використання паралельних обчислювальних систем при розв’язанні жорсткої задачі Коші.

Ключові слова: паралельні методи, чисельні методи, жорстка задача Коші, блокові методи, ефективність.

Список публікацій здобувача:

Бервін М.С. Дослідження ефективності паралельних методів розв’язання жорстких задач Коші / М.С. Бервін. – «ТАК»: телекомунікації, автоматика, комп’ютерно-інтегровані технології: зб. доповідей Всеукр. наук.-практ. конф. молодих вчених, Грудень 2023 р. // ДВНЗ «ДонНТУ; відп. ред. Г.В. Ступак. – Луцьк: ДВНЗ «ДонНТУ», 2022. – С. 4-6 [1].

ANNOTATION

Mikhailo BERVIN. Research on the efficiency of parallel algorithms for the numerical solution of hard Cauchy problems / Graduate qualification work for master's degree in 121 Software Engineering of DonNTU, Pokrovsk-Lutsk, 2023.

Object of research: efficiency of parallel methods for solving hard Cauchy problems.

Subject of research: development and creation of a software tool for analysing the efficiency of parallel methods for solving hard Cauchy problems.

Purpose: to study and use parallel methods for solving hard Cauchy problems, to compare their efficiency.

Practical significance: to increase the efficiency of using parallel computing systems in solving hard Cauchy problems.

Keywords: parallel methods, numerical methods, hard Cauchy problem, block methods, efficiency

List of applicant`s publications:

Bervin M. Research on the efficiency of parallel methods for the numerical solution of hard Cauchy problems / M. Bervin. // TAC: telecommunications, automation, computer-integrated technologies: a collection of reports of the All-Ukrainian scientific and practical conference of young scientists, December, 2023 / SHEI "DonNTU; editor-in-chief G.V. Stupak: SHEI "DonNTU", 2023. – P. 4-6 [1].

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

SIMD – (single instruction multiple data) одиночний потік команд, множинний потік даних

MIMD – (multiple instruction multiple data) множинний потік команд, множинний потік даних

SISD – (single instruction single data) одиночний потік команд і даних

MISD – (multiple instruction single data) множинний потік команд, одиночний потік даних

ЕОМ – електронно обчислювальна машина

СЗДР – система звичайних диференційних рівнянь

ДР – диференційне рівняння

ПНМРК – повністю неявний метод Рунге-Кути

ПЕ – процесорні елементи

UML – (Unified Modeling Language) уніфікована мова моделювання

ВСТУП

Застосування високопродуктивних паралельних обчислювальних систем (ОС) є одним з основних напрямків розвитку сучасної комп'ютерної науки. Ця обставина викликана не тільки принциповим обмеженням максимально можливої швидкодії звичайних послідовних машин, а й постійною появою нових обчислювальних задач, для розв’язання яких можливостей існуючих засобів обчислювальної техніки завжди виявляється недостатньо. Моделювання реальних багатовимірних динамічних процесів, що описуються системами звичайних диференціальних рівнянь (СЗДР), і являє собою клас задач, при реалізації яких використання паралельних суперкомп'ютерів не лише виправдане, але й необхідне. Про це свідчить відомий список "великий виклик", в якому такі задачі займають одне з провідних місць [2].

В останні роки істотно зросла пікова продуктивність паралельних обчислювальних систем, радикально змінилася технологічна база та архітектура. Але, як і раніше, головною перешкодою до впровадження практично всіх паралельних архітектур є відсутність ефективних прикладних паралельних методів. Як підсумок, проблема підвищення ефективності функціонування паралельних обчислювальних систем, незважаючи на наявні успіхи, далека до повного вирішення, як в теоретичному, так і в практичному відношенні.

Найбільш складною є задача визначення оптимальної алгоритмічної та структурної організації паралельних обчислень с метою досягнення достатніх характеристик ефективності. На сьогодні не викликає сумнівів, що реалізувати потенційні можливості паралельних комп'ютерів можливо перш за все на основі проведення цілеспрямованої теоретичної роботи з побудови нових та адаптації існуючих методів розв’язання складних науково-технічних завдань.

Перераховані вище проблеми визначили спрямованість дипломної роботи, що присвячена розробці та обґрунтуванню паралельних методів розв’язання широкого класу науково-технічних і комплексних стратегічних динамічних задач із зосередженими параметрами.

Мета роботи полягає у підвищенні ефективності багатопроцесорних обчислювальних систем за рахунок розробки паралельних методів вирішення жорстких динамічних завдань із зосередженими параметрами та їхньої топологічної реалізації у сучасних обчислювальних структурах.

Об'єктом дослідження є високопродуктивні паралельні обчислювальні системи довільної архітектури з розподіленою пам'яттю та різними базовими топологічними характеристиками.

Предметом дослідження є паралельні методи вирішення динамічних завдань із зосередженими параметрами для систем звичайних диференціальних рівнянь, які забезпечують підвищення ефективності багатопроцесорних обчислювальних систем.

Методи досліджень. Розробка та обґрунтування паралельних методів вирішення початкових завдань, реалізація альтернативних методів оцінки локальної апостеріорної похибки базувалася на використанні теорії звичайних диференціальних рівнянь та методів обчислювальної математики. Розпаралелювання послідовних методів проводилося на основі теорії графів. При дослідженні властивостей розроблених паралельних методів та їх відображення на паралельні структури застосовувалися методи теорії алгоритмів та паралельних обчислень. Обґрунтування запропонованих паралельних методів вирішення динамічних завдань підтверджується шляхом проведення обчислювальних експериментів та результатами емуляції бібліотеки обміну повідомленнями MPI (Arg MPICH-1.2.5) для мови C++ [3].

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Сучасні класи методів розв’язання систем звичайних диференційних рівнянь

Однією з найбільш поширених задач математичного моделювання, є моделювання динамічних процесів, що описуються системами звичайних диференційних рівнянь (СЗДР), а саме пошук чисельного розв'язання задачі Коші для СЗДР. У зв'язку з цим, є актуальними завдання, пов'язані з розробкою нових паралельних чисельних методів і паралельної реалізацією існуючих.

Задача Коші для одного звичайного диференційного рівняння першого порядку має вигляд:

$$\begin{cases} y' = f(x, y) \\ y(x_0) = y_0 \end{cases}, \tag{1.1}$$

де $x -$ незалежна змінна;

$y(x) -$ невідома, шукана функція;

$f(x, y) -$ права частина диференційного рівняння

$y(x_0) = y_0 -$ початкова умова / точка.

Функція $y(x)$ є рішенням цього рівняння, якщо для всіх x виконується наступне: $\frac{dy(x)}{dx} = f\big(x, y(x)\big)$. Зазвичай рішення містить у собі незалежний параметр, тому визначається єдиним чином лише тоді, коли є задане початкове значення: $y(x_0) = y_0$. При цьому задача Коші полягає в такому, щоб знайти функцію $y = y(x)$ на заданому відрізку $[a, b]$, що задовольняє рівнянню (1.1) і початковій умові.

Чисельні методи розв'язання задачі Коші знаходять розв'язок (тобто функцію $y(x)$ на відрізку $[a, b]$) у табличному вигляді, а саме у вигляді набору точок (x_i, y_i), $i = 1, .., n$, де $x_0 = a,\ x_i = x + ih$, $i = 1, .., n$, $h = \frac{b-a}{n}$, n – задане

число розбиття відрізка [a, b], h – крок інтегрування, а y_i, $i = 1,..,n$ – знайдені наближені значення функції $y(x)$ в точках x.

Задача Коші для системи з m звичайних диференційних рівнянь (СЗДР) першого порядку, що розв'язана відносно похідних (нормальна форма):

$$\begin{cases} y'_1 = f_1(x, y_1, y_2, \dots, y_m), & y_1(x_0) = y_{01} \\ y'_2 = f_2(x, y_1, y_2, \dots, y_m), & y_2(x_0) = y_{02} \\ \dots \\ y'_m = f_m(x, y_1, y_2, \dots, y_m), & y_m(x_0) = y_{0m} \end{cases}. \quad (1.2)$$

Векторний запис задачі Коші для СЗДР:

$$\begin{cases} \bar{y}' = \bar{f}(x, \bar{y}) \\ \bar{y}(x_0) = \bar{y}_0 \end{cases} \text{ або } \begin{cases} Y' = F(x, Y) \\ Y(x_0) = Y_0 \end{cases}, \quad (1.3)$$

де $Y = \bar{y} = (y_1, y_2, \dots, y_m)^T$, $\bar{y} = F: R \to R^m$ – вектор невідомих;

$Y_0 = \bar{y}_0 = (y_{10}, y_{20}, \dots, y_{m0})^T$ – вектор початкових значень;

$F = \bar{f} = (f_1, f_2, \dots, f_m)^T$ – права частина СЗДР, в загальному випадку нелінійна функція.

Більшість чисельних методів розв'язання задачі Коші можна представити у вигляді:

1) для рівняння: $y_{n+1} = \Phi\left(y_{n-q}, y_{n-q+1}, \dots, y_n, y_{n+1}, \dots, y_{n+s}\right)$,

2) для системи: $\bar{y}_{n+1} = \Phi(\bar{y}_{n-q}, \bar{y}_{n-q+1}, \dots, \bar{y}_n, \bar{y}_{n+1}, \dots, \bar{y}_{n+s})$,

де Ф – деяка функція від вказаних аргументів, що визначається обраним методом, видом рівняння та побудованою сіткою.

При $q = 0$, $0 \le s \le 1$ чисельний метод називають однокроковим: $y_{n+1}=\Phi(y_n, y_{n+1}, \dots, y_{n+s})$. Однокрокові методи називають явними при $s = 0$, неявними при $s = 1$. Іншими словами, метод називається однокроковим, якщо обчислення рішення в наступній точці проводиться з використанням тільки одного попереднього значення. При $q \ge 1$ або $s \ge 1$ чисельний метод називають багатокроковим.

Багатокрокові чисельні методи при $s > 1$ називають з запобіганням вперед. На рис. 1.1 приведена неповна класифікація найбільш відомих чисельних однокрокових, багатокрокових та з вбудованими

способами оцінки локальної апостеріорної похибки для вибору кроку інтегрування при розв’язанні диференційних рівнянь, що має особливе значення у разі жорстких задач Коші [4-9].

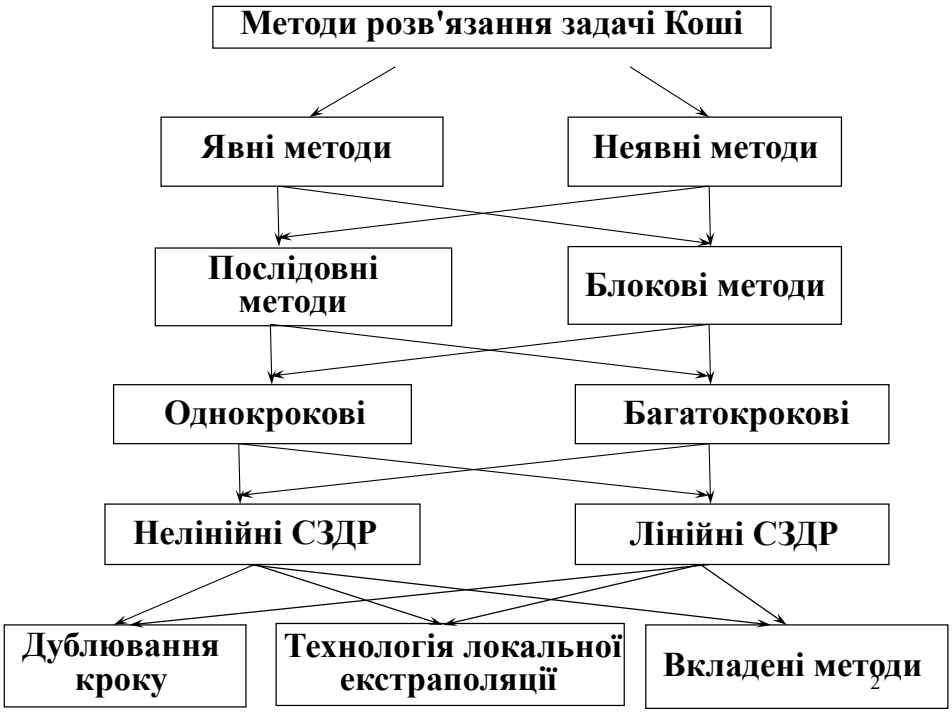


Рисунок 1.1 – Класифікація чисельних методів розв’язання СЗДР

1.2 Поняття жорсткої задачі Коші та методи керування кроком інтегрування

До основних недоліків однокрокових та багатокрокових методів відносять те, що для розрахунку наступного кроку необхідне значення попереднього або попередніх кроків і таке, щоб забезпечувало припустиму помилку на кроці. В таких методах, якщо помилка на поточному кроці неприпустима, то зменшують крок інтегрування і обчислюють значення заново, що веде до значного зростання використаного машинного часу. Наприклад, модифікований метод Ейлера має похибку порядку h^2, де $h = x_k - x_{k+1}$, що є кроком інтегрування. Але на практиці, як правило, застосовують «подвійний прорахунок», де спочатку чисельне розв‘язання

рівняння ведеться з кроком h, потім крок дроблять і повторний розрахунок ведеться з кроком $h/2$.

Багатокрокові методи схожі на однокрокові, але для розрахунку наступного кроку використовуються декілька попередніх прорахованих значень функції з меншим порядком точності, що дозволяють завдяки допоміжному рішенню $\tilde{y}_{n+1}$ оцінювати локальну похибку та керувати кроком інтегрування. Наприклад, метод Рунге-Кутти дозволяє підвищити точність за рахунок використання четвертого порядку точності, а саме, обмежитись членами до h^4 включно. Саме тому такі ітерації, де розрахунок стадійних коефіцієнтів знаходиться у прямій залежності від попередньої стадії, виконуються послідовно. Існують розробки та модифікації методу Рунге-Кутти в яких розрахунок окремих компонентів стадійних коефіцієнтів та значень $\bar{y}_{n+1}$ та $\bar{\bar{y}}_{n+1}$ обчислюють паралельно. Та все ж подібні методи потребують значної кількості операцій обміну між паралельно працюючими обчислювальними вузлами.

Також важливо враховувати жорсткість поставленої задачі. Жорсткі задачі мають розповсюджений характер, та зустрічаються майже в усіх областях, де використовується математичне комп'ютерне моделювання. Жорсткість в деяких джерелах [4-5] описуються так: «Жорсткі рівняння – це рівняння, для яких певні неявні методи дають незрівнянно кращий результат, ніж явні методи». Більше інформації щодо суджень про жорсткість описані в багатьох джерелах [4-9].

Саме з причин жорсткості та великої розмірності систем ЗДР явні однокрокові та багатокрокові методи не підходять для вирішення подібних систем ще тому, що виникає необхідність ефективного розміщення даних, що розподіляються, на процесорну решітку у випадку, коли виконується паралельне обчислення системи.

Одна з перших спроб дати визначення жорстким системам належить Ч. Кертісс і Дж. Хіршфельдер (C.F. Curtiss, J.O. Hirschfelder). У літературі наводиться кілька суджень про жорсткість, кожне з яких відображає певні

аспекти поведінки чисельного рішення (наприклад, неможливість використання явних методів інтегрування, наявність швидко згасаючих збурень, великі постійні Липшиця або логарифмічні норми матриць, велика різниця власних значень матриці Якобі, заповненість матриці Якобі, апріорна знаковизначеність рішення, кількість перехідних фаз і так далі) [4-5].

У роботах [8-9] авторам важко дати суворе визначення жорсткої системи з його складності, а вводять робочий опис жорсткої завдання. Це завдання, що моделює фізичний процес, компоненти якого мають непорівнянні характерні часи, або процес, характерний час (зворотні величини власних значень якобіана системи) якого набагато менше відрізка інтегрування.

Методи, які відносяться до однокрокових мають певні загальні риси. В основі усіх однокрокових методів лежить розклад функції в ряд Тейлора, в якому зберігаються члени, що мають h в степені до k включно. Ціле число k називається порядком метода. Похибка на кроці має порядок $k+1$. Також всі одноточкові методи не потребують дійсного обчислення похідних, тому що обчислюється лише сама функція, однак можуть потребуватися її значення в деяких проміжних точках. Це тягне за собою, звичайно, додаткові затрати часу і зусиль. Характерною рисою для однокрокових методів є те, що для отримання інформації у новій точці, потрібно мати дані лише в попередній точці [4-7].

Найбільш простим однокроковим методом, який потребує мінімальних затрат обчислювальних ресурсів, але дає змогу обчислювати результат із порівняно низькою точністю, є метод Ейлера.

В цьому методі для оцінювання наступної точки на кривій $Y = F(x_0)$ використовується лише один лінійний член в формулі Тейлора,

$$Y(x_0 + h) = Y(x_0) + hY'(x_0) \tag{1.4}$$

де $Y'(x_0)$ визначається з початкового рівняння.

Цей процес можна розповсюдити на наступні кроки:

$$Y_{n+1} = Y_n + hF(x_n, Y_n), \tag{1.5}$$

Метод Ейлера є методом першого порядку (p =1)

$$\Delta \leq ch^2, \tag{1.6}$$

де $c = (M_1 + M_0 M_2)/2,\ M_0, M_1, M_2$ – визначаються як

$$\begin{aligned} M_0 &\geq |F(t,Y)|, \\ M_1 &\geq \left|\frac{\partial F(t,Y)}{\partial x}\right|, \\ M_2 &\geq \left|\frac{\partial F(x,Y)}{\partial Y}\right|, \end{aligned} \tag{1.7}$$

для всіх $x \in [a,b]$ і $Y = Y(x)$.

Метод Ейлера є порівняно грубим і застосовується в основному для орієнтованих розрахунків. Однак ідеї, покладені в основу методу Ейлера, є базовими для інших методів. Цей метод можна вдосконалити різними способами. Найбільш відомі два з них: виправлений метод Ейлера і модифікований метод Ейлера. Метод Ейлера і його модифікації ще називають методами Рунге-Кутти першого і другого порядку.

Використовуючи метод Рунге-Кутти четвертого порядку точності, на кожному кроці доводиться обчислювати чотири значення функції, але для збіжності методу прогнозу і корекції того ж порядку точності часто достатньо двох значень функції. Тому методи прогнозу і корекції вимагають майже вдвічі менше машинного часу, ніж методи Рунге-Кутти порівнюваної точності.

Одноточкові методи і методи прогнозу і корекції забезпечують приблизно однакову точність результатів. Однак другі на відміну від перших дозволяють лише оцінити похибку на кроці. З цієї причини, користуючись одноточковими методами, величину кроку h звичайно обирають трохи менше, ніж це необхідно, тому методи прогнозу і корекції виявляються найбільш ефективними [5].

В методах прогнозу і корекції (багатоточкові методи) для обчислення значення нової точки розв‘язку ДР використовується інформація про декілька раніше отриманих точок відрізку дослідження. Для цього використовуються дві формули, що називаються відповідно формулами прогнозу і корекції.

Схеми алгоритмів для всіх таких методів майже однакові, а самі методи відрізняються лише формулами.

Так як в методах, що розглядаються використовується інформація про декілька раніше отриманих точок, то на відміну від однокрокових методів вони не володіють властивістю «самостартування». Тому, перед тим як застосовувати метод прогнозу і корекції, необхідно обчислювати вихідні данні за допомогою якого-небудь однокрокового методу. Часто для цього використовують метод Рунге-Кутти. Обчислення проводять наступним чином. Спочатку по формулі прогнозу і початковим значенням змінних знаходять значення $Y_{n+1}^{(0)}$. Верхній індекс (0) означає, що прогнозоване значення є одним з послідовності значень Y_{n+1}, що розташовані в порядку зростання точності. По прогнозованому значенню $Y_{n+1}^{(0)}$ за допомогою приведеного вище диференціального рівняння знаходять похідну

$$Y_{n+1}^{(0)\prime} = F\left(x_{n+1}, Y_{n+1}^{(0)}\right), \tag{1.8}$$

яка потім підставляється у формулу корекції для обчислення уточненого значення $Y_{n+1}^{(j+1)}$.

В свою чергу $Y_{n+1}^{(j+1)}$ використовується для отримання більш точного значення похідної за допомогою диференціального рівняння

$$y_{n+1}^{(j+1)\prime} = F\left(x_{n+1}, Y_{n+1}^{(j+1)}\right) \tag{1.9}$$

Якщо це значення похідної недостатньо близьке до попереднього, то воно вводиться у формулу корекції й ітераційний процес продовжується. Якщо ж похідна змінюється в допустимих границях, то значення $Y_{n+1}^{(j+1)\prime}$ використовується для обчислення остаточного значення Y_{n+1} , яке і видається на друк. Після цього процес повторюється – здійснюється наступний крок, на якому обчислюється Y_{n+2}. Якщо диференціальне рівняння $Y'(x) = F(x, Y)$ проінтегровано в інтервалі значень від t_0 до t_{n+k}, то результат прийме вигляд

$$Y(x_{n+k}) - Y(x_n) = \int_{x_n}^{x_{n+k}} F(x, Y)\partial t. \tag{1.10}$$

Цей інтеграл не можна обчислити безпосередньо, так як залежність *Y(x)* заздалегідь невідома. Наближене значення інтегралу можна знайти за допомогою одного з кінцево-різницевих методів. Вибір методу і буде визначати метод розв'язку диференціальних рівнянь. На етапі прогнозу можна використовувати будь-яку формулу чисельного інтегрування, якщо до неї не входить попереднє значення $Y'(x_{n+1})$ [10].

1.3 Аналіз сучасних архітектур багатопроцесорних паралельних обчислювальних систем

Сучасний етап розвитку обчислювальної техніки характеризується застосуванням величезної кількості високопродуктивних паралельних комп'ютерів. В них використовуються різні апаратні складові: процесори (Intel, IBM, AMD, HP, NEC, Cray), мережеві карти (Ethernet, Myrinet, Infiniband, SCI). Вони функціонують під управлінням різних операційних систем (Unix , Linux, Windows) та реалізують різне прикладне програмне забезпечення. Велика розмаїтість паралельних систем спричинила необхідність введення для них ефективної системи класифікації, що представляє можливість диференціації істотно різних обчислювальних систем, а також володіє простотою і практичною доцільністю.

Однією з перших і найбільш поширених схем класифікації архітектур обчислювальних систем є систематика М. Флінна [10]. В її основу покладено опис роботи комп'ютера з потоками виконуваних команд і оброблюваних даних (рис. 1.2).

У результаті такого підходу розрізняють наступні основні типи паралельних систем:

1) SISD (Single Instruction, Single Data) – поодинокий потік команд і поодинокий потік даних;

2) SIMD (Single Instruction, Multiple Data) – поодинокий потік команд і множинний потік даних;

3) MISD (Multiple Instruction, Single Data) – множинний потік команд і поодинокий потік даних;

4) MIMD (Multiple Instruction, Multiple Data) – множинний потік команд і множинний потік даних.

Загальна класифікація архітектури обчислювальних систем за ознаками наявності паралелізму в потоках команд і даних була запропонована М. Флінном (рис. 1.2).

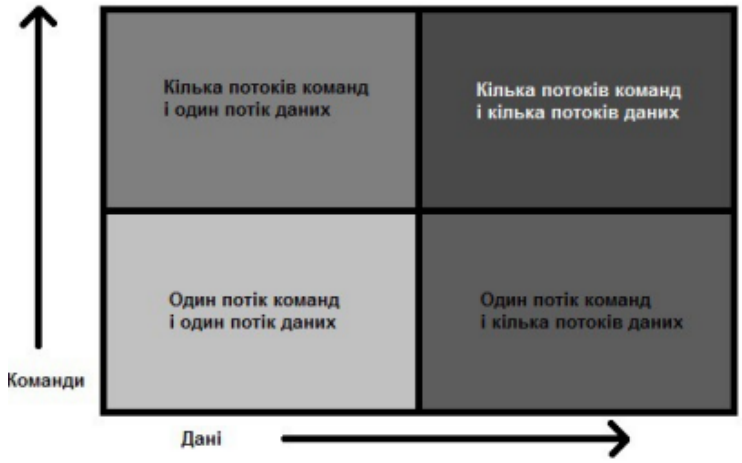


Рисунок 1.2 – Класифікація паралельних архітектури по Флінну

Систематика Флінна широко використовується при конкретизації типів комп’ютерних систем, але вона призводить до того, що практично всі типи паралельних систем, незважаючи на їх істотну різнорідність, виявляються віднесеними до однієї групи MIMD [11]. В результаті вживаються спроби деталізації систематики Флінна. Найбільшого визнання здобули класифікації Шора, Кришнамарфі, Шнайдера, Хокні [6-7].

Паралельні обчислювальні системи мають у своєму складі поле процесорних елементів, а також один або декілька модулів пам'яті. Ці функціональні компоненти повинні бути пов'язані через відповідні комутаційні мережі. Всі високорівневі концепції комунікацій, такі, як загальна пам'ять: реальна чи віртуальна або обмін повідомленнями, реалізуються в паралельних системах деякими фізичними структурами.

У силу специфіки задач, що вирішуються на паралельних ОС [6-7,12], структура ліній зв'язку (топологія мережі передачі даних) повинна задовольняти різним вимогам. По-перше, забезпечувати зв'язок між двома будь-якими процесорами або модулями. Крім цього, підтримувати максимальну кількість одночасних зв'язків, щоб кошти комутації не обмежували паралельну обробку інформації. До числа типових топологій за типом «точка-точка» зазвичай відносять наступні схеми: лінійка/кільце, сітка/2D-тор, гіперкуб, кліка або повний граф.

Лінійка представляє собою лінійний масив процесорів. Кожен процесор з'єднаний з двома сусідніми (за винятком кінцевих процесорів, що мають по одному з'єднанню). Топологія кільце виходить з лінійки шляхом з'єднання першого і останнього процесорів [6-7]. Кільцева структура зберігає переваги лінійки та скорочує максимальну відстань між процесорами – $p/2$.

Сітка (2D-сітка, матриця – *mesh*) являє собою систему, в якій граф ліній зв’язку утворює прямокутну сітку, тобто процесорні елементи розташовані у вигляді правильної двовимірної таблиці та кожен процесор (крім крайніх) з'єднаний із чотирма сусідами.

Перевагою схеми є простота, а недолік полягає в тому, що при обміні між віддаленими процесорами дані повинні пройти через ряд проміжних процесорів. Всі квадратні сітки мають максимальну відстань між процесорами порядку $O(\sqrt{p})$. Більшість реально побудованих сіткових масивів використовують двовимірні схеми з'єднання. Крім двовимірних є схеми з тривимірним масивом процесорів [12], в яких кожен процесор з'єднаний з шістьма найближчими сусідами. У кубічної, 3D-сітці процесори розміщені в просторі куба, максимальна відстань між процесорними елементами пропорційно кореню кубічному із p: $\sqrt[3]{p}$. Замкнута 2D-сітка або тор з'єднання мають діаметр, що пропорційний $\sqrt{p}$. Квадратний тор з чотирма лініями зв'язків у кожного процесора має діаметр: $\sqrt{p}$.

В існуючих обчислювальних системах однією з найбільш часто використовуваних та ефективних є топологія міжпроцесорних зв'язків – гіперкуб. Структура гіперкуб (*hypercube*) є окремим випадком сітки, коли за кожним розміром сітки є тільки два процесори, тобто гіперкуб містить $p = 2^N$ процесорів при розмірності N. При тривимірній схемі з'єднання процесори розміщені у вершинах тривимірного куба, а ребра куба грають роль локальних зв'язків між процесорами. Кожен процесор з'єднаний з трьома найближчими сусідами. При гіперкубовій архітектурі кількість зв'язків між процесорами невелика: у N-мірному гіперкубі кожний процесор має N сосідів. Невеликим порівняно з кількістю процесорів виявляється й діаметр системи, рівний $log_2 p$. Топологія гіперкуб надає можливість логічно моделювати деякі важливі типи зв'язків: лінійка, кільце, сітка. Недоліком описаної архітектури є існування практичної межі збільшення розміру гіперкуба. Гіперкуб є універсальною і одною, що найбільш ефективних систем зв'язку, її концепції використані в обчислювальних системах Intel iPSC, Ncube Corp. та інших.

Для оцінки часу виконання операції передачі одного повідомлення обсягом V байтів між двома процесами, локалізованими на різних процесорах при розподіленій пам'яті, використовується наступна лінійна модель, запропонована Хокні [6-7,12-14]:

$$T_{p-p} = t_s + t_w \cdot V \cdot l,\ t_w = \frac{y}{B}, \tag{1.11}$$

де t_s – латентність, тривалість підготовки повідомлення для передачі;

l – довжина маршруту;

t_w – час передачі одного байту;

y – кількість байтів у слові;

B – пропускна здатність каналу передачі даних (байт/секунда).

Оскільки паралельні методи розв'язання динамічних задач із зосередженими параметрами використовують структурну інформаційну взаємодію за типом комунікацій, надається можливість розробити єдині

комунікаційні примітиви для різних операцій передачі даних у різних топологіях.

Розглянемо топології, що найбільш поширені у використанні: лінійка/кільце, сітка/тор, гіперкуб та три комунікаційні операції:

1) передача даних між двома процесорами мережі – операція типу "точка-точка" або "point-point";

2) передача даних від одного процесора всім іншим процесорам мережі – операція "один-усім" або "one-to-all";

3) передача даних від усіх процесорів мережі усім процесорам мережі – множинна пересилка "усі-усім" або "all-to-all".

Трудомісткість одиночної операції пересилання даних між двома процесорами може бути отримана шляхом підстановки довжини максимального шляху (діаметра мережі) у вираз (1.1). Для обчислення часу виконання множинної пересилки необхідно вибрати алгоритм маршрутизації. До числа найбільш поширених оптимальних алгоритмів передачі даних відносять методи покоординатної маршрутизації [13-14].

Ідея цих методів полягає в тому, що пошук шляхів передачі даних здійснюється послідовно для кожної розмірності розглянутої топології.

Для кільцевої топології кожен процесор може ініціювати розсилку повідомлення в будь-якому обраному напрямку за кільцем. Без істотних обмежень припускаємо, що будь-який процесор має можливість здійснювати одночасно прийом та передачу даних (двонапрямлені лінки). Таким чином, для топології кільце час виконання поодинокої пересилки даних складає:

$$T_{p-p}^{R} = t_s + V \cdot t_w \cdot \lfloor p/2 \rfloor. \qquad (1.12)$$

Передача даних від одного процесора всім іншим для топології кільце визначається співвідношенням:

$$T_{one-to-all}^{R} = t_s + V \cdot t_w \cdot \lceil p/2 \rceil, \qquad (1.13)$$

а час виконання множинної пересилки "усі-усім" становить:

$$T_{all-to-all}^{R} = (t_s + V \cdot t_w) \cdot (p-1). \qquad (1.14)$$

Для топології сітка-тор (рис. 1.7) поодинока операція пересилки даних вимагає наступного часу для виконання з урахуванням моделі (1.1) та діаметра топології:

$$T_{p-p}^{M} = t_s + 2V \cdot t_w \cdot \left\lfloor \sqrt{p} / 2 \right\rfloor \quad (1.15)$$

Пересилка даних від одного процесора всім іншим в умовах топології тор може бути отримана з цього ж способу передачі для кільцевої топології, виконаного у два етапи:

$$T_{one-to-all}^{M} = t_s + 2V \cdot t_w \cdot \left\lfloor \sqrt{p} / 2 \right\rfloor. \quad (1.16)$$

Множинна розсилка повідомлень також може бути виконана за допомогою алгоритму, одержаного узагальненням способу передачі даних для кільцевої мережі на основі ідеї покоординатної маршрутизації:

$$T_{all-to-all}^{M} = 2t_s(\sqrt{p} - 1) + V \cdot t_w \cdot (p - 1). \quad (1.17)$$

Для топології гіперкуб (рис. 1.8) час виконання одиночної операції пересилки даних дорівнює:

$$T_{p-p}^{H} = t_s + V \cdot t_w \cdot log_2\, p. \quad (1.18)$$

Час виконання операції передачі даних від одного процесора всім іншим у топології гіперкуб становить:

$$T_{one-to-all}^{H} = (t_s + V \cdot t_w) \cdot log_2\, p. \quad (1.19)$$

Алгоритм виконання множинної розсилки повідомлень для гіперкуба може бути отриманий шляхом узагальнення способу передачі даних "усі-усім" для топології сітка на розмірність гіперкуба $lp = log_2\, p$.

Кожному процесору ставиться у відповідність двійковий еквівалент його номера. Процесори, що мають безпосереднє з'єднання у гіперкубі, будуть мати номери, що відрізняються один від одного тільки одним розрядом. На кожному етапі $i, i = \overline{1,lp}$ алгоритму функціонують усі процесори мережі, які обмінюються даними зі своїми сусідами за i-тою розмірністю та формують об'єднані повідомлення:

$$T_{all-to-all}^{H} = \sum_{i=1}^{lp} (t_s + 2^{i-1} \cdot V \cdot t_w) = t_s \cdot log_2\, p + t_w \cdot V \cdot (p - 1) \quad (1.20)$$

Особливістю кластерних систем є використання стандартних мережевих технологій. На даний момент найбільш популярними за рейтингом TOP-50 для СНД є Gigabit Ethernet, InfiniBand, Myrinet (рис. 1.10). Список наведено в порядку зменшення кількості кластерів, що використовують ту або іншу технологію. Аналогічна тенденція спостерігається й у світовому масштабі [11-12, 14-18].

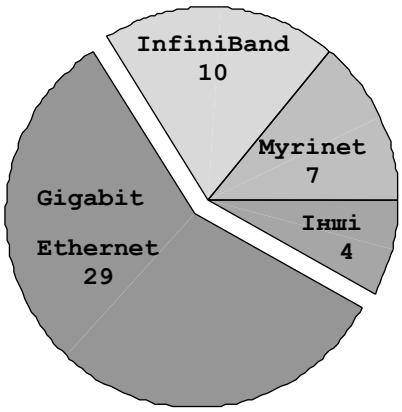


Рисунок 1.10 – Мережеві технології в рейтингу ОС TOP-500

Ефективність використання тієї чи іншої мережевої технології визначається її числовими параметрами. У таблиці 1.1 на підставі матеріалів [11-12] приведені комунікаційні константи обміну для найбільш відомих комунікаційних мереж.

Таблиця 1.1 –Комунікаційні константи для мережевих технологій

Мережева технологія	Латентність (мкс)	Швидкість передачі, Мб/с
Gigabit Ethernet	50	70
Myrinet 2000	10	200
SCI (Scalable Coherent Interface)	4	325
InfiniBand	7	1000 (пікова)

Огляд властивостей типових топологій, уведення комунікаційних примітивів операцій передачі даних, дають підстави проводити теоретичний порівняльний аналіз виконання різних паралельних алгоритмів.

1.4 Опис методики дослідження

Методика дослідження, що використана в даній кваліфікаційній роботі базується з одного боку, на загальній теорії розв'язання звичайних диференційних рівнянь та апарату чисельних методів в більш широкому спектрі ніж тільки задача Коші (розв'язок систем нелінійних алгебраїчних рівнянь, методи оцінки локальної похибки, стійкість тощо). З іншого боку, для оцінки ефективності як послідовних, так і паралельних методів, застосовано методи теорії складності – розділу сучасної теорії алгоритмів. І очевидно, що для дослідження якості паралельних методів використано підходи та моделі теорії паралельних обчислень.

Використання суперкомп'ютерів для розв'язання багатовимірних динамічних задач різної природи, (в даному випадку із зосередженими параметрами) призводить до необхідності застосування нових підходів до проектування методів та програмних додатків, які б враховували особливості реалізації на паралельній архітектурі.

В даній кваліфікаційній роботі для розробки паралельних методів використовується ієрархічна декомпозиційна методика [6-7,14], що містить наступні етапи:

– аналіз задачі та виявлення її потенційного паралелізму;

– вибір моделі програмування та схеми розпаралелювання;

– розподілення процесу обчислень на частки, які можуть бути виконані одночасно;

– визначення необхідних інформаційних взаємозв'язків для визначених паралельних процесів обробки даних;

– розподіл сформованого набору процесів обробки даних по процесорам (відображення паралельної схеми виконання завдання на архітектуру комп'ютерної обчислювальної системи).

Даний підхід найчастіше використовується для обчислювальних систем з розподіленою пам'яттю і моделлю передачі повідомлень.

Для опису та побудови паралельних алгоритмів та їх відображень на реальні багатопроцесорні структури використано апарат графів впливу [6-7].

Теоретична оцінка якості паралельних алгоритмів базується на застосування загальноприйнятої системи метрик. Метрики паралелізму або динамічні характеристики паралельних обчислень – це система показників, що дозволяє оцінювати переваги, які можуть бути отримані при паралельному розв'язанні завдань на p процесорах, у порівнянні з послідовним вирішенням тих же завдань на одному процесорі.

Загальноприйняті і найбільш використовувані в паралельних обчисленнях динамічні показники та їх позначення:

1) T_1 – час, необхідний для розв'язання задачі певного розміру на одному процесорі за допомогою найкращого (розглядаємого) послідовного алгоритму;

2) $T_{p,comp}$ – час для розв'язання задачі з використанням паралельного алгоритму на комп'ютері з p процесорів без обліку обмінних операцій (час реалізації обчислень/арифметичних операцій);

3) $T_{p,comm}$ – комунікаційна трудомісткість або складність алгоритму (час виконання операцій міжпроцесорного обміну);

4) T_p – загальний час реалізації паралельного алгоритму на паралельній архітектурі: $T_p = T_{p,comp} + T_{p,comm,}$;

5) Z – доля часу операцій обміну в загальному часі виконання паралельного алгоритму: $Z = T_{p,comm}/T_p$;

6) Dop – ступінь паралелізму або максимальна кількість процесорів, що можуть бути використані при виконанні паралельного алгоритму.

Коефіцієнт прискорення (speedup) – одна з найбільш важливих динамічних характеристик паралельного алгоритму. Коефіцієнт прискорення визначається, як відношення часу розв’язання задачі на однопроцесорній обчислювальної системі до часу виконання паралельного алгоритму.

Коефіцієнт потенційного прискорення враховує тільки внутрішній паралелізм методу розв'язання задачі без урахування операцій обміну та інших накладних витрат: $S_{pot} = T_1 / T_{p,comp}$, коефіцієнт реального прискорення враховує операції обміну між процесорами: $S_p = T_1 / T_p$.

Наступною важливою характеристикою паралельного алгоритму є коефіцієнт ефективності (efficiency), що визначає середню частку часу виконання алгоритму, протягом якого процесори реально використовуються для розв'язання задачі, тобто якість завантаження обладнання: $E_p = S_p/p = T_1 / (p \times T_p)$. Динамічні характеристики паралельних алгоритмів є функціями багатьох параметрів (ОС, завдання та методу його розв'язання) і, насамперед, кількості процесорів паралельного комп'ютеру, p та розміру розв'язуваного завдання, m.

Під час аналізу ефективності паралельних алгоритмів необхідно враховувати тимчасові машинно-залежні параметри, що впливають на швидкість виконання паралельних обчислень. Такими параметрами є час виконання арифметичної операції додавання/добутку t_{ad}, t_{mul}, або час виконання довільної операції з рухомою точкою. Кількість операцій з рухомою точкою в секунду (*Floating Point Operations Per Second – FLOPS*) є характеристикою продуктивності процесора. Час виконання операції з рухомою точкою будемо позначати: $t_{op} = 1/FLOPS$. При підрахунку динамічних характеристик алгоритмів часто вважається, що будь-яка арифметична операція з рухомою точкою виконується за один і той же час t_{op} незалежно від виду операції. Це припущення справедливе для більшості комп'ютерів RISC архітектури.

1.5 Висновки за розділом

У першому розділі дано визначення задачі Коші для системи звичайних диференційних рівнянь та наведено сучасну класифікацію методів розв’язання СЗДР, описані сучасні класи методів розв’язання систем звичайних диференційних рівнянь.

Наведено визначення жорсткості для початкової задачі та особливості методів застосування для вирішення таких задач та задач з особливостями.

Проведено аналіз існуючих класифікацій та топології паралельних обчислювальних систем, що дозволяє визначитись з їх вибором для конкретних поставлених задач.

Описані основні комунікаційні операції для методів вирішення динамічних задач та моделі оцінювання часу передачі даних для різних топологій у режимі передачі повідомлень з орієнтацією на інтерфейс MPI.

Проведено опис методики дослідження, що базується на технології декомпозиційної ієрархічної методики, наведено систему метрики оцінювання якості паралельних обчислень.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПАРАЛЕЛЬНИХ НЕЯВНИХ БАГАТОСТАДІЙНИХ МЕТОДІВ РУНГЕ-КУТТИ

За допомогою задачі Коші для багатовимірних систем звичайних диференційних рівнянь можна моделювати різні процеси (наприклад, економічні, технічні тощо), що є широким класом завдань, для вирішення яких застосування паралельної обчислювальної техніки є, безумовно, необхідністю.

Дослідження методів розв’язання динамічних задач із зосередженими параметрами виявило, що паралельні властивості таких методів багато в чому визначаються видом чисельної схеми, покладеної в їх основу. Найменш трудомісткими і найбільш поширеними є явні багатостадійні методи, проте властиві цим схемам недоліки, зокрема умовна стійкість, істотно обмежують сферу їх застосування. У зв'язку з цим значний інтерес представляють методи, що базуються на неявних схемах (Радо, Лобатто тощо), які, не дивлячись на велику обчислювальну складність, не мають альтернативи серед однокрокових методів особливо при вирішенні жорстких задач [4-5].

Серед неявних багатостадійних однокрокових методів за типом схем Рунге-Кути розрізняють:

1) повністю неявні методи Рунге-Кутти (ПНМРК);

2) напів’явні МРК або діагонально-неявні методи типу Рунге-Кутти (ДНМРК);

3) однократно діагонально-неявні методи типу Рунге-Кутти (ОДНМРК).

В другому розділі найбільш детально розглядаються ПНМРК для одного та системи ЗДР, оскільки їх характеристики стійкості подібні до відповідних характеристик стійкості блокових неявних методів. За рахунок цього ці два

класи методів мають одні й ті ж області застосування та різні обчислювальні витрати, як у послідовному так і у паралельному варіантах (див. розділ 3).

2.1 Загальна характеристика однокрокових повністю неявних багатостадійних методів

Чисельне вирішення задачі Коші повністю неявним s-стадійним методом типа Рунге-Кутти (ПНМРК) можна отримати послідовно за кроками:

$$Y_{n+1} = Y_n + h \cdot \sum_{i=1}^{s} b_i K_i, i = \overline{1,s}, \quad (2.1)$$

$$K_i = F\left[x_n + c_i h;\ Y_n + h \cdot \sum_{i=1}^{s} a_{ij} K_j\right], \quad (2.2)$$

де $C = (c_1, \dots, c_s)^T$, $B = (b_1, \dots, b_s)^T$ та $A = \left(a_{ij}\right), i,j = \overline{1,s}$ задають унікальний варіант методу й вибираються з міркувань точності;

$K_i, i = \overline{1,s}$ – вектор стадійних коефіцієнтів;

h – крок інтегрування;

s – кількість стадій методу, що визначає його порядок апроксимації;

Y_n – вектор наближеного розв'язку в точці n.

Традиційно ПНМРК описується матрицею або таблицею Батчера (рис. 2.1), і є на відміну від явних схем (нижня трикутна) є повністю заповненою.

C	A
	B^{T}

c_1	a_{11}	a_{12}	…	a_{1s-1}	a_{1s}
c_2	a_{21}	a_{22}	…	a_{2s-1}	a_{2s}
c_3	a_{31}	a_{32}	…	a_{3s-1}	a_{3s}
…	…	…	…	…	…
c_s	a_{s1}	a_{s2}	…	a_{ss-1}	a_{ss}
	b_1	b_2	…	b_{s-1}	b_s

Рисунок 2.1 – Матриця Батчера для s-стадійного повністю неявного методу Рунге-Кутти

До переваг повністю неявних однокрокових методів [5-7] слід віднести добрі характеристики стійкості, достатні для розв'язання жорстких задач Коші. Недоліками цих методів є висока обчислювальна складність, обумовлена необхідністю розв'язання систем нелінійних алгебраїчних рівнянь (рис. 2.2) на базі деякого ітераційного процесу для визначення стадійних коефіцієнтів.

$$\begin{cases} \bar{k}_1 = \bar{f}[x_n + c_1 h;\ \bar{y}_n + h(a_{11}\bar{k}_1 + a_{12}\bar{k}_2 + \ldots + a_{1s}\bar{k}_s)] \\ \bar{k}_2 = \bar{f}[x_n + c_2 h;\ \bar{y}_n + h(a_{21}\bar{k}_1 + a_{22}\bar{k}_2 + \ldots + a_{2s}\bar{k}_s)] \\ \qquad\qquad \ldots \\ \bar{k}_s = \bar{f}[x_n + c_s h;\ \bar{y}_n + h(a_{s1}\bar{k}_1 + a_{s2}\bar{k}_2 + \ldots + a_{ss}\bar{k}_s)]. \end{cases}$$

Рисунок 2.2 − Система нелінійних алгебраїчних рівнянь (СНАР) для визначення вектору стадійних коефіцієнтів $K_i = \bar{k}_i, i = \overline{1,s}$

До методів розв'язання отриманої СНАР слід віднести схеми простої ітерації та підходи Ньютона. Наприклад, підчас вирішення СЗДР повністю неявними методами $m \times s$ стадійних коефіцієнтів можуть бути отриманими на основі наступної ітераційної схеми:

$$K_i^{(0)} = F\left[x_n + c_i h;\ G_i^{(0)}\right], \tag{2.3}$$

$$K_i^{(l+1)} = F\left[x_n + c_i h;\ G_i^{(l)}\right], \tag{2.4}$$

де $$G_i^{(0)} = Y_0, \tag{2.5}$$

$$G_i^{(l)} = Y_n + h \cdot \sum_{i=1}^{s} a_{ij} K_j^{(l)}, \tag{2.6}$$

$$i = \overline{1,s}, l = \overline{1,N}.$$

Обчислювальна схема (2.3)-(2.6) описує ітераційний процес, повторений l −разів, та представляє $\bar{k}_i^{(l)},\ i = \overline{1,s},$ як $l-$ ту апроксимацію стадійного коефіцієнта $\bar{k}_i$. Детальніше ітераційний процес за схемою простої функціональної показано на рисунках 2.3-2.5.

$$\begin{cases} k_1^{(0)} = f[x_n + c_1 h;\ y_n)] \\ k_2^{(0)} = f[x_n + c_2 h;\ y_n)] \\ \quad \dots \\ k_s^{(0)} = f[x_n + c_s h;\ y_n)]. \end{cases}$$

Рисунок 2.3 – Нульова ітерація розв'язання СНАР для обчислення вектору стадійних коефіцієнтів

$$\begin{cases} k_1^{(1)} = f[x_n + c_1 h;\ y_n + h(a_{11}k_1^{(0)} + a_{12}k_2^{(0)} + \dots + a_{1s}k_s^{(0)})] \\ k_2^{(1)} = f[x_n + c_2 h;\ y_n + h(a_{21}k_1^{(0)} + a_{22}k_2^{(0)} + \dots + a_{2s}k_s^{(0)})] \\ \quad \dots \\ k_s^{(1)} = f[x_n + c_s h;\ y_n + h(a_{s1}k_1^{(0)} + a_{s2}k_2^{(0)} + \dots + a_{ss}k_s^{(0)})]. \end{cases}$$

Рисунок 2.4 – Перша ітерація розв'язання СНАР для обчислення вектору стадійних коефіцієнтів

$$\begin{cases} k_1^{(l)} = f[x_n + c_1 h;\ y_n + h(a_{11}k_1^{(l-1)} + a_{12}k_2^{(l-1)} + \dots + a_{1s}k_s^{(l-1)})] \\ k_2^{(l)} = f[x_n + c_2 h;\ y_n + h(a_{21}k_1^{(l-1)} + a_{22}k_2^{(l-1)} + \dots + a_{2s}k_s^{(l-1)})] \\ \quad \dots \\ k_s^{(l)} = f[x_n + c_s h;\ y_n + h(a_{s1}k_1^{(l-1)} + a_{s2}k_2^{(l-1)} + \dots + a_{ss}k_s^{(l-1)})]. \end{cases}$$

Рисунок 2.5 – l-та ітерація розв'язання СНАР для обчислення вектору стадійних коефіцієнтів

2.2 Розробка та оцінка ефективності паралельного алгоритму ПНМРК для ЗДР

Паралельне розв'язання звичайних диференційних рівнянь на базі повністю неявного методу типу Рунге-Кутти концентрується на виконанні одного кроку інтегрування. Як і у разі явних методів, алгоритм складається з двох складових:

1) визначення стадійних коефіцієнтів $k_i, i = \overline{1,s}$;

2) наближеного розв'язку на наступному кроці $y_{n+1}(h)$.

Зауважимо, що оскільки метод інтегрування є неявним, то для визначення стадійних коефіцієнтів потрібно розв'язати систему нелінійних у загальному випадку алгебраїчних рівнянь, метод розв'язання яких містить свій внутрішній паралелізм. Більш того, в неявних методах на відміну від явних, навіть для одного рівняння є можливість використовувати паралелізм методу.

Паралельний алгоритм уведеної мікрооперації, розроблений з використанням графа впливу, зображено на рис. 2.6.

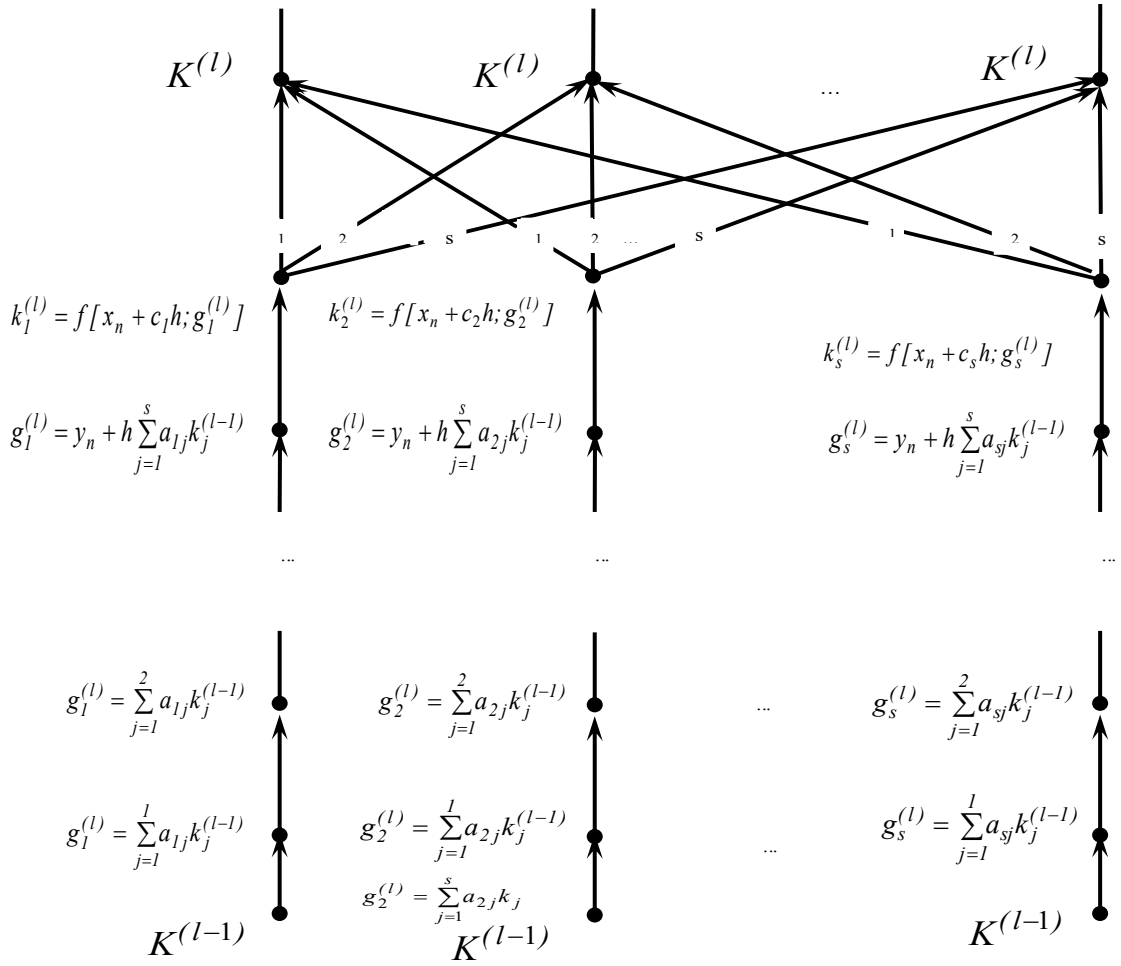


Рисунок 2.6 – Граф впливу обчислення l – тої ітерації стадійних векторів $K^{(l)} = (k_1^{(l)}, k_2^{(l)}, ..., k_s^{(l)})$ ПНМРК для ЗДР

Розподілимо обчислення кожного зі стадійних векторів на окремий процесор у межах одного кроку ітерації. Тоді, послідовний алгоритм за

обчислювальною схемою 2.1-2.6 може бути представлений, як $N+1$ підзадача, що полягає у виконанні однотипних обчислень одного кроку за методом функціональної ітерації. Тому, як основна макрооперація, вводиться задача обчислення всіх коефіцієнтів $K^{(l)} = \left(k_1^{(l)}, k_2^{(l)}, ..., k_s^{(l)}\right)$ на одному l-тому кроці ітераційного процесу.

Час послідовного методу для одного рівняння при реалізації ПНМРК може бути описаний наступними співвідношеннями (рис. 2.7).

$$T_1^{\text{ПНМРК}} = \sum_{i=1}^{s} T(k_i^{(0)}) + \sum_{i=1}^{s} T(k_i^{(l)}) + T(y_{n+1})$$

$$T_1^{\text{ПНМРК}} = s(T_f + 2t_{op}) + N\left[2s^2 t_{op} + sT_f\right] + 2(s+1)\, t_{op}$$

Рисунок 2.7 – Час послідовного виконання для одного ЗДР при реалізації ПНМРК

Рисунок 2.8 представляє паралельний алгоритм вирішення одного диференціального рівняння повністю неявним методом на мультикомп'ютері з p процесорів, об'єднаних однорідною комутаційною мережею. Розподіл даних збігається з розподілом даних для приведеної схеми макрооперації, на кожному кроці ітераційного процесу паралельно обчислюються стадійні коефіцієнти, потім послідовно наближене вирішення на $n+1$ кроці.

Для паралельного алгоритму за описаною обчислювальною схемою та відображенням на паралельну архітектуру з розподіленою пам’яттю, ПНМРК у застосуванні до одного звичайного диференційного рівняння, час виконання обчислень на p процесорах з урахуванням обмінів та інших накладних витрат складається з двох доданків: $T_p^{\text{ПНМРК}} = T_{p,comp}^{\text{ПНМРК}} + T_{p,comm}^{\text{ПНМРК}}$.

Зауважимо, що в цьому випадку максимальний ступінь паралелізму алгоритму ПНМРК дорівнює: $Dop = s$, де це s – кількість стадій методу, що визначає в кінцевому випадку похибку його апроксимації.

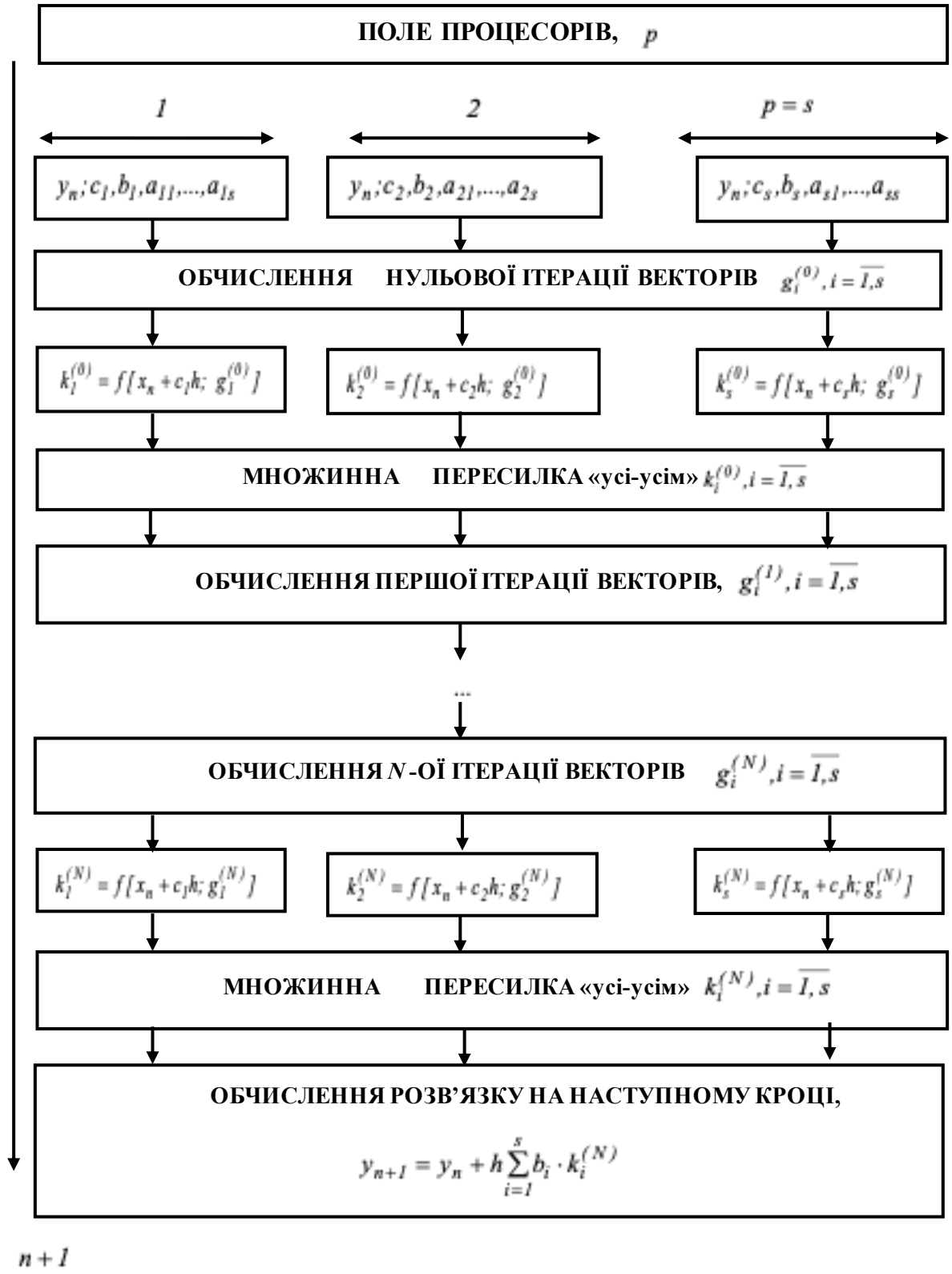


Рисунок 2.8 – Обчислювальна схема паралельного алгоритму повністю неявного однокрокового методу типа Рунге-Кутти для ЗДР

Перший доданок або час паралельних обчислень у якості тимчасових параметрів має флоп (t_{op}) або час виконання однієї операції з плаваючою точкою та час обчислення правої частини ЗДР (T_f):

$$T_{p,comp}^{\text{ПНМРК}} = (T_f + 2t_{op}) + N\left[2st_{op} + T_f\right] + +2(s+1)\,t_{op}. \qquad (2.7)$$

Коефіцієнт потенційного прискорення для ЗДР зі складною правою частиною визначається так:

$$S_{pot}^{\text{ПНМРК}} = {T_1^{\text{ПНМРК}}}\Big/{T_{p,comp}^{\text{ПНМРК}}},$$

$$S_{pot}^{\text{ПНМРК}} \approx \frac{s(N+1)T_f}{(N+1)T_f} = s = p.$$

Аналогічний результат маємо для тривіальних правих частин.

Час, необхідний на реалізацію міжпроцесорних обмінів даного алгоритму визначається $N+1$ раз повтореною множинною операцією за типом "усі-усім" для p процесорів: $T_{p,comm}^{\text{ПНМРК}} = (N+1)T_{all_{to_{all}}}(p)$.

– кільце: $T_{p,comm}^{\text{ПНМРК},R} = (N+1)(t_s + t_w)\,(p-1)$;

– 2D-тор: $T_{p,comm}^{\text{ПНМРК},M} = (N+1)\left[2t_s\left(\sqrt{p}\,-1\right) + t_w(p-1)\right]$;

– гіперкуб: $T_{p,comm}^{\text{ПНМРК},H} = (N+1)[t_s log_2 p + t_w(p-1)]$.

Реальна ефективність (рис. 2.8) паралельних обчислень багато в чому визначається трудомісткістю комунікаційних операцій.

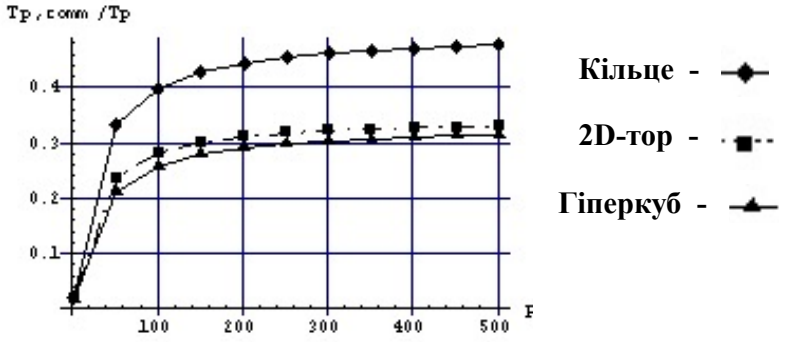


Рисунок 2.8 – Доля комунікаційних операцій до загальних накладних витрат паралельного алгоритму ПНМРК для ЗДР

Аналіз запропонованих топологій, як для високошвидкісних, так і для низькошвидкісних мереж показав, що, безумовно, найгіршим варіантом для даного алгоритму є з'єднання типу кільце (рис. 2.9).

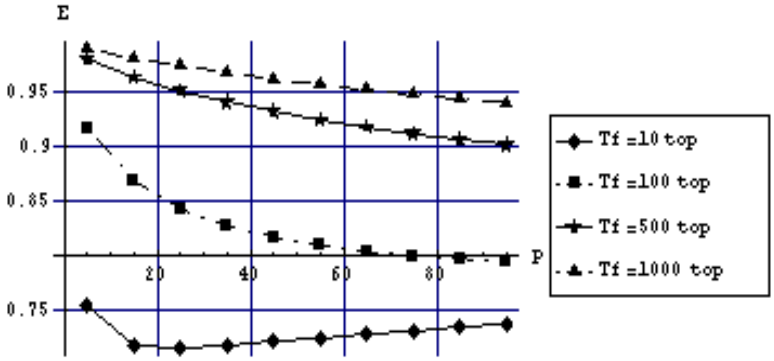


Рисунок 2.9 – Коефіцієнт ефективності паралельного ПНМРК для ЗДР при різній складності правої частини

Динамічні характеристики ПНМРК для ЗДР мають великий розкид по значеннях. Суттєве значення має такий фактор, як складність правої частини диференційного рівняння (рис. 2.9). Для тривіальних правих частин коефіцієнт ефективності замалий, для домінуючих достатньо великий. Це дозволяє зробити висновок щодо можливості вживання паралельних алгоритмів ітераційних методів при ретельному обліку всіх складових паралельної системи, алгоритму та вхідної задачі для того, щоб це розв’язання було ефективним та таким, що масштабується.

2.3 Особливості розпаралелювання повністю неявного методу Рунге-Кутти для СЗДР при рішенні нелінійної задачі

При паралельному розв’язанні систем звичайних диференціальних рівнянь (СЗДР) із використанням повністю неявних методів Рунге-Кутти окрім паралелізму методу з'являється можливість використовувати і системний паралелізм.

Вживання ПНМРК до СЗДР вимагає вирішення системи нелінійних рівнянь для визначення $\bar{k}_i = (k_{i1}, \dots, k_{im}), i = \overline{1,s}$.

Метод простої ітерації для отримання стадійних коефіцієнтів ПНМРК:

$$\begin{cases} \bar{g}_i^{(0)} = 0, \ \bar{k}_i^{(0)} = F(x_n + c_i h, \bar{y}_n), \\ \bar{g}_i^{(l)} = a_{i1}\bar{k}_1^{(l)} + a_{i2}\bar{k}_2^{(l)} + \dots + a_{is}\bar{k}_s^{(l)}, \\ \bar{k}_i^{(l)} = F[x_n + c_i h, \bar{y}_n + h \cdot g_i^{(l-1)}], \\ i = 1,\dots,s; l = 1,\dots,N. \end{cases}$$

Число нелінійних рівнянь, як і кількість невідомих компонентів стадійних векторів $k_{ij,}\, i = \overline{1,s},\ j = \overline{1,m}$ дорівнює sm.

Як і для всіх методів, що вже були розглянуті, розпаралелювання повністю неявних методів типу Рунге-Кутти базується на виконанні одного кроку інтегрування. Послідовний алгоритм можна представити як виконання двох підзадач: вирішення системи нелінійних рівнянь і обчислення чергової апроксимації вирішення.

Для здобуття ефективного паралельного алгоритму розв’язання поставленої задачі проаналізуємо внутрішній паралелізм кожної з підзадач. Це дозволить визначити рівень декомпозиції задачі на незалежні процеси і ввести оптимальну множину макрооперацій.

Обчислювальна схема паралельного ПНМРК для розв’язання нелінійної задачі Коші на основі СЗДР приведена на рисунку 2.10. Кількість нелінійних рівнянь, як і кількість невідомих компонент стадійних векторів $k_{ij}\ \left(i = \overline{1,s};\ j = \overline{1,m}\right)$ дорівнює $s \times m$. Ітераційний процес, що описується обчислювальною схемою, є суто послідовним. Проте він дозволяє розподілити обчислення l-тої ітерації $K(h)$ на s незалежних процесів по кількості стадійних векторів, де кожен вектор має розмірність, що дорівнює розміру системи m.

Таким чином, максимальний ступінь паралелізму цієї частини ПНМРК складає $Dop_1 = s \times m$, для другої підзадачі – $Dop_2 = m$.

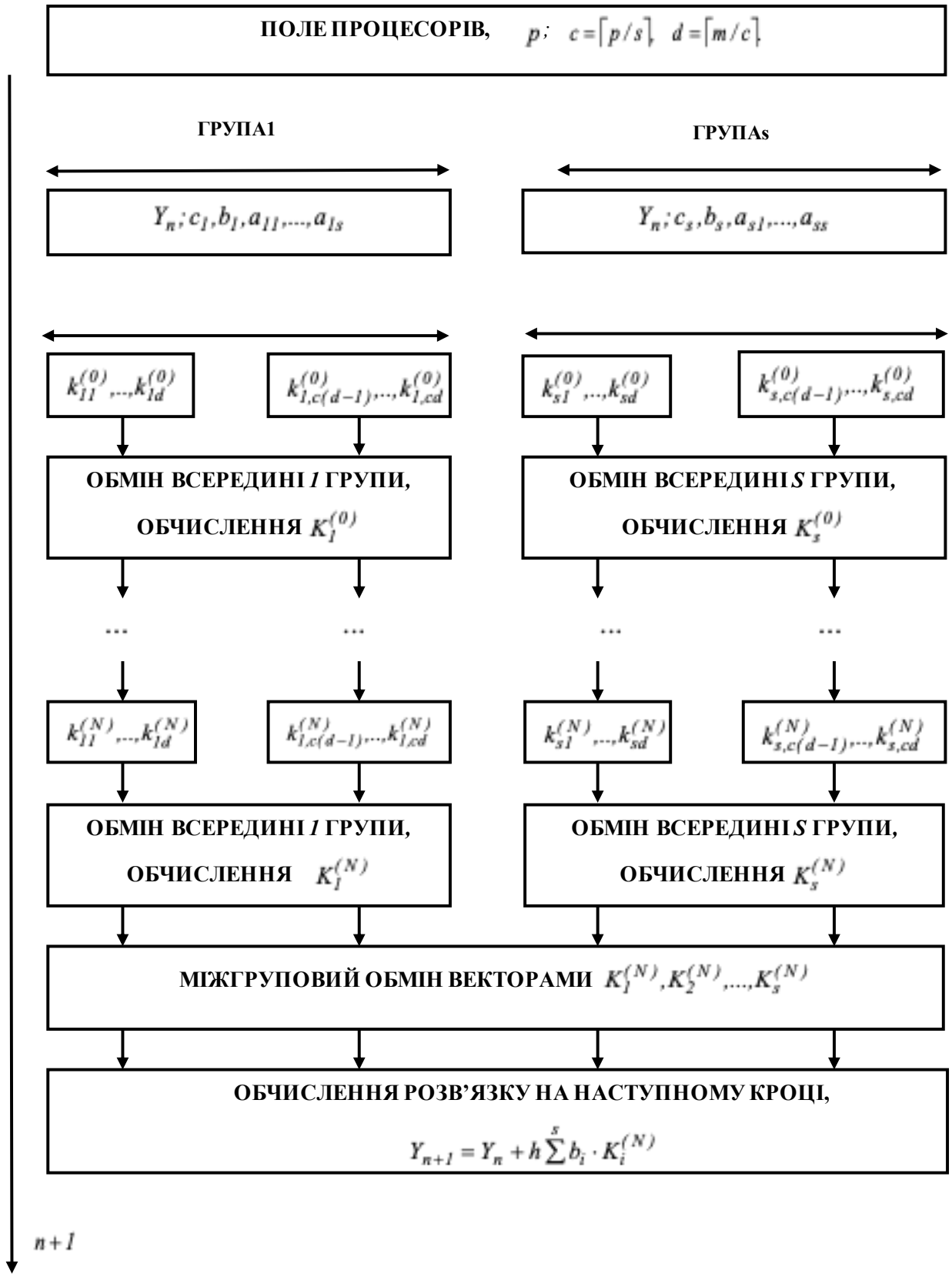


Рисунок 2.10 – Обчислювальна схема паралельного алгоритму повністю неявного однокрокового методу типу Рунге-Кутти для системи звичайних диференційних рівнянь

Якщо скористатися шириною всього паралельного алгоритму, рівною $Dop_1 = p = s \cdot m$, то при реалізації другої підзадачі велика частина процесорів простоюватиме ($Dop_2 = p = m$). Якщо за основу узяти $Dop_2 = m$, то не буде використаний у повному обсязі внутрішній паралелізм першої підзадачі.

Найбільш ефективне вирішення полягає у комбінації першого і другого підходів. Розіб'ємо всю множину процесорів на s груп, кожна група відповідатиме за обчислення одного стадійного коефіцієнта $\bar{k}_i = (k_{i1}, k_{i2}, ..., k_{im}), i = \overline{1,s}$, кількість процесорів в групах однакова і рівна $p_i = \lceil p/s \rceil = c, \forall i = \overline{1,s}$. Далі кожен з p_i процесорів s-тої групи паралельно обчислюватиме $d = \lceil m/p_i \rceil$ компонент вектору $\bar{g}_i$ на кожній з $N+1$ ітерацій. Усередині ітераційного процесу кожний процесор у групі обмінюється своїми даними з усіма іншими (обмін «усі-усім» у групі) процесами групи компонентами вектору $\bar{g}_i$. Відмітимо, що немає необхідності у внутрішньо груповому обміні векторами $\bar{k}_i, i = \overline{1,s}$, оскільки на наступному кроці ітерації кожен процесор в групі обчислюватиме ті ж компоненти стадійних векторів.

Після закінчення ітераційного процесу необхідно перерозподілити стадійні вектори між процесорами, для цього має бути проведений обмін даними між групами – міжгруповий обмін за типом “усі-усім”. При обчисленні апроксимації розв’язку на кожному тимчасовому кроці на одному процесорі розташовується $\lceil m/p \rceil$ компонент вектору розв’язку.

Міжгруповий обмін може бути здійснений двома способами:

1) кожен перший процесор у групі передає вектор $\bar{k}_i$ в перший елемент кожній іншій групи і проводиться груповий обмін у групі;

2) міжгрупова передача за типом “усі-усім”.

Визначимо динамічні характеристики отриманого алгоритму.

Час обчислень за послідовною реалізацією дорівнює:

$$T_1^{\text{ПНМРК,СЗДР}} = (N+1)s \cdot \sum_{i=1}^{m} T_{f_i} + (2Nms^2 + 2ms + 2s + 2m) \cdot t_{op.} \quad (2.8)$$

Час обчислень для алгоритму рис. 2.10 залежить від типу паралельної обчислювальної системи. Для SIMD архітектури у загальному випадку обчислення різних компонент вектору функцій F не може бути виконане паралельно, отже, на це буде потрібно:

$$T_F = \sum_{i=1}^{m} T_{f_i}. \tag{2.9}$$

У той же час для MIMD систем з урахуванням можливості паралельної реалізації звернення до правої частини СЗДР, час для обчислення ПНМРК дорівнює:

$$T_{p,comp}^{\text{ПНМРК,СЗДР}} = \underbrace{t_{mul} + t_{ad} + \left\lceil \frac{ms}{p} \right\rceil T_F}_{\bar{k}^{(0)}} + N \cdot \underbrace{\left\lceil \frac{ms}{p} \right\rceil [T_F + s(t_{mul} + t_{ad})]}_{\bar{k}^{(l)}} +$$

$$+ \underbrace{\left\lceil \frac{m}{p} \right\rceil (s+1)(t_{mul} + t_{ad})}_{\bar{y}_{n+1}}, T = \max_{i=1}^{m} T_{f_i} \tag{2.10}$$

Легко переконатися, що коефіцієнт потенційного прискорення практично лінійно залежить від кількості процесорів як при T_F»t_{op}, так і при $T_F \approx t_{op}$. Накладні витрати на операції обміну включають час пересилок даних всередині групи та між групами:

$$T_{p,comm}^{\text{ПНМРК,СЗДР}} = T_{p,comm;1} + T_{p,comm;2}. \tag{2.11}$$

Проаналізуємо складність комунікаційної складової паралельного алгоритму ПНМРК для СЗДР. Внутрішньогрупова операція обміну даними є $N+1$ разів повтореною операцією множинної пересилки за типом “усі-усім” на p_i процесорах, обсяг даних, що пересилаються, рівний d:

$$T_{p,comm;1} = (N+1) \cdot T_{all-to-all}(d, p_i). \tag{2.12}$$

Для різних базових топологій внутрішньогруповий обмін потребує:

– кільце: $T_{p,comm;1}^{\text{ПНМРК}-\text{СЗДР},R} = (N+1)\left(t_s + \left\lceil \frac{ms}{p} \right\rceil t_w\right)\left(\left\lceil \frac{p}{s} \right\rceil - 1\right)$;

– тор: $T_{p,comm;1}^{\text{ПНМРК}-\text{СЗДР},M} = (N+1)\left[2\left(\sqrt{\left\lceil \frac{p}{s} \right\rceil} - 1\right)t_s + \left\lceil \frac{ms}{p} \right\rceil \left(\left\lceil \frac{p}{s} \right\rceil - 1\right) t_w\right]$;

– гіперкуб: $T_{p,comm;1}^{\text{ПНМРК}-\text{СЗДР},H} = (N+1)\left[log_2\left(\left\lceil\frac{p}{s}\right\rceil\right)t_s + \left\lceil\frac{ms}{p}\right\rceil\left(\left\lceil\frac{p}{s}\right\rceil - 1\right)t_w\right]$.

Міжгруповий обмін виконується двічі для обміну стадійними коефіцієнтами та перегрупування даних після обчислення чергової апроксимації рішення; він вимагає наступних тимчасових витрат:

– кільце: $T_{p,comm;2}^{\text{ПНМРК}-\text{СЗДР},R} = \left[2t_s + \left(\left\lceil\frac{m}{p}\right\rceil + \left\lceil\frac{ms}{p}\right\rceil\right)t_w\right](p-1)$;

– 2D-тор: $T_{p,comm;2}^{\text{ПНМРК}-\text{СЗДР},M} = 4\left(\sqrt{p}\ - 1\right)t_s + \left(\left\lceil\frac{m}{p}\right\rceil + \left\lceil\frac{ms}{p}\right\rceil\right)(p-1)t_w$;

– гіперкуб: $T_{p,comm;2}^{\text{ПНМРК}-\text{СЗДР},H} = 2log_2 p\ t_s + \left(\left\lceil\frac{m}{p}\right\rceil + \left\lceil\frac{ms}{p}\right\rceil\right)(p-1)t_w$.

Динамічні характеристики отриманого паралельного методу розв'язання систем ЗДР на базі ПНМРК аналогічні відповідним характеристикам інтегрування одного звичайного рівняння.

2.4 ПНМРК з оцінкою локальної апостеріорної похибки для керування кроком інтегрування

Одним з основних питань, що виникають при чисельному інтегруванні СЗДР, є проблема оцінки похибки наближеного розв'язку. Апріорна оцінка глобальної похибки чисельного методу дозволяє судити про збіжність наближеного розв'язання задачі до точного і, отже, його застосування. Апостеріорна оцінка локальної похибки, що одержується на кожному кроці обчислень, дозволяє автоматично вибирати крок інтегрування, що забезпечує задану точність наближеного рішення з мінімальними обчислювальними витратами.

Однокрокові методи безпосередньо не надають простої можливості отримати інформацію про локальну точність наближеного розв'язку за результатами проміжних обчислень.

Відомими та загальноприйнятими методами оцінки локальної апостеріорної похибки розв’язання СЗДР є:

- дублювання кроку за правилом Рунге,

- технологія локальної екстраполяції Річардсона,

- вкладені методи або методи вкладених форм [6-7].

Найбільш простий спосіб визначення локальної похибки це дублювання кроку за правилом Рунге. Для точного розв'язку при переході з точки x_n до точки $x_n + 2h, \bar{y}(x_n + 2h)$ на основі однієї й тієї ж формули порядку r отримують дві апроксимації: $\bar{y}_{n+1}^{(1)}$ (один крок $2h$) и $\bar{y}_{n+1}^{(2)}$ (два кроки h).

Різниця між цими значеннями використовується в якості оцінки локальної похибки інтегрування, як розв'язок приймається апроксимація, отримана з половинним кроком, як найбільш точна. Процес уточнення розв'язку з половинним кроком $\bar{y}_{n+1}^{(2)}$ підвищує точність вирішення на одиницю та носить назву локальної екстраполяції. Даний спосіб оцінки похибки досить простий, однак має великі накладні витрати: обсяг обчислень на один вузол сітки зростає майже втричі.

Метод локальної екстраполяції Річардсона є узагальненням технології подвоєння кроку за правилом Рунге [6-7]. Ідея методу полягає в багаторазовому подрібненні кроку інтегрування, і також у багаторазовому застосуванні процесу обчислення, названого вище локальною екстраполяцією.

Розв'язання задачі Коші розглядається при переході з точки x_n у точку $x_{n+1} = x_n + H$, де H – базова довжина кроку. Вибирається ряд натуральних чисел $P_i = \{n_1, n_2, ..., n_k, ..\}$ такий, що: $n_1 < n_2 < ... < n_k < ...$ та, відповідно, послідовність кроків: $h_1 > h_2 > ... > h_{k-1} > h_k > ...$, де $h_i = H / n_i$. Задається опорний чисельний метод порядку r_0 та, після виконання n_i кроків інтегрування з довжиною h_i, обчислюється наближений розв'язок вхідної задачі в точці x_{n+1}:

$$T_{i,1} = \bar{y}_{h_i}(x_n + H) = \bar{y}^{(i)}_{n+1},\ i = \overline{1,k}. \tag{2.13}$$

Після виконання обчислень для ряду послідовних значень *i*, за рекурентним співвідношенням визначають $T_{i,j+1}$ – екстрапольовані значення для довільних *i,j*:

$$T_{i,j+1} := T_{i,j} + \frac{T_{i,j} - T_{i-1,j}}{(n_i / n_{i-j})^b - 1}. \tag{2.14}$$

Обчислення за (2.14) є багаторазово повтореним процесом локальної поліноміальної екстраполяції за схемою Ейткена-Невілла [6-7]. Величина *b* дорівнює одиниці в загальному випадку, для симетричних опорних методів *b* дорівнює двом.

T_{ij} – наближений розв'язок задачі Коші, отриманий чисельним методом порядку $r_0 + (j-1)\cdot b$ з кроком h_i.

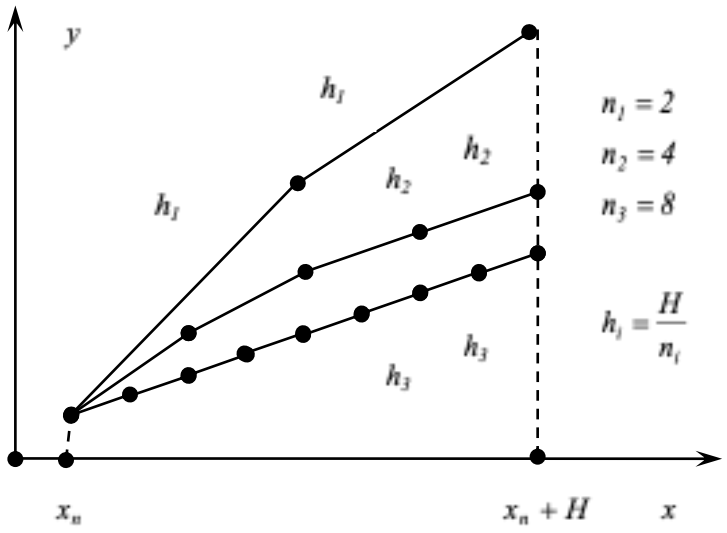


Рисунок 2.11 – Технологія локальної екстраполяції Річардсона

Переваги цього методу полягають у тому, що він дає таблицю результатів обчислень (табл. 2.1), які утворюють послідовність вкладених методів, дозволяють оцінити локальну похибку та вибрати стратегію для методів змінного порядку.

Таблиця 2.1 – Екстраполяційна таблиця

	$r_0 + b$	$r_0 + 2b$	…	$r_0 + (k-2)b$	$r_0 + (k-1)b$
T_{11}			…		
T_{21}	T_{22}		…		
…	…	...	…	T_{k-1k-1}	
T_{k1}	T_{k2}	T_{k3}	…	T_{k-1k}	T_{kk}

Альтернативним способом визначення локальної апостеріорної похибки при розв'язанні задачі Коші є вкладені методи Рунге-Кутти (ВМРК). Цей спосіб заснований на використанні двох наближених значень розв'язку в одній точці, але на відміну від правила Рунге наближення обчислюються не за одною, а за двома формулами різних порядків точності r і $\hat{r}$ із одним й тим же кроком.

$$\begin{cases} \bar{y}_{n+1} = \bar{y}_n + h_n \cdot \sum\limits_{l=1}^{s} b_l \cdot \bar{k}_l, \\ \hat{\bar{y}}_{n+1} = \bar{y}_n + h_n \cdot \sum\limits_{l=1}^{s'} b'_l \cdot \bar{k}_l, \\ d = \left\| (\hat{\bar{y}}_{n+1} - \bar{y}_{n+1}) \right\|. \end{cases} \tag{2.15}$$

Для визначення локальної похибки менш точного результату та управління величиною кроку інтегрування використовується величина d. Вкладений метод Рунге-Кутти $r(\hat{r})$ – це схема, в якій метод $\hat{r}$-го порядку ("оцінювача похибки") виходить як побічний продукт методу r-го порядку (рис. 2.12). Порядок апроксимацій: $\bar{y}_{n+1}$ та $\hat{\bar{y}}_{n+1}$ зазвичай відрізняються на 1, тобто $r = \hat{r} + 1$ або $r = \hat{r} - 1$. Обидва наближення використовують практично одні й ті ж значення стадійних коефіцієнтів, але комбінують їх по-різному. Вкладені методи Рунге-Кутти – це найменш трудомісткий з відомих способів визначення локальної апостеріорної похибки розв'язку як для явних, так і для неявних у послідовній та паралельній реалізаціях.

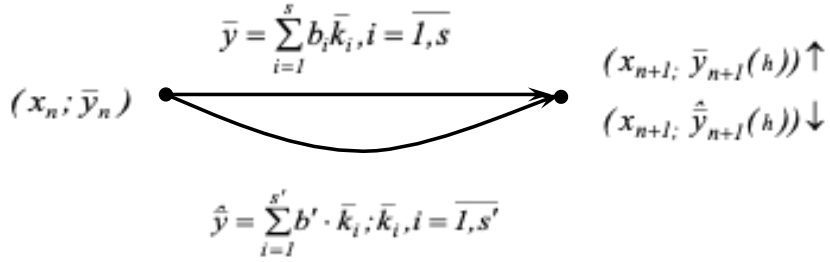


Рисунок 2.12 – Схема обчислень для вкладених методів

Дійсно, якщо для розв’язання задачі Коші використовується класичний метод Рунге-Кутти четвертого порядку з контролем точності за правилом Рунге, то потрібно одинадцять обчислень похідної – правої частини рівняння, а для вкладеного методу Фельбергу 4(5) того ж порядку точності всього 6. Обчислювальна складність вкладених методів залежить від порядку методу, проте ця величина є похідною від кількості стадій, що використовуються. Взаємозв'язок між цими величинами для відомих найбільш використовуваних вкладених методів Рунге-Кути наведено на рисунку 2.13.

Вкладені явні методи Рунге-Кути	Порядок методу, $r(\hat{r})$	Число стадій, s
Фельберга	4(5)	6
Дормана-Прінса	5(4)	7
Фельберга	7(8)	12
Дормана-Прінса	8(7)	13

Рисунок 2.13 – Зв'язок порядку вкладених методів та числа стадій для ЯМРК

Застосування ідеї вкладених форм для ПНМРК містить особливості за рахунок того, що необхідно для інтегрування розв’язати систему нелінійних рівнянь на основі деякого ітераційного процесу.

Матриця або схема Батчера для повністю неявного багатостадійного методу Рунге-Кути з вкладеними формулами має вигляд, представлений на рисунку 2.14.

c_1	a_{11}	a_{12}		$a_{1,\,s-1}$	a_{s1}
c_2	a_{21}	a_{22}	…	$a_{2,\,s-1}$	a_{2s}
c_3	a_{31}	a_{32}	…	$a_{3,\,s-1}$	a_{3s}
…	…	…	…	…	…
c_s	a_{s1}	a_{s2}	…	$a_{s,s-1}$	$a_{s,s}$
	b_1	b_2	…	b_{s-1}	b_s
	$\widehat{b}_1$	$\widehat{b}_2$	…	$\widehat{b}_{s-1}$	$\widehat{b}_s$

Рисунок 2.14 – Матриця Батчера для вкладених методів ПНМРК

Оцінки похибки для паралельних вкладених повністю неявних багатостадійних алгоритмів розв’язання нелінійної задачі Коші для ЗДР може бути реалізована за рахунок:

1) використання двох незалежних неявних методів типу Рунге-Кутти суміжних порядків точності;

2) використання двох апроксимацій розв’язку різних порядків точності на базі методу Крилова, послідовного підвищення порядку точності.

Перший підхід полягає у комбінації двох різних незалежних неявних методів суміжних порядків точності $r(\hat{r}), \hat{r} = r \pm 1$ на одній і тій же сітці інтегрування:

$$Y_{n+1} = Y_n + h \cdot \sum_{i=1}^{s} b_i K_i, i = \overline{1,s}, \tag{2.16}$$

$$K_i = F\left[x_n + c_i h;\ Y_n + h \cdot \sum_{i=1}^{s} a_{ij} K_j\right],$$

$$\widehat{Y}_{n+1} = \widehat{Y}_n + h \cdot \sum_{i=1}^{\hat{s}} \widehat{b}_i \widehat{K}_i, i = \overline{1,\hat{s}}, \tag{2.17}$$

$$\widehat{K}_i = F\left[x_n + c_i h;\ \widehat{Y}_n + h \cdot \sum_{i=1}^{\hat{s}} a_{ij} \widehat{K}_j\right],$$

де $C = (c_1, \dots, c_s)^T$, $B = (b_1, \dots, b_s)^T$, $\hat{B} = (\hat{b}_1, \dots, \hat{b}_s)^T$ та $A = (a_{ij}), i, j = \overline{1, s}$ задають унікальний варіант методу й вибираються з міркувань точності.

Тут перший метод визначає апроксимацію розв'язку на основі s-стадійного, а другий $\hat{s}$-стадійного однокрокового, методу. Другий наближений розв'язок у відповідних точках інтегрування використовується для оцінки апостеріорної локальної похибки і дає порядок на 1 більше ніж перший:

$$K_i^{(0)} = F\left[x_n + c_i h;\ G_i^{(0)}\right], \tag{2.18}$$
$$K_i^{(l+1)} = F\left[x_n + c_i h;\ G_i^{(l)}\right],$$

де $G_i^{(0)} = Y_0, G_i^{(l)} = Y_n + h \cdot \sum_{i=1}^{s} a_{ij} K_j^{(l)}, i = \overline{1, s}, l = \overline{1, N};$

$$\widehat{K}_i^{(0)} = F\left[x_n + c_i h;\ \widehat{G}_i^{(0)}\right], \tag{2.19}$$
$$\widehat{K}_i^{(l+1)} = F\left[x_n + c_i h;\ \widehat{G}_i^{(l)}\right],$$

де $\widehat{G}_i^{(0)} = \widehat{Y}_0, \widehat{G}_i^{(l)} = \widehat{Y}_n + h \cdot \sum_{i=1}^{\hat{s}} a_{ij} \widehat{K}_j^{(l)}, i = \overline{1, \hat{s}}, l = \overline{1, \widehat{N}}.$

Другий підхід розглядає дві послідовні ітерації одного ПНМРК в рамках використаного ітераційного процесу, очевидно номери ітерації менше ніж максимальна кількість, що дає порядок методу.

$$K_i^{(0)} = F\left[x_n + c_i h;\ G_i^{(0)}\right], \tag{2.20}$$
$$K_i^{(l1)} = F\left[x_n + c_i h;\ G_i^{(l1-1)}\right],$$
$$K_i^{(l2)} = F\left[x_n + c_i h;\ G_i^{(l2-1)}\right], l1, l2 \le N,$$

де $G_i^{(0)} = Y_0, G_i^{(l)} = Y_n + h \cdot \sum_{i=1}^{s} a_{ij} K_j^{(l)}, i = \overline{1, s}, l = \overline{1, N};$

Час послідовної реалізації першого вкладеного ПНМРК вдвічі перевищує такий же час для однієї унікальної схеми ПНМРК:

$$T_1^{\text{ПНМРК,СЗДР}} = 2(N+1)s \cdot \sum_{i=1}^{m} T_{f_i} + (2Nms^2 + 2ms + 2s + 2m) \cdot t_{op.}$$

Час виконання послідовного алгоритму при використанні другого підходу на основі ідеї вкладеності дає зростання часу виконання такого ж

алгоритму для однієї схеми на константу, що в залежить кількості стадій методу, але в більшості випадків дорівнює одиниці, якщо точність вкладеного на 1 більше базового методу.

Підходи до розпаралелювання в даному випадку нічим не відрізняються від відповідних підходів і отриманих методів у разі ПНМРК без отримання локальної похибки.

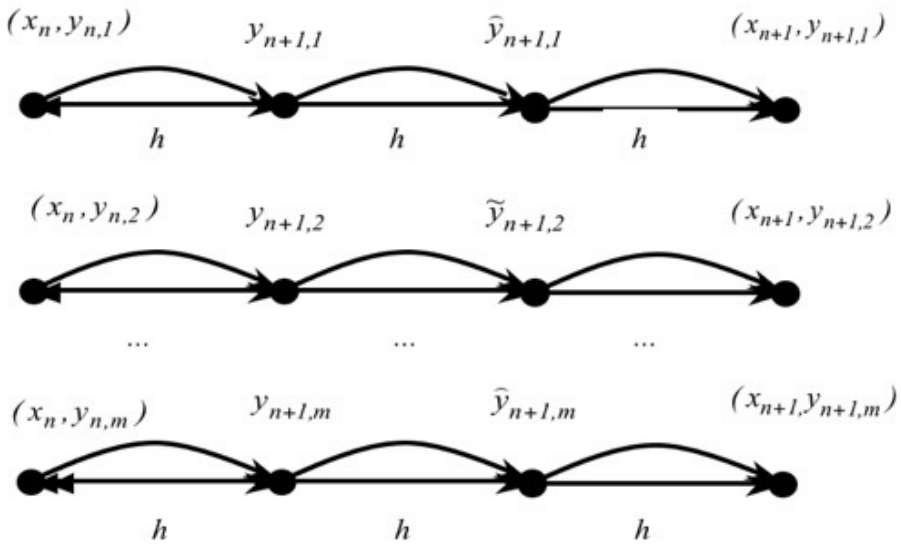


Рисунок 2.15 – Граф впливу для вкладеного методу ПНМРК

Динамічні характеристики отриманих вкладених повністю неявних методів, характер їх поведінки, аналогічні щодо відповідних характеристик ПНМРК. Як і в попередніх випадках з'єднання за типом кільце є найгіршим варіантом, топології гіперкуб і тор дають практично ідентичні результати. Частка обмінних операцій зменшується зі зростанням складності правої частини СЗДР для всіх типів паралельних ОС та топологій.

Коефіцієнти ефективності та прискорення паралельного вкладеного ПНМРК для СЗДР суттєво залежать від розміру системи та складності правої частини: обидві характеристики зростають з зростанням розміру системи рівнянь та складності її правої частини. Для тривіальних правих частин та невеликих за розміром систем рівнянь розпаралелювання стає неефективним.

ВПНМРК 1 ВПНМРК 2

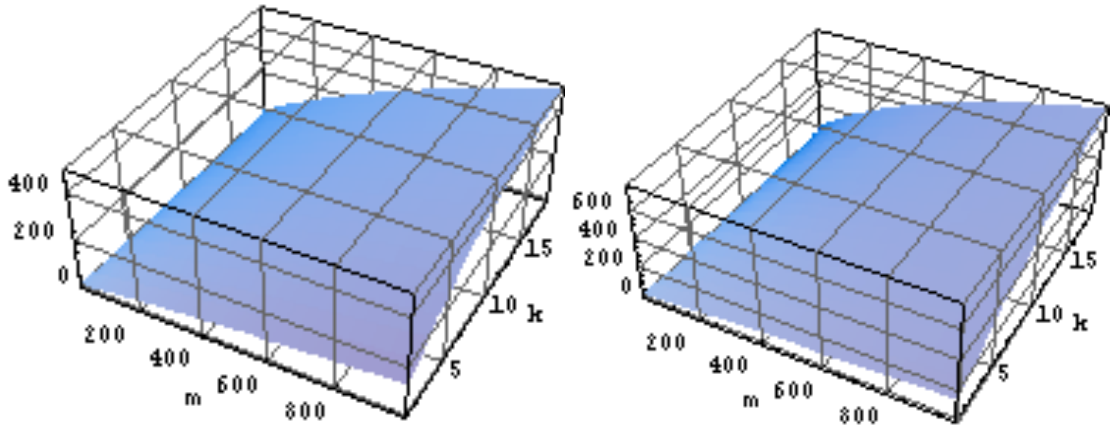


Рисунок 2.16 – Коефіцієнти прискорення вкладених ПНМРК від k порядку методу

Величина локальної похибки використовується для того, щоб забезпечити довжину кроку інтегрування, яка достатня для необхідної точності результатів, але таку, що гарантує відсутність марної обчислювальної роботи. Тому отримання надійних і в той же час простих та ефективних способів оцінки крокової похибки – одна з основних проблем, що виникають при конструюванні чисельного методу розв'язання задачі Коші в паралельній реалізації.

Узагальнюючи відомі теоретичні дослідження та обчислювальні експерименти необхідно відзначити, що, незважаючи на те, що методи вирішення СЗДР використовуються в обчислювальній практиці досить давно, розробка нових ефективних паралельних обчислювальних схем для розв'язання конкретних динамічних задач залишається актуальною проблемою.

2.5 Висновки за розділом

1. Розроблено паралельні методи повністю неявних однокрокових схем типу Рунге-Кутти для одного диференціального рівняння.

2. Здійснено модифікацію отриманих методів з метою узагальнення ПНМРК на системи ЗДР.

3. Проведено порівняльний аналіз ефективності вкладених неявних однокрокових методів розв'язання загального завдання Коші:

1) на основі s-стадійного та $\hat{s}$-стадійного ПНМРК суміжних порядків точності для послідовної та паралельної реалізацій;

2) вкладених ПНМРК суміжних порядків точності на основі послідовного підвищення.

Встановлено, що динамічні характеристики якості другого підходу перевершують відповідні характеристики комбінації різних суміжних схем для багатостадійних однокрокових методів практично в раз для послідовної і паралельної реалізацій.

4. Досліджено ефективність отриманих обчислювальних схем відображення паралельних алгоритмів на структури ОС залежно від розмірності СЗДР, кількості процесорів, типу паралельної ОС (SIMD, MIMD та кластерні системи), латентності та часу передачі даних у мережах різних топологій.

РОЗДІЛ 3
ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПАРАЛЕЛЬНИХ БЛОКОВИХ БАГАТОТОЧКОВИХ МЕТОДІВ РОЗВ'ЯЗАННЯ ЗАДАЧІ КОШІ

Блокові або багатоточкові паралельні методи розв'язання задачі Коші особливо актуальні, оскільки добре узгоджуються з архітектурою паралельних ОС і не вимагають обчислення значень у проміжних точках, що значно підвищує ефективність розрахунків. Дані методи володіють достатніми характеристиками стійкості і є по своїй суті паралельними [6-7,19-20], оскільки дозволяють отримувати розв'язок одночасно у декількох точках сітки інтегрування.

3.1 Загальна характеристика блокових/багатоточкових методів

У випадку блокових методів розв'язання задачі Коші множина точок рівномірної сітки, отриманої з кроком інтегрування h:

$$\Omega_h : \{ x_j \}, j = \overline{1,M}, N \le M \tag{3.1}$$

розбивається на N блоків. Кожен блок містить k точок і при цьому $N \le M$. Передбачається, що в межах блоку всі точки рівновіддалені одна від одної:

$$x_{n,i} = x_{n,o} + ih, i = \overline{1,k}, \tag{3.2}$$

де $i -$ номер точки в блоці $i = \overline{1,k}$, $n -$ номер блоку $n = 1, \dots, N$;

$x_{n,i} -$ точка з номером i, що належить блоку n;

$x_{n,0} -$ початкова точка n-го блоку; $x_{n,k}$ – кінцева точка n-го блоку.

Множина точок n-го блоку з k точок позначається, як $T_n^{(k)}$.

При цьому має місце рівність: $x_{n,k} = x_{n+1,0}$ (рис. 3.1). Нехай $y_{n,0}$ є наближене значення розв'язку задачі Коші в точці $x_{n,0}$ – початковій точці оброблюваного блоку.

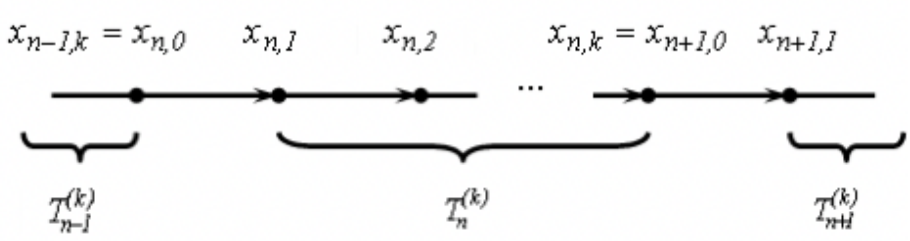


Рисунок 3.1 – Схема розбиття на блоки для однокрокового k-точкового методу

Рівняння однокрокових блокових різницевих методів у вживанні до ЗДР для блоку з k точок можуть бути записані таким чином:

$$y_{n,i} = y_{n,0} + ih\left[b_i F_{n,0} + \sum_{j=1}^{k} a_{i,j} F_{n,j} \right]; i = \overline{1,k}; n = \overline{1,N}. \tag{3.3}$$

Розкладом в ряд Тейлора функцій, що входять у нев'язку можна показати, що однокроковий k-точковий блоковий метод має найбільший порядок апроксимації, рівний $k+1$, отже, локальна похибка у вузлах блоку має порядок $O(h^{k+2})$ [22-25]. Блокові паралельні методи відносяться до класу неявних, тому для обчислення наближеного розв'язку задачі Коші необхідно вирішити систему, у загальному випадку нелінійних, рівнянь.

Одним із засобів одержання розв'язку є метод простої функціональної ітерації:

$$\begin{cases} y_{n,i,0} = y_{n,0} + ihF_{n,0}, i = \overline{1,k}, n = 1,2,...,N, \\ y_{n,i,l+1} = y_{n,0} + ih(b_i F_{n,0} + \sum_{j=1}^{k} a_{i,j} F_{n,j,l}), \quad l = \overline{0,L-1}, \end{cases} \tag{3.4}$$

де n – номер блоку, $n = 1,2,...,N$;

i – номер точки блоку, $i = \overline{1,k}$;

l – номер поточної ітерації $l = \overline{0,L-1}$;

L – максимальне число ненульових ітерацій.

На відміну від явних методів розв’язання СЗДР, реалізація альтернативних засобів оцінки апостеріорної локальної похибки на основі блокових методів пов'язана з рядом особливостей:

– немає відповідних послідовних аналогів, отже, потрібно розробити і обґрунтувати метод оцінки локальної похибки;

— зміна кроку інтегрування можлива лише після виконання обчислень у всіх k вузлах поточного n-го блоку

– при умові незадовільної оцінки локальної похибки практично всі обчислення для точок блоку виявляться даремними (деякі звернення до правої частини СЗДР можуть бути використані знову).

3.2 Ефективність паралельної реалізації правила дублювання кроку в блокових однокрокових методах інтегрування ЗДР

Нехай розв’язання задачі Коші для ЗДР виконується на основі k-точкового однокрокового блокового методу. При реалізації правила дублювання кроку необхідно провести обчислення за однією й тією ж групою формул, що мають такий вигляд (3.3):

$$\begin{cases} y_{n,i}^{(1)} = y_{n,0}^{(1)} + ih \cdot \left[b_i f(x_{n,0}; y_{n,0}^{(1)}) + \sum_{j=1}^{k} a_{i,j} f(x_{n,j}; y_{n,j}^{(1)}) \right], n = \overline{1,N},\ i = \overline{1,k} \\ y_{n,i}^{(2)} = y_{n,0}^{(2)} + i\frac{h}{2} \cdot \left[b_i f(x_{n,0}; y_{n,0}^{(2)}) + \sum_{j=1}^{k} a_{i,j} f(x_{n,j}; y_{n,j}^{(2)}) \right], n = \overline{1,N1}, i = \overline{1,k} \end{cases} \quad (3.5)$$

на двох різних рівномірних сітках:

1) $\Omega_h = \{ x_j \}, j = \overline{1,M}$ з кроком h у N блоках;

2) $\Omega_{h/2} = \{ x_j \}, j = \overline{1,M1}$ з половинним кроком у $N1$ блоках.

На рисунку 3.2 приведена схема обчислень при використанні правила Рунге для однокрокового блокового k-точкового методу. Як і у попередньому розділі, апроксимація розв’язку з одинарним кроком позначається $y_{n,i}^{(1)}$, а з половинним, відповідно: $y_{n,i}^{(2)}$.

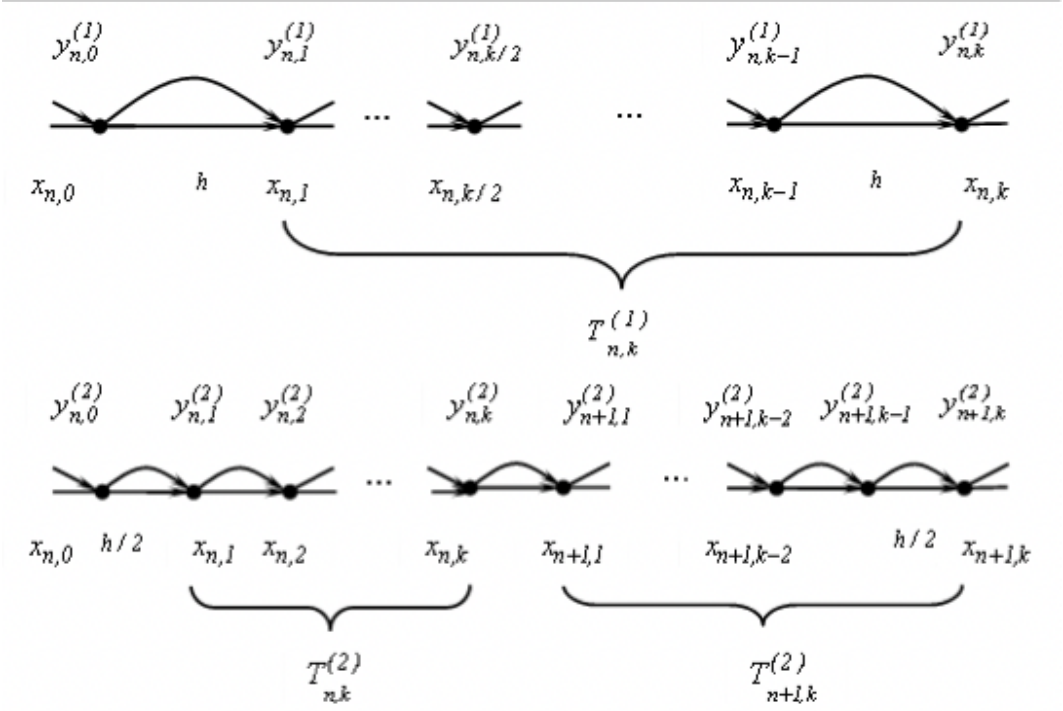


Рисунок 3.2 – Схема використання правила дублювання кроку для однокрокового блокового k-точкового паралельного методу

Точки n-го блоку сітки Ω_h складають множину $T^{(1)}_{n,k}$, а сітки $\Omega_{h/2}$ – $T^{(2)}_{n,k}$. Оскільки кількість точок у блоці для обох сіток дорівнює k, то для одного й того ж інтервалу інтегрування кількість блоків другої сітки точно у два рази більше, ніж для першої. Основою розрахунку при інтегруванні є сітка Ω_h, при цьому кожен вузол із парним номером у блоках сітки $\Omega_{h/2}$ використовується для обчислення оцінки локальної похибки на цьому кроці. Більш того, як розв'язок у цих вузлах часто приймається апроксимація, отримана з половинним кроком або екстрапольована, як найбільш точна. Вузли сітки $\Omega_{h/2}$ із непарними номерами використовуються лише як допоміжні. Оскільки дані методи є неявними, вживання правила Рунге до блокових однокрокових методів вимагає розв'язання трьох різних систем нелінійних алгебраїчних рівнянь розміру k.

Загальний час послідовної реалізації блокових методів з правилом Рунге T_1^{11} складається з суми часу обчислення розв'язку з одинарним кроком у блоці n плюс час розв'язку з половинним кроком в n-тому і $(n+1)$-ому блоках. Оскільки для здобуття кожного з трьох розв'язків реалізується свій ітераційний процес, введемо наступні позначення.

Нехай $L1-$ гранична кількість ітерацій для знаходження апроксимації розв'язку $y_{n,i}^{(1)}$, $L2-$ для $y_{n,i}^{(2)}$ и $L3-$ для $y_{n+1,i}^{(2)}$. Тоді, відповідно, поточну кількість ітерацій, що забезпечує достатню для кожної з даних задач точність, позначимо: $li,\ li \le Li,\ i = \overline{1,3}$.

Час обчислень для послідовного алгоритму блокових методів з правилом Рунге включає час на визначення нульових, а також подальших ітерацій вирішення:

$$T_1^{11} = \underbrace{6kt_{mul} + 3kt_{ad} + 2T_F}_{y_{n,i;0}^{(1)} + y_{n,i;0}^{(2)} + y_{n+1,i;0}^{(2)}} + (l1+l2+l3)\cdot \underbrace{[(k^2+2k)\cdot t_{mul} + (k^2+2k)\cdot t_{ad} + k\cdot T_F]}_{y_{n,i;l1}^{(1)} + y_{n,i;l2}^{(2)} + y_{n+1,i;l3}^{(2)}},$$

$$T_1^{11} = [k(l1+l2+l3)+2]\cdot T_F + [(l1+l2+l3)(2k^2+4k)+9k]\cdot t_{op} \qquad (3.6)$$

де T_F – час обчислень для правої частини ЗДР.

Обчислювальна схема паралельного блокового k-точкового методу з контролем локальної апостеріорної похибки за правилом Рунге приведена на рисунку 3.3. Тут кожен процесор обчислює розв'язок в одному вузлі сітки, тобто максимальний ступінь паралелізму обчислювальної схеми складає: $Dop = k$. Для кожної із трьох задач послідовно виконуються обчислення нульової і подальших ітерацій.

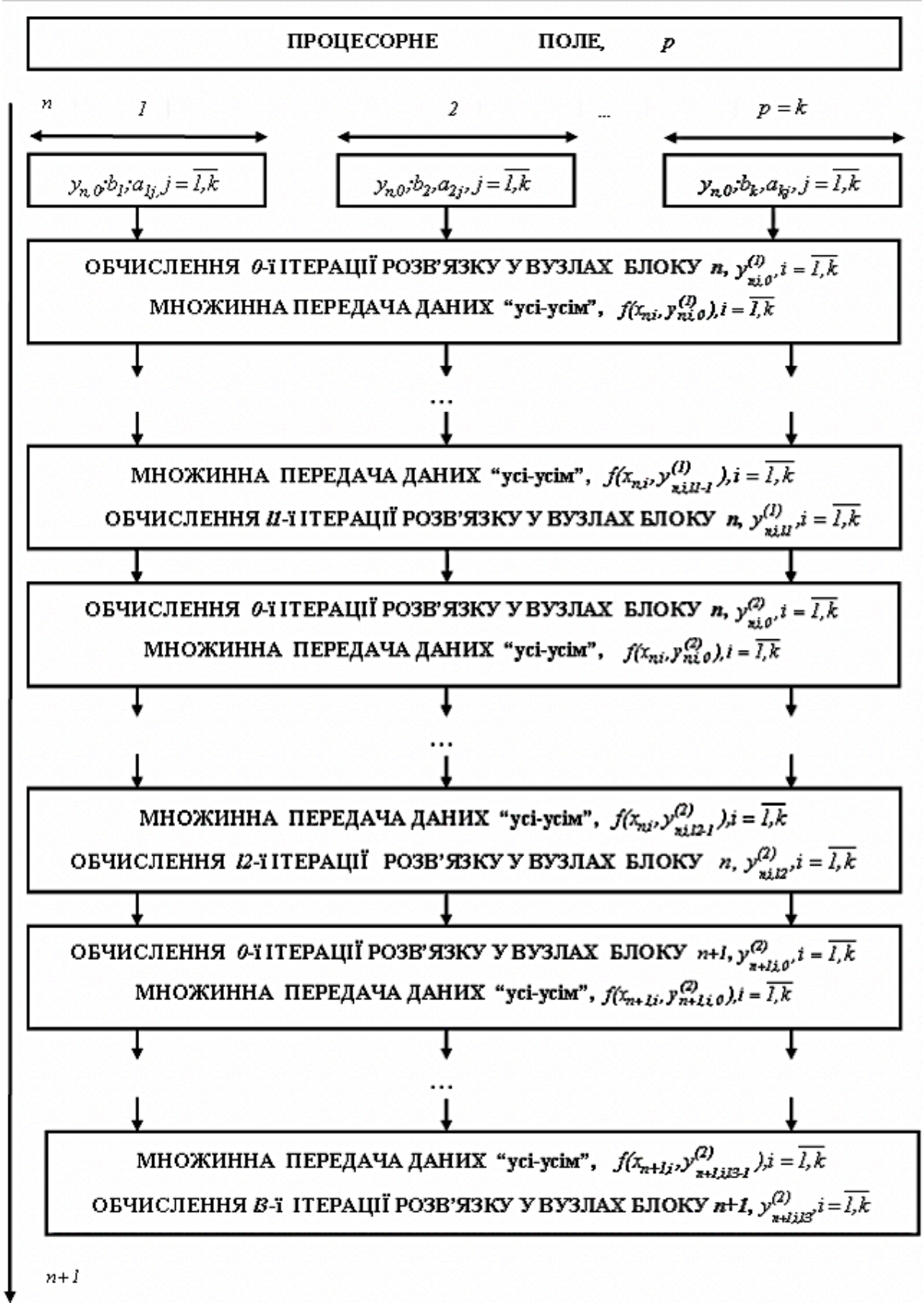


Рисунок 3.3 – Обчислювальна схема паралельного алгоритму блокового методу з контролем локальної похибки за правилом Рунге

При цьому нульова ітерація складається з наступних кроків: обчислення нульового наближення паралельно у кожному вузлі нового блоку за першою формулою (3.4); обчислення правої частини ЗДР від нульового наближення; множинний обмін обчисленими значеннями правої частини за типом “усі-усім”. Потім li разів виконується аналогічна група операцій для подальших ітерацій:

1) обчислення чергового наближення у кожному вузлі нового блоку за другою формулою (3.4), базовою операцією є множення матриці A на вектор значень правих частин ЗДР;

2) обчислення правої частини ЗДР від отриманого наближення і множинний обмін значеннями правої частини за типом “усі-усім”.

Таким чином, час паралельних обчислень за схемою (3.5) з локальною точністю $O(h^{k+li+2})$ у вузлах відповідних сіток складає:

$$T_{p,comp}^{11} = (\sum_{i=1}^{3} li + 2) \cdot T_F + [\sum_{i=1}^{3} li \cdot (k+3) + 3] \cdot t_{mul} + [\sum_{i=1}^{3} li \cdot (k+2) + 1] \cdot t_{ad}.$$

Для RISC архітектури, відповідно:

$$T_{p,comp}^{11} = (\sum_{i=1}^{3} li + 2) \cdot T_F + [\sum_{i=1}^{3} li \cdot (2k+5) + 4] \cdot t_{op}. \quad (3.7)$$

Для реалізації обмінів буде потрібно виконання групових операцій пересилок за типом “усі-усім”:

$$T_{p,comp}^{11} = \left(\sum_{i=1}^{3} li + 2 \right) \cdot T_{all-to-all}(p). \quad (3.8)$$

Потенційні характеристики паралелізму запропонованого методу можна оцінити по числу звернень до правої частини ЗДР.

При $T_F \gg t_{op}$:

$$S_{pot}^{11} \approx \left[\left(k \sum_{i=1}^{3} li + 2 \right) \cdot T_F \right] / \left[\left(\sum_{i=1}^{3} li + 2 \right) \cdot T_F \right] \approx k, \; E_{pot}^{11} \approx \frac{S_{pot}^{11}}{p} \approx 1,$$

тобто має місце практично лінійне прискорення і одинична ефективність. Такі ж потенційні характеристики можуть бути отримані і у разі, коли права

частина за часом обчислення сумірна з часом виконання однієї операції з рухомою точкою.

Реальні динамічні характеристики отриманого паралельного алгоритму істотно залежать не лише від параметрів задачі і алгоритму, але й від ефективності організації міжпроцесорних зв'язків та дорівнюють:

$$S^{11} = \frac{(k \cdot \sum_{i=1}^{3} li + 2) \cdot T_F + [\sum_{i=1}^{3} li \cdot (2k^2 + 4k) + 5k] \cdot t_{op}}{(\sum_{i=1}^{3} li + 2) \cdot T_F + [\sum_{i=1}^{3} li \cdot (2k + 5) + 4] \cdot t_{op} + \left(\sum_{i=1}^{3} li + 2\right) \cdot T_{all-to-all}(p)}, \quad (3.9)$$

$$E^{11} = \frac{(k \cdot \sum_{i=1}^{3} li + 2) \cdot T_F + [\sum_{i=1}^{3} li \cdot (2k^2 + 4k) + 5k] \cdot t_{op}}{k(\sum_{i=1}^{3} li + 2) \cdot T_F + [\sum_{i=1}^{3} li \cdot (2k + 5) + 4] \cdot t_{op} + \left(\sum_{i=1}^{3} li + 2\right) \cdot T_{all-to-all}(p)}. \quad (3.10)$$

Аналіз теоретичного виконання й обчислювальний експеримент показують, що для виконання групових операцій обміну в запропонованому алгоритмі ефективними є топології гіперкуб та тор (рис. 3.4), гірший варіант з'єднання процесорів – кільце.

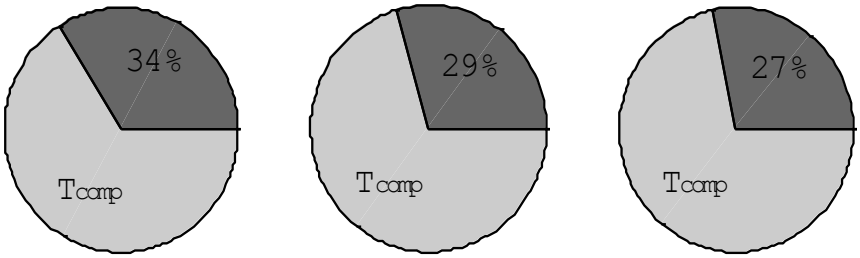
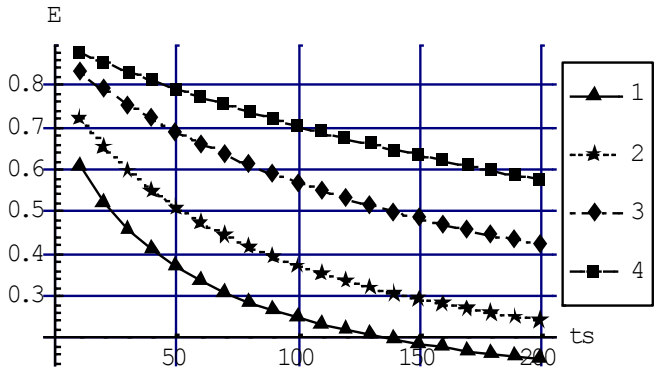


Рисунок 3.4 – Доля обмінів до спільного часу виконання блокового методу з правилом Рунге для різних топологій: кільце, 2D-тор та гіперкуб

Окрім топології з'єднання, на величину часу міжпроцесорних обмінів і динамічні характеристики паралелізму істотний вплив мають тип паралельної архітектури та визначені ним машинно-залежні константи обміну, такі, як латентність та час передачі одного слова (рис. 3.5).

Відмітимо, що із зростанням величини латентності комунікаційного середовища час на реалізацію обмінів збільшується, а прискорення й ефективність алгоритму зменшуються $\uparrow t_s \Rightarrow \uparrow T_{p,comm} \Rightarrow \downarrow S \Rightarrow \downarrow E.$

Аналогічні залежності зв'язують динамічні характеристики та час передачі одного слова: $\uparrow t_w \Rightarrow \uparrow T_{p,comm} \Rightarrow \downarrow S \Rightarrow \downarrow E.$ Проте величина латентності є найбільш істотним параметром, ступінь впливу швидкості передачі даних, як правило, зростає при збільшенні обсягів переданих даних.



$1 - T_F = 100t_{op}$; $2 - T_F = 200t_{op}$; $3 - T_F = 500t_{op}$;
$4 - T_F = 1000t_{op}$

Рисунок 3.5 – Залежність коефіцієнта ефективності блокового методу з правилом Рунге від значення латентності

Для даного алгоритму обчислення є однорідними, і, як результат, коефіцієнт ефективності для SIMD-архітектур за інших рівних умов вищий, ніж для MIMD-систем за рахунок меншого значення латентності мережі (рис. 3.6).

З тимчасових характеристик алгоритму та початкової задачі якість паралелізму найбільш істотно залежить від необхідного обсягу обчислень на реалізацію правої частини (3.1) й кількості точок в одному блоці.

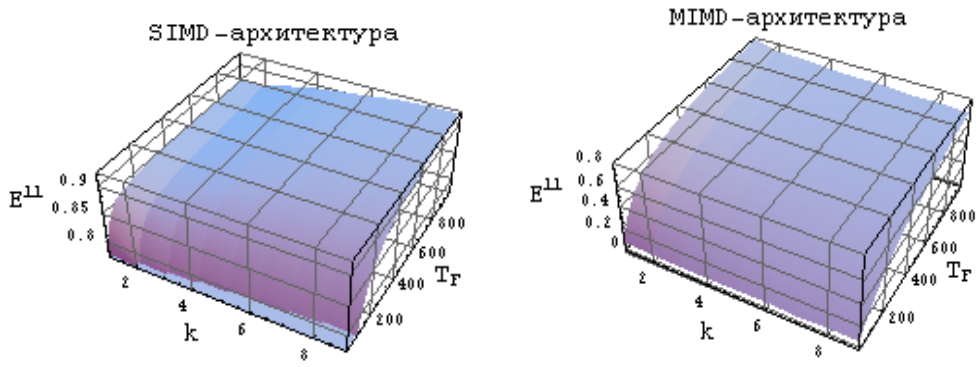
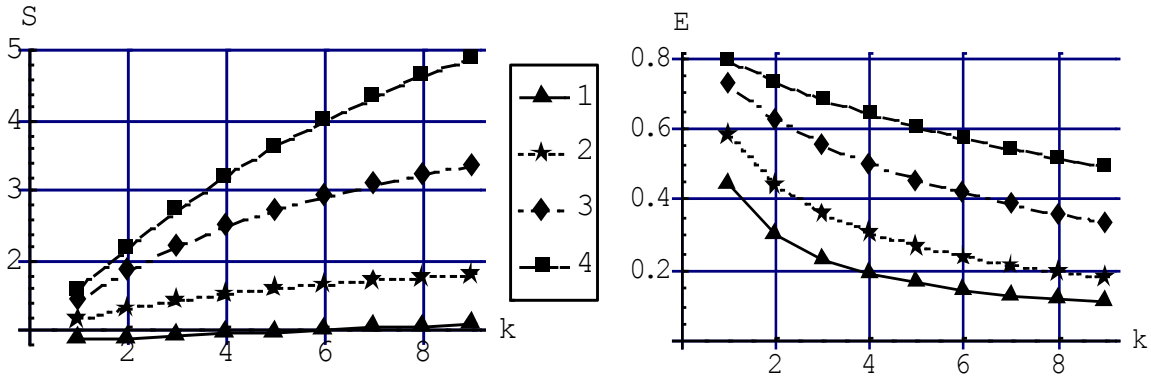


Рисунок 3.6 – Залежність коефіцієнтів ефективності блокових методів для ЗДР з правилом Рунге від кількості точок у блоці, k та складності правої частини, T_F

Залежності реальних коефіцієнтів прискорення і ефективності паралельного процесу контролю локальної похибки на основі правила Рунге від кількості точок блоку при зростанні складності правих частин ЗДР представлені за допомогою рис. 3.7.

Очевидно, чим складніша права частина ЗДР, тим кращі характеристики паралелізму: $\uparrow T_F \Rightarrow \uparrow S \Rightarrow \uparrow E$ і, одночасно, чим більше розмір блоку, співпадаючий з кількістю процесорів, тим більше прискорення та менша ефективність даного методу: $\uparrow k \Rightarrow \uparrow p \Rightarrow \uparrow S \Rightarrow \downarrow E$.



$1 - T_F = 100t_{op}$; $2 - T_F = 200t_{op}$; $3 - T_F = 500t_{op}$; $4 - T_F = 1000t_{op}$

Рисунок 3.7 – Коефіцієнти прискорення та ефективності блокового методу з правилом Рунге, $p = k$

Таким чином, найкращі характеристики паралелізму при розв'язанні нелінійної задачі Коші для одного рівняння блоковими методами з контролем локальної похибки за правилом Рунге досягаються для будь-якої паралельної архітектури за умовами:

1) великого розміру задачі;

2) складної правої частини;

3) топології гіперкуб у високошвидкісних мережах передачі інформації.

3.3 Вкладені блокові паралельні однокрокові методи розв'язання задачі Коші

Ідея вкладених форм, що запропонована для оцінки локальної похибки чисельного розв'язання звичайних диференційних рівнянь методами Рунге-Кути, може бути використана і для однокрокових блокових багатоточкових методів. У даному підрозділі приведено обґрунтування цього способу оцінки похибки для паралельних вкладених блокових алгоритмів розв'язання нелінійної задачі Коші для ЗДР на основі двох різних підходів:

1) комбінації незалежних формул суміжних порядків точності розв'язку;

2) комбінації спеціально підібраних формул різних порядків на базі методу послідовного підвищення порядку точності.

Перший підхід полягає у використанні двох різних незалежних блокових методів суміжних порядків точності $r(\hat{r}), \hat{r} = r \pm 1$ на одній і тій же сітці інтегрування Ω_h:

$$\begin{cases} y_{n,i} = y_{n,0} + ih\left[b_i f(x_{n,0}; y_{n,0}) + \sum_{j=1}^{k} a_{i,j} f(x_{n,j}; y_{n,j}) \right]; i = \overline{1,k}; n = \overline{1,N}, \\ \hat{y}_{n,i} = \hat{y}_{n,0} + ih\left[\hat{b}_i f(x_{n,0}; \hat{y}_{n,0}) + \sum_{j=1}^{\hat{k}} \hat{a}_{i,j} f(x_{n,j}; \hat{y}_{n,j}) \right]; i = \overline{1,\hat{k}}; n = \overline{1,\hat{N}}. \end{cases} \quad (3.11)$$

При цьому перший метод визначає апроксимацію розв'язку на основі k-точкового однокрокового методу, а другий $\hat{k}$-точкового, також однокрокового, методу. Другий наближений розв'язок у співпадаючих вузлах блоків $T_{n,i}^{(k)}$ і $T_{n,j}^{(\hat{k})}$ сітки Ω_h використовується для оцінки апостеріорної локальної похибки (рис. 3.8).

Нехай для визначеності основним є блоковий метод нижчого порядку точності, тобто $\hat{k} = k + 1$.

Локальна похибка наближеного розв'язку за однокроковим *k*-точковим методом у *i*-тому вузлі блоку визначається наступною формулою:

$$y(x_{n,i}) - y_{n,i}^{r} = O(h^{k+2}); i = \overline{1,k},$$

і для *(k+1)*-точкового методу у тому ж вузлі дорівнює:

$$y(x_{n,i}) - y_{n,i}^{\hat{r}} = O(h^{k+3}); i = \overline{1,\hat{k}}.$$

З отриманих співвідношень виходить, що оцінка локальної похибки формули меншого порядку точності *k*-точкового методу, приблизно може бути обчислена, як:

$$y_{n,i}^{r} - y_{n,i}^{\hat{r}}; i = \overline{1,k}.$$

Такий підхід до оцінки локальної похибки є більш ефективним у порівнянні з правилом Рунге, оскільки при достатній простоті дозволяє зменшити обчислювальні витрати.

Так, для вживання правила дублювання кроку необхідно вирішити три системи нелінійних алгебраїчних рівнянь розмірності k, а для застосування вкладеного методу дві нелінійні алгебраїчні системи: одну тієї ж розмірності, другу розмірності $(k+1)$.

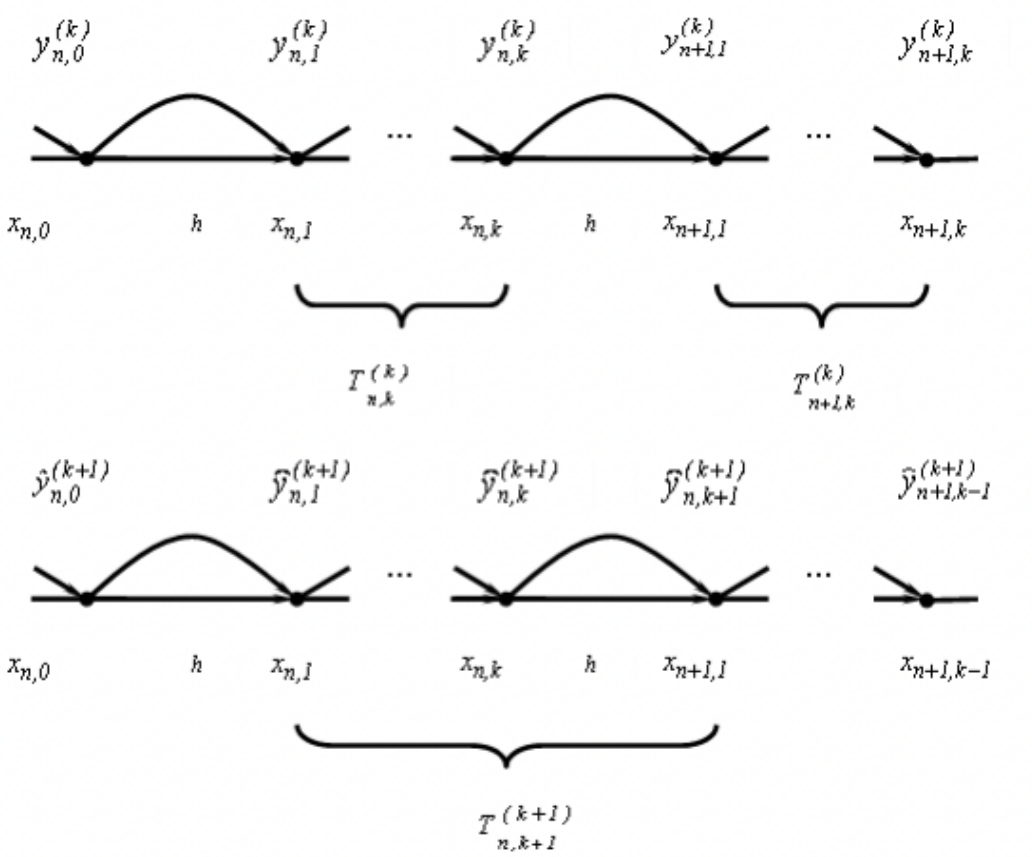


Рисунок 3.8 – Схема обчислення для вкладених однокрокових $k(\hat{k})$-точкових блокових методів, $\hat{k} = (k+1)$

Визначимо час послідовної реалізації даного методу в припущенні ідентичності обчислювальної та ємкісної потужності однопроцесорної та багатопроцесорної ОС. Нехай $T_1^{12}(k)$ – час послідовних обчислень за k-точковим методом, а $T_1^{12}(k+1)$ – аналогічний час, отриманий за $(k+1)$-точковим методом. Загальний час послідовного виконання схеми (3.12) дорівнює: $T_1^{12} = T_1^{12}(k) + T_1^{12}(k+1)$.

Кожен з блокових методів вимагає застосування ітераційного процесу для обчислення вирішення відповідних систем нелінійних рівнянь.

Нехай L – гранична кількість ітерацій для реалізації k-точкового методу, l – поточна кількість ітерацій, що забезпечує достатню для даної

задачі точність $l \le L$. Аналогічно, для вкладеного методу вищого порядку маємо наступні позначення: $\hat{L}$ і $\hat{l}$, $\hat{l} \le \hat{L}$.

Тоді, $T_1^{12}(k) = \underbrace{k[t_{mul} + t_{ad}] + T_F}_{y_{n,i,0}} + l \cdot \underbrace{[[k^2 + 2k] \cdot (t_{mul} + t_{ad}) + k \cdot T_F] + kt_{mul}}_{y_{n,i,l}}$,

$$T_1^{12}(\hat{k}) = \underbrace{\hat{k}[t_{mul} + t_{ad}] + T_F}_{\hat{y}_{n,i,0}} + \hat{l} \cdot \underbrace{\{[\hat{k}^2 + 2\hat{k}](t_{mul} + t_{ad}) + \hat{k}T_F] + \hat{k}t_{mul}}_{\hat{y}_{n,i,\hat{l}}}. \quad (3.12)$$

При $t_{mul} = t_{ad} = t_{op}$ та $\hat{k} = k + 1$ час виконання послідовного алгоритму:

$$T_1^{12} = (lk + \hat{l}k + \hat{l} + 2) \cdot T_F + [2l \cdot [(k^2 + 2k) + 2\hat{l}(k^2 + 4k + 3) + 6k + 3] \cdot t_{op}, \quad (3.13)$$

а з урахуванням співвідношення $\hat{l} = l + 1$, маємо:

$$T_1^{12} = (2lk + k + l + 3) \cdot T_F + [(4l + 2) \cdot k^2 + 12lk + 6l + 14k + 9] \cdot t_{op}. \quad (3.14)$$

Обчислювальна схема паралельного алгоритму №1 на основі першого підходу, схема описується формулами (3.11), представлена на рис. 3.9. Оскільки $\hat{k} = k + 1$, то кількість процесорів складає: $p = \hat{k}$ (при реалізації k-точкового методу один процесор простоює).

Кожен процесор обчислює розв'язок у одному вузлі сітки, тобто для такої обчислювальної схеми $Dop = \hat{k} = k + 1$. Спочатку визначається апроксимація розв'язку на основі k-точкового методу, потім – $(k+1)$-точкового.

Для кожної з двох вирішуваних задач послідовно виконуються обчислення нульової і подальших ітерацій. Тоді, час на обчислення й операції обміну паралельного вкладеного блокового методу №1 складає (при $\hat{l} = l + 1$ та $t_{op} = t_{mul} = t_{ad}$):

$$T_{p,comp}^{12} = (2l + 3) \cdot T_F + (4lk + 10l + 2k + 12) \cdot t_{op} \quad (3.15)$$

$$T_{p,comm}^{12} = (l + \hat{l} + 2) \cdot T_{all-to-all}(p) = (2l + 3) \cdot T_{all-to-all}(k + 1). \quad (3.16)$$

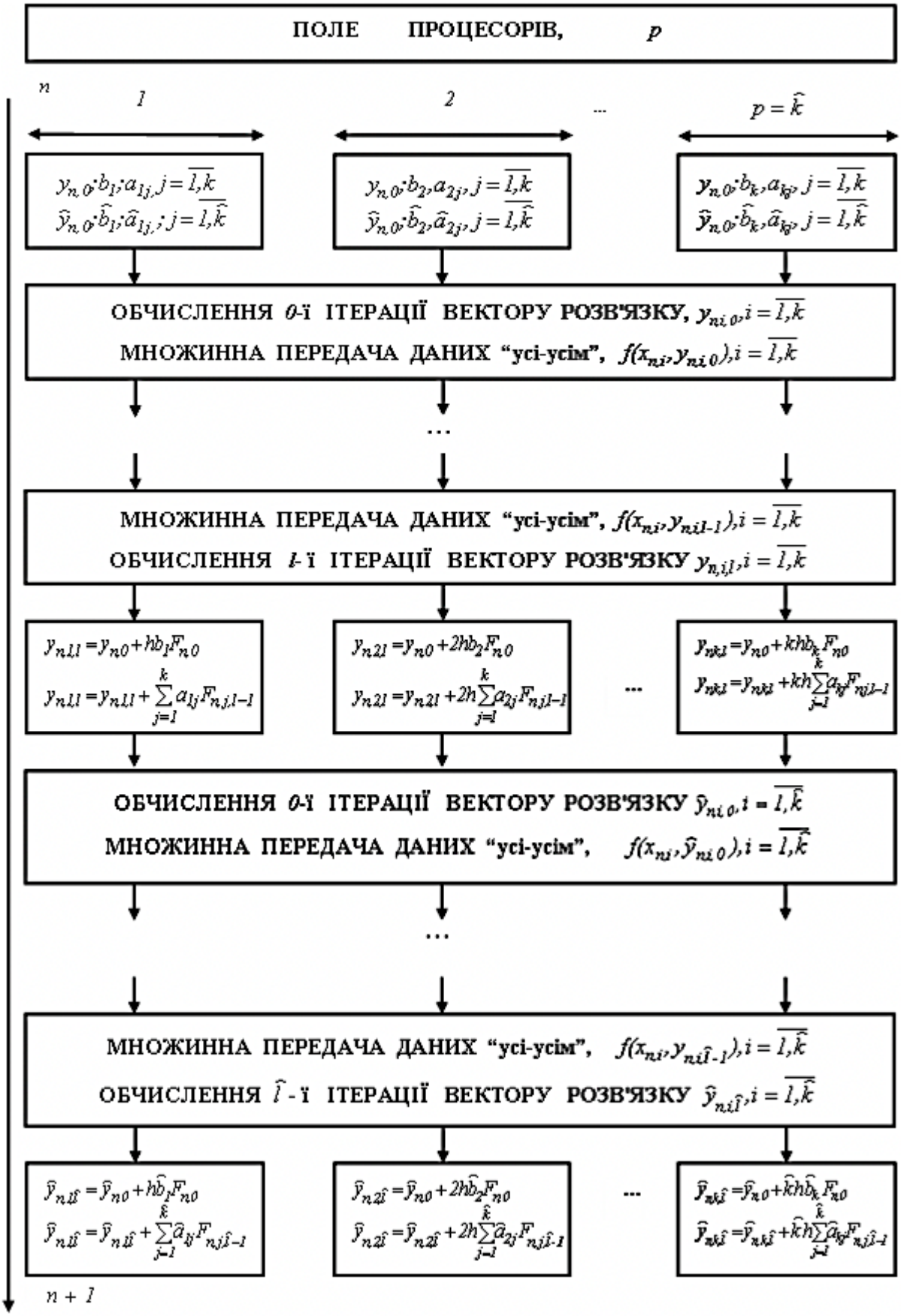


Рисунок 3.9 – Обчислювальна схема паралельного алгоритму №1

вкладеного однокрокового $k(\hat{k})$-точкового блокового методу для ЗДР

Потенційний паралелізм отриманого методу при складній правій частині ЗДР визначається співвідношенням:

$$S_{pot}^{12} = [k \cdot (l + \hat{l}) + \hat{l} + 2] \cdot T_F / [(l + \hat{l} + 2) \cdot T_F] \approx k, \; E_{pot}^{12} = S_{pot}^{12} / p \approx 1.$$

Аналогічні потенційні динамічні характеристики маємо й при $T_F \approx t_{op}$, тобто при тривіальній правій частині ЗДР.

Для вкладених методів якість або ступінь впливу топології з'єднання процесорів й параметрів комунікаційного середовища на динамічні характеристики паралелізму співпадають із результатами, отриманими при аналізі блокових методів з правилом Рунге.

Другий підхід до розробки блокових вкладених методів оснований на використанні ідеї послідовного підвищення порядку точності [118-119], і має своєю метою скорочення обчислювальних витрат за рахунок комбінації спеціально підібраних формул суміжних порядків.

Нехай розв'язання задачі Коші для звичайного диференційного рівняння на деякому інтервалі інтегрування виконується на основі k-точкового однокрокового блокового методу.

Покажемо, що у якості оцінки локальної похибки у кожному вузлі поточного n-го блоку може бути прийнята наступна величина:

$$d_{n,i} = \hat{y}_{n,i} - y_{n,i} = y_{n,i,\hat{l}} - y_{n,i,\breve{l}}, i = \overline{1,k}, \hat{l} = \breve{l} \pm 1, \qquad (3.17)$$

де $y_{n,i,\hat{l}}$ і $y_{n,i,\breve{l}}$ – $\hat{l}$ і $\breve{l}$-та апроксимації, які отримані при вирішенні (3.3) ітераційним методом (3.4).

Введемо наступні позначення:

1) $y_{n,i}^{[v]}, i = \overline{1,k}$ – значення чисельного розв'язку в i-тому вузлі $x_{n,i}$ n-го блоку, що обчислене з локальною похибкою $O(h^v)$,

2) $F_{n,i}^{[v]} = f\left(x_{n,i}, y_{n,i}^{[v]}\right)$ – звернення до правої частини початкового диференційного рівняння, яке обчислене в точці $\left(x_{n,i}, y_{n,i}^{[v]}\right)$.

Нехай наближене значення вирішення $y_{n,0}$ в початковій точці n-го блоку обчислене з деякою локальною помилкою $O(h^v)$, що забезпечує достатню для поставленої задачі точність. Тоді, звернення до правої частини початкового диференційного рівняння може бути обчислене з такою ж похибкою: $F_{n,0}^{[v]}$.

Виконавши обчислення для нульової ітерації за першою формулою ітераційного методу, отримаємо $y_{n,i,0}^{[2]} = y_{n,0}^{[v]} + ihF_{n,i,0}^{[v]}, i = \overline{1,k}$, оскільки локальна похибка формули Ейлера має порядок $O(h^2)$.

Кожне подальше обчислення за другою формулою (3.4) дає підвищення ладу точності для методу простих ітерацій на 1:

$$l = 0: \; y_{n,i,1}^{[3]} = y_{n,0}^{[v]} + ih \cdot \left(b_i F_{n,0}^{[v]} + \sum_{j=1}^{k} a_{i,j} F_{n,j,0}^{[2]} \right), i = \overline{1,k},$$

$$l = 1: \; y_{n,i,2}^{[4]} = y_{n,0}^{[v]} + ih \cdot \left(b_i F_{n,0}^{[v]} + \sum_{j=1}^{k} a_{i,j} F_{n,j,1}^{[3]} \right), i = \overline{1,k},$$

$$\ldots,$$

оскільки $F_{n,j,0}^{[v]} = f(x_{n,j}, y_{n,j,0}^{[v]}), j = \overline{1,k}$.

Цей процес не може бути продовжений нескінченно, при $l = k - 1$, отримаємо результати, відповідні граничній локальній точності наближених формул (3.4). Оскільки різницеві схеми, відповідні блоковим однокроковим k-точковим методом, апроксимують диференційне рівняння (3.1) з порядком $O(h^{k+1})$, то подальші ітерації підвищення порядку точності результатів не дають:

$$l = k - 1: y_{n,i,k}^{[k+2]} = y_{n,0}^{[v]} + ih \cdot \left(b_i F_{n,0}^{[v]} + \sum_{j=1}^{k} a_{i,j} F_{n,j,k-1}^{[k+1]} \right), i = \overline{1,k}. \qquad (3.18)$$

Таким чином, для оцінки локальної похибки можуть бути вибрані дві довільні суміжні апроксимації вирішення $y_{n,i,\breve{l}}$ і $y_{n,i,\hat{l}}$ із урахуванням міркувань точності і обмежень:

$\breve{l} < \hat{l} \le L - 1 = k - 1$.

Як правило, $\breve{l} = l,\ \hat{l} = l + 1$. При описаному вкладеному блоковому k-точковому методі, усі додаткові СНАР мають розмірність k.

Розрахункові формули для одного, n-го блоку вкладеного багатоточкового алгоритму №2 мають вигляд:

$$\begin{cases} y_{n,i,0} = y_{n,0} + ihF_{n,0},\ i = \overline{1,k}, \\ y_{n,i,l+1} = y_{n,0} + ih(b_i F_{n,0} + \sum\limits_{j=1}^{k} a_{i,j} F_{n,j,l}),\ \ l = \overline{0,\breve{l} - 1}, \\ \hat{y}_{n,i,\hat{l}} = y_{n,0} + ih(b_i F_{n,0} + \sum\limits_{j=1}^{k} a_{i,j} F_{n,j,\breve{l}}),\ \ \breve{l} = l, \hat{l} = l + 1. \end{cases} \tag{3.19}$$

Оцінка часу виконання послідовного алгоритму, що визначається обчислювальною схемою (3.19), дорівнює:

$$T_1^{13} = \underbrace{kt_{mul} + kt_{ad} + T_F}_{y_{n,i,0}} + \underbrace{\hat{l} \cdot [(k^2 + 2k) \cdot t_{mul} + (k^2 + 2k) \cdot t_{ad} + k \cdot T_f] + k \cdot t_{mul}}_{y_{n,i,\breve{l}} + \hat{y}_{n,i,\hat{l}}}.$$

При $t_{mul} = t_{ad} = t_{op}$ маємо:

$$T_1^{13} = (\hat{l}k + 1) \cdot T_F + [2\hat{l}(k^2 + 2k) + 3k] \cdot t_{op}. \tag{3.20}$$

Обчислювальна схема паралельного вкладеного блокового методу №2 приведена на рис. 3.10, як і раніше показано обчислення в рамках одного n-го блоку. Для простоти розглядується випадок, коли кількість процесорів збігається з розмірністю блоку і за кожним вузлом блоку закріплений один процесор.

Таким чином, для одного диференційного рівняння внутрішній паралелізм методу вичерпується незалежними обчисленнями кожної точки блоку й обмежений кількістю точок у блоці.

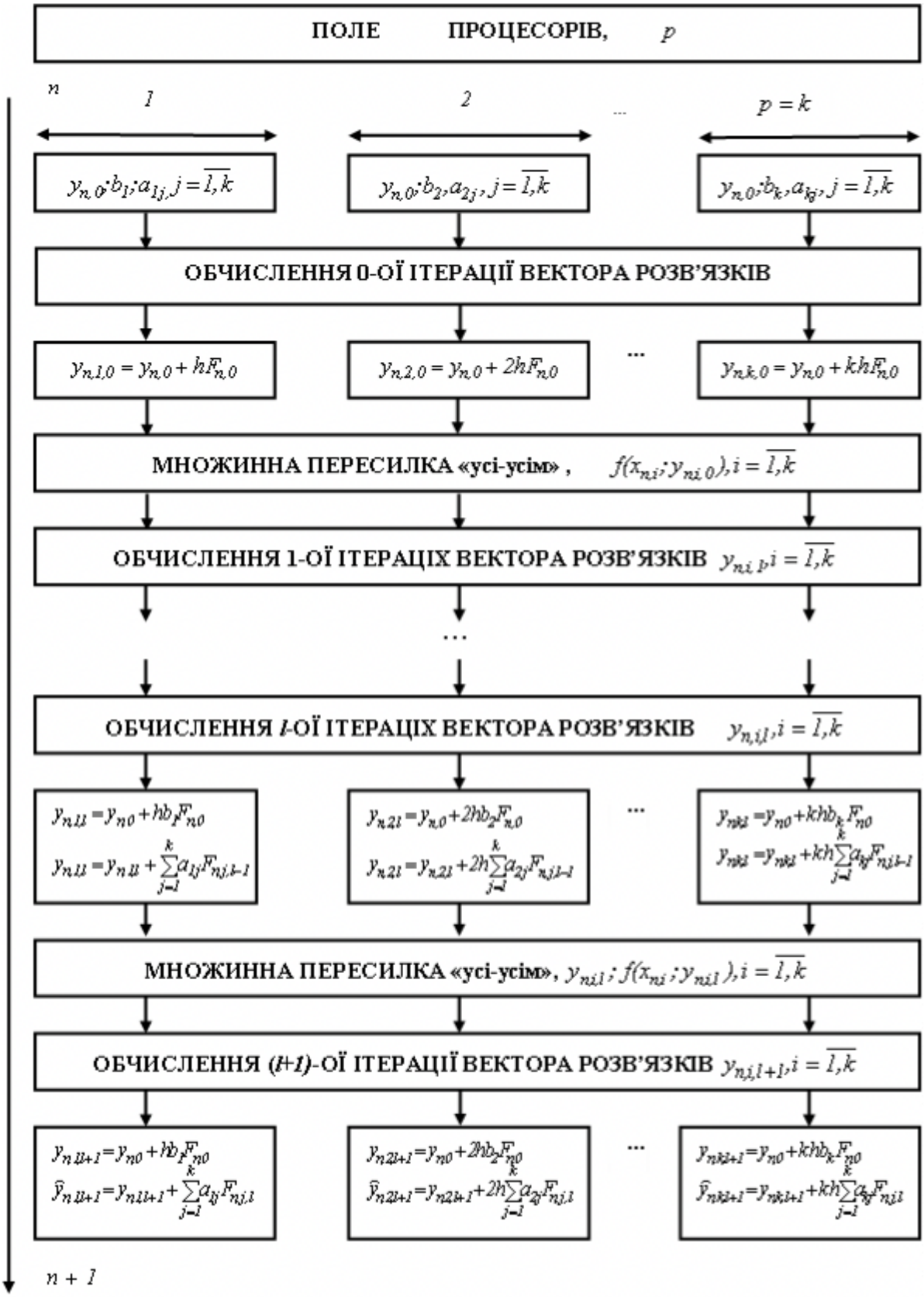


Рисунок 3.10 – Обчислювальна схема 3 паралельного алгоритму вкладеного однокрокового k -точкового блокового методу для ЗДР

Ітераційний процес розв'язання отриманих СНАР є суто послідовним. Після кожного з $l = \overline{0,\hat{l}}$ кроків необхідний обмін даними за типом "усі-усім". При реалізації алгоритму на k процесорах можна одночасно обчислювати значення $F_{n,i,l}$, а потім також одночасно набути по формулах значення $y_{n,i,l}$ для кожного фіксованого l. Час паралельного обчислення наближених значень розв'язку з точністю $O(h^{\hat{l}+2})$ для всіх вузлів блоку при виконанні $\hat{l}$ кроків ітераційного процесу дорівнює

$$T^{13}_{p,comp} = (\hat{l}+1)\cdot T_F + [\hat{l}(k+2)+2]\cdot t_{mul} + [\hat{l}(k+2)+1]\cdot t_{ad},$$

$$T^{13}_{p,comp} = (\hat{l}+1)\cdot T_F + [2\hat{l}k + 4\hat{l} + 3]\cdot t_{op}.$$

Час обміну даними при цьому з використанням уведених комунікаційних примітивів буде рівний:

$$T^{13}_{p,comm} = (\hat{l}+1)\cdot T_{all-to-all}(1,p) = (k+1)\cdot T_{all-to-all}(1,p).$$

Потенційно обидва вкладені блокові методи володіють високим ступенем внутрішнього паралелізму: практично лінійним прискоренням і одиничною ефективністю. Це витікає з приведених нижче співвідношень для коефіцієнтів потенційного прискорення і ефективності алгоритму у випадку, якщо час звернення до правої частини ЗДР домінує над іншими обчисленнями:

$$S^{13}_{pot} = T^{13}_1 / T^{13}_p \approx [(\hat{l}k+1)T_F] / [(\hat{l}+1)T_F] \approx k = p,\ E^{13}_{pot} = S^{13}_{pot} / p \approx 1.$$

Аналогічний результат маємо для тривіальної правої частини. Як і раніше найкращою топологією для розроблених методів є топологія гіперкуб. В цілому вплив параметрів комунікаційної середи на динамічні характеристики паралелізму цих двох алгоритмів збігається. Проведемо порівняння двох різних вкладених блокових методів (рис. 3.11-3.13) з метою визначити мінімум накладних витрат при оцінці локальної похибки розв'язання на основі ідеї вкладеності.

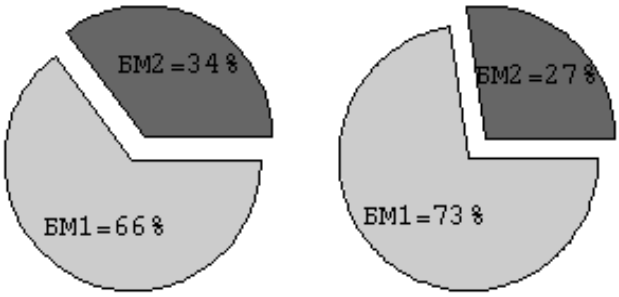


Рисунок 3.11 – Порівняння тимчасової складності двох вкладених блокових методів для ЗДР в послідовній та паралельній реалізаціях

Оскільки швидкість збіжності ітераційного процесу при вирішенні систем нелінійних алгебраїчних рівнянь істотно залежить від властивостей конкретної системи, а, отже, коефіцієнтів самого блокового методу, для спільності міркувань проведемо порівняння при гранично допустимій кількості ітерацій.

Аналіз теоретичного виконання й проведений експеримент дозволяють зробити наступні висновки:

1) час виконання першого вкладеного блокового алгоритму більше часу виконання другого, як у послідовній, так і в паралельній реалізаціях: $T_1^{12} > T_1^{13}$ і $T_p^{12} > T_p^{13}$, причому для паралельних алгоритмів ця різниця більша, ніж для послідовних;

2) коефіцієнти прискорення і ефективності першого вкладеного блокового методу менше відповідних коефіцієнтів другого метода при різних значеннях параметрів завдання, метода і паралельної системи: $S^{12} < S^{13}$, $E^{12} < E^{13}$ (рис. 3.12-3.13).

Відмітимо, що вплив чинника “складність правої частини ЗДР” є таким, що багато в чому визначає якість паралелізму. При однакових значеннях комунікаційних констант і параметрів, які задають унікальний блоковий метод, коефіцієнти прискорення і ефективності зменшуються практично у два рази при переході від домінуючих до тривіальних правих

частин. Ступінь впливу комунікаційних констант традиційний для даних методів і операцій обміну. Збільшення кількості точок у блоці призводить до зростання коефіцієнта прискорення $\uparrow k \Rightarrow \uparrow S$ та к зменшенню коефіцієнта ефективності $\uparrow k \Rightarrow \downarrow E$.

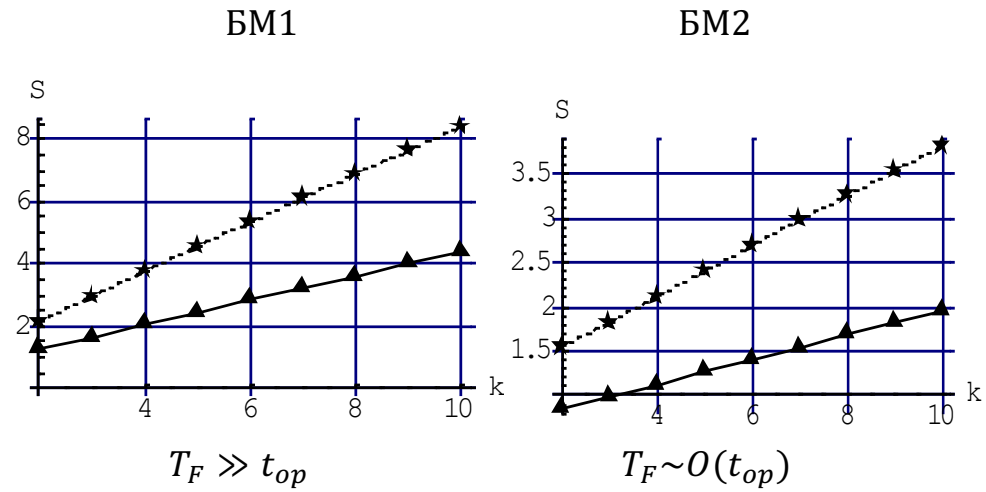


Рисунок 3.12 – Залежність коефіцієнта прискорення вкладених блокових методів від кількості точок блоку та складності правої частини ЗДР

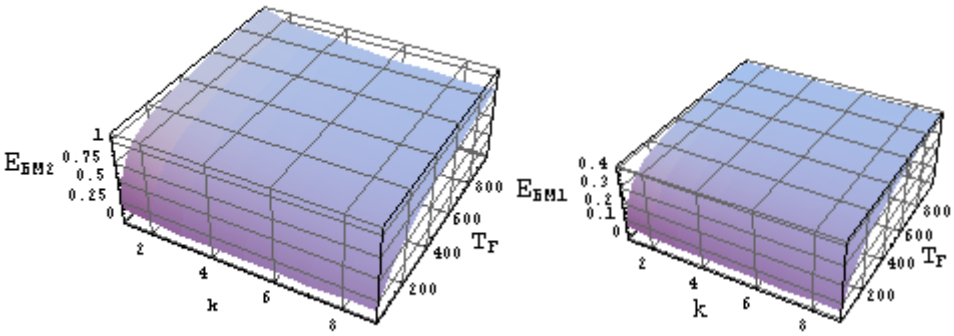


Рисунок 3.13 – Коефіцієнт ефективності вкладених блокових методів від числа точок блоку і складності правої частини ЗДР

Така залежність пояснюється тим, що число точок блоку пов'язане із кількістю використовуваних процесорів. Підсумовуючи всі отримані результати, можна зробити висновок, що з двох методів вкладених форм для

блокових однокрокових способів розв'язання нелінійної задачі Коші для ЗДР, другий метод володіє безперечними перевагами.

3.4 Технологія локальної екстраполяції на основі однокрокових блокових методів для ЗДР

Реалізація технології локальної екстраполяції Річардсона для блокових методів вимагає багатократних обчислень на одному й тому ж інтервалі інтегрування з використанням опорного блокового k_0-точкового методу порядку r_0 на рівномірних сітках, що згущуються:

а) $\Omega_{h/n_1} = \{ x_j \}, j = \overline{1, M_1}$ з кроком $h_1 = h / n_1$ в N_1 блоках;

б) $\Omega_{h/n_2} = \{ x_j \}, j = \overline{1, M_2}$ з кроком $h_2 = h / n_2$ в N_2 блоках;

.....

в) $\Omega_{h/n_k} = \{ x_j \}, j = \overline{1, M_k}$ з кроком $h_k = h / n_k$ в N_k блоках, де k – кількість рядків екстраполяційної таблиці.

Базовий крок інтегрування дорівнює $h_1 = h$, $n_1 = 1$, тобто основою рахунку є сітка: Ω_h. Для генерації допоміжних сіток вибирається гармонійний ряд $P_1 = \{ n_2, ..., n_k, ... \} = \{ 2,3,4,5,... \}$, як найменш витратний у разі опорного методу довільного типу (див. 2.20). Блоковий опорний метод повинен мати малий порядок точності:

$$T_{11,i} = y_{n,i} = y_{n,0} + ih \left[b_i F_{n,0} + \sum_{j=1}^{k_0} a_{i,j} F_{n,j} \right]; i = \overline{1, k_0}, k_0 \le 4, \qquad (3.21)$$

оскільки із зростанням r_0 обчислювальні витрати на технологію у цілому істотно зростають, не дивлячись на лінійне зменшення довжини екстраполяційної таблиці. Для неявних методів, до яких відносяться і дані блокові методи, це особливо важливо, оскільки збільшення порядку, а, отже, і кількості точок блоку, спричиняє за собою збільшення порядку багатократно вирішуваних СНАР.

Схема розбиття базового інтервалу на блоки при вживанні технології локальної екстраполяції для двоточкового опорного методу і гармонійної послідовності приведена на рис. 3.14. Відмітимо, що для блокових методів базовий інтервал інтегрування – це деякий блок із номером n основної сітки довжини $H = k_0 h$. Загальне число точок, що обчислюються з різними кроками інтегрування $h_i = h/i, i = \overline{1,k}$ і визначають вузли сіток Ω_{h_i} по P_1, дорівнює $M_i = H/h_i = (k_0 h)/h_i = k_0 i.$ Відповідно, кількість блоків $i-$ тої сітки $N_i = M_i / k_o = i$ і $n_i = N_i, i = \overline{1,k}$.

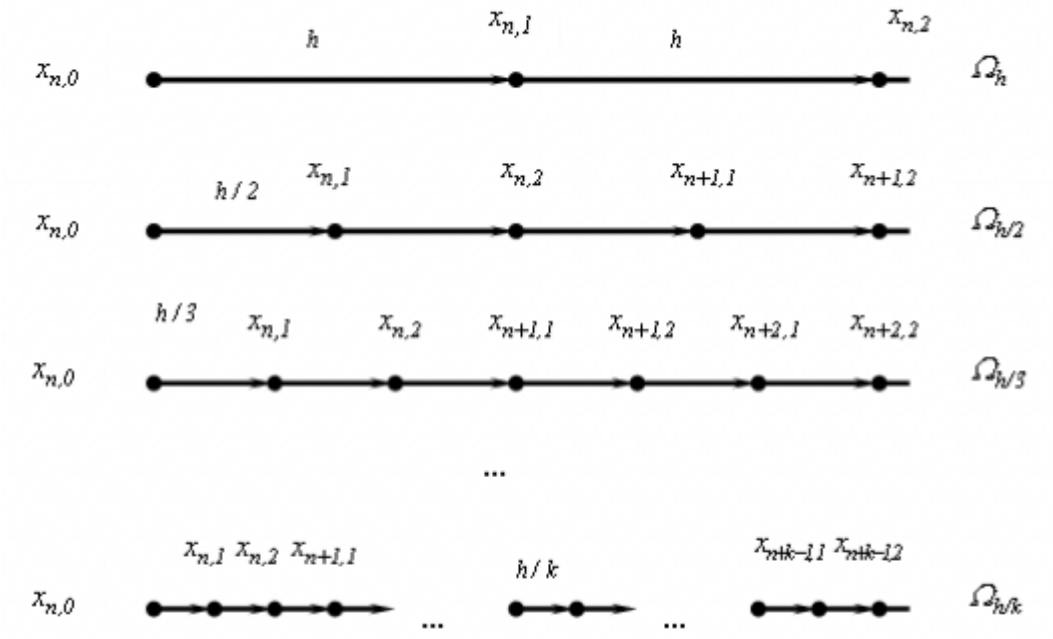


Рисунок 3.14 – Схема розбиття на блоки за технологією локальної екстраполяції для однокрокових блокових методів

Оскільки екстраполюються значення у вузлах базової сітки, необхідно встановити механізм відповідності вузлів сітки Ω_h основного рахунку вузлам сіток для екстраполяції $\Omega_{h_i}, i = \overline{2,k}$. Для двоточкового опорного блокового методу і гармонійного ряду маємо:

1) i - парне число: $x_{n,1}(h) \Leftrightarrow x_{n+\frac{i}{2}-1,2}(h_i)$;

2) i - непарне число: $x_{n,1}(h) \Leftrightarrow x_{n+\lceil\frac{i}{2}\rceil,1}(h_i)$;

3) для будь-яких i : $x_{n,2}(h) \Leftrightarrow x_{n+i-1,2}(h_i)$.

Значення першого стовпця екстраполяційної таблиці визначаються на основі наступних формул:

$$\begin{cases} T_{11,n,i} = y^{(1)}_{n,i} = y^{(1)}_{n,0} + ih \cdot \left[b_i F_{n,0} + \sum_{j=1}^{k_0} a_{i,j} F_{n,j} \right]; i = \overline{1,k_0}; n = \overline{1,N_1}, \\ \ldots \\ T_{k1,n,i} = y^{(k)}_{n,i} = y^{(k)}_{n,0} + i\frac{h}{k} \cdot \left[b_i F_{n,0} + \sum_{j=1}^{k_0} a_{i,j} F_{n,j} \right]; i = \overline{1,k_0}; n = \overline{1,N_k}. \end{cases} \quad (3.22)$$

Кожна з апроксимацій виходить за рахунок N_i разів застосованої схеми блокового k_0-точкового методу з різними кроками інтегрування:

$$\begin{cases} T_{T_{11}} = T_{\bar{y}^{(1)}_{n,i}(h)} = N_1 \cdot T_1^{r_0}(h), \\ \ldots \\ T_{T_{k1}} = T_{\bar{y}^{(k)}_{n,i}\left(\frac{h}{k}\right)} = N_k \cdot T_1^{r_0}(h/k), \end{cases} \quad (3.23)$$

де $T_1^{r_0}(h/i), i = \overline{1,k}$ – час, необхідний для вирішення задачі Коші для ЗДР опорним методом порядку r_0 з кроком h_i. Потім за формулою Ейткена-Невілла обчислюються наближення $T_{22}, T_{33,\ldots}$ і $T_{k,k}$. Час послідовних обчислень за схемою локальної екстраполяції складається з часу визначення k апроксимацій вирішення з різними кроками інтегрування і часу обчислення елементів екстраполяційної таблиці:

$$T_1^{14} = T_{T_{11}} + T_{T_{21}} + \ldots + T_{T_{k1}} + T_1^{ext-tab}, \quad T_1^{14} = T_1^{r_0} \cdot \sum_{i=1}^{k} n_i + T_1^{ext-tab}.$$

Для блокових опорних методів порядку $r_0 = k_0 + 1$ та P_1 маємо:

$$N_{P_1}(k) = \sum_{i=1}^{k} n_i = (k^2 + k)/2; \; k = r - r_0 + 1 \quad \Rightarrow$$

$$\Rightarrow \quad N_{P_1(k)}^{r_0} = [r^2 - r(2r_0 - 3) + (r_0 - 2)(r_0 - 1)] / 2 \quad \Rightarrow \tag{3.24}$$

$$N_{P_1(k)}^{r_0=3} = (r^2 - 3r + 2) / 2 .$$

Час обчислення елементів екстраполяційної таблиці дорівнює

$$T_1^{ext-tab} = \frac{(k^2 - k)}{2} \cdot (2t_{mul} + 3t_{ad}) = 5 \cdot \frac{r^2 - r(2r_0 - 1) + r_0(r_0 - 1)}{2} \cdot t_{op} \quad \Rightarrow \tag{3.27}$$

при $r_0 = 3$: $T_1^{ext-tab} = 2.5 \cdot (r^2 - 5r + 6) \cdot t_{op}$.

Нехай L_0 – гранична, максимальна кількість ітерацій, необхідна для розв'язання СНАР (3.24) ітераційним методом. Тоді час виконання послідовного алгоритму вирішення ЗДР на базі блокових однокрокових методів з вбудованим методом оцінки похибки за технологією локальної екстраполяції складає:

$$T_1^{14} = T_1^{r_0} \cdot \left(\frac{r^2 - 3r + 2}{2} \right) + 5 \left(\frac{r^2 - 5r + 6}{2} \right) \cdot t_{op}, \tag{3.25}$$

де $T_1^{r_0} = (L_0 k_0 + 1) \cdot T_F + [2L_0(k_0{}^2 + 2k_0) + 3k_0] \cdot t_{op}$,

при $L_0 = k_0$: $T_1^{r_0} = (k_0^2 + 1) \cdot T_F + [2k_0{}^3 + 4k_0{}^2 + 3k_0] \cdot t_{op}$,

$$T_1^{14} = (k_0^2 + 1) \left(\frac{r^2 - 3r + 2}{2} \right) \cdot T_F +$$

$$+ \left[(2k_0{}^3 + 4k_0{}^2 + 3k_0) \left(\frac{r^2 - 3r + 2}{2} \right) + 5 \left(\frac{r^2 - 5r + 6}{2} \right) \right] \cdot t_{op} . \tag{3.26}$$

Тоді, при $k_0 = 2 \quad \Rightarrow \quad L_0 = 2 \quad \Rightarrow r_0 = 3$ маємо: $T_1^{r_0=3} = 5T_F + 38t_{op}$.

Як результат: $T_1^{14} = (5T_F + 38t_{op}) \left(\frac{r^2 - 3r + 2}{2} \right) + 5 \left(\frac{r^2 - 5r + 6}{2} \right) \cdot t_{op}$,

$$T_1^{14} = 5 \cdot \left(\frac{r^2 - 3r + 2}{2} \right) \cdot T_F + \left(\frac{43r^2 - 89r + 106}{2} \right) \cdot t_{op} . \tag{3.27}$$

Для побудови паралельного алгоритму локальної екстраполяції на основі однокрокового блокового неявного методу (рис. 3.15) скористаємося результатами другої глави. Реалізуємо першу схему макрооперації, граф впливу якої приведений на рис. 2.20. Як основна макрооперація вибирається однократне обчислення деякої апроксимації розв'язку на основі k_0-точкового опорного методу малого порядку. Кожна макрооперація є ітераційним процесом, виконаним максимально можливим числом разів, тобто L_0. Потім формується остання частина екстраполяційної таблиці. Визначимо оцінки часу виконання паралельного алгоритму: $T_p^{14} = T_p^{r0} \cdot \sum_{i=1}^{k} n_i + T_p^{ext-tab}$.

Реалізація паралельних обчислень для опорного блокового методу потребує наступного часу обчислювальних і обмінних операцій:

$$T_{p,comp}^{r_0} = (L_0 + 1) \cdot T_F + [2L_0(k_0 + 2) + 3] \cdot t_{op}, \tag{3.28}$$

$$T_{p,comm}^{r_0} = (L_0 + 1) \cdot T_{all-to-all}(1, p). \tag{3.29}$$

Час паралельного виконання технології Річардсона складе:

$$T_p^{r_0} \cdot \sum_{i=1}^{k} n_i = \left(\frac{r^2 - r(2r_0 - 3) + (r_0 - 2)(r_0 - 1)}{2} \right) \cdot \left((L_0 + 1) \cdot T_F + [2L_0(k_0 + 2) + 3] \right) \cdot t_{op},$$

$$T_p^{ext-tab} = 5 \cdot \frac{r^2 - r(2r_0 - 1) + r_0(r_0 - 1)}{2} \cdot t_{op}.$$

Для двоточкового опорного методу:

$$T_{p,comp}^{14} = T_{p,comp}^{r_0=3} \cdot \sum_{i=1}^{k} n_i + T_{p,comp}^{ext-tab} = (3T_F + 19t_{op}) \cdot \left(\frac{r^2 - 3r + 2}{2} \right) + T_{p,comp}^{ext-tab}. \tag{3.30}$$

Відповідно, коефіцієнт потенційного прискорення алгоритму у випадку, якщо час звернення до правої частини ЗДР домінує над іншими обчисленнями, визначається таким чином:

$$S_{pot}^{14} = T_1^{14} / T_p^{14} \approx [(L_0 k_0 + 1) \cdot T_F] / [(L_0 + 1) \cdot T_F] \approx k_0 = p, \; E_{pot}^{14} = S_{pot}^{14} / p \approx 1.$$

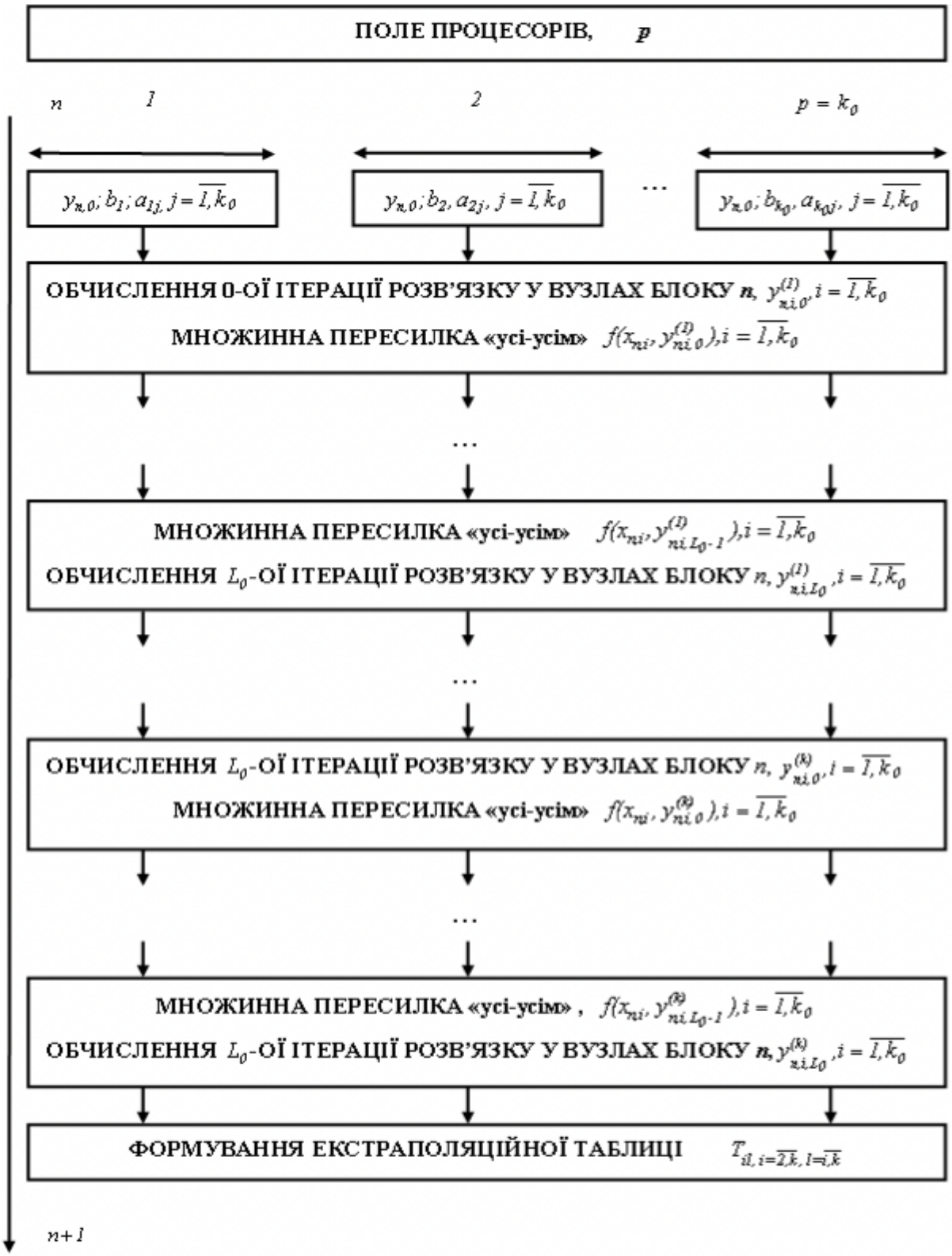


Рисунок 3.15 – Обчислювальна схема паралельного алгоритму технології локальної екстраполяції на основі однокрокового k_0-точкового блокового методу для ЗДР

Аналогічний результат маємо для тривіальної правої частини, тобто потенційно метод локальної екстраполяції для багатоточкових блокових методів володіє високим ступенем внутрішнього паралелізму. Обмінні операції розробленого паралельного алгоритму, потребують:

$$T_{p,comm}^{14} = \left(\frac{r^2 - r(2r_0 - 3) + (r_0 - 2)(r_0 - 1)}{2} \right) (L_0 + 1) \cdot T_{all-to-all}(p),$$

$$T_{p,comm}^{14} = \left(\frac{r^2 - 3r + 2}{2} \right) \cdot (L_0 + 1) \cdot T_{all-to-all}(p). \qquad (3.31)$$

Аналіз комунікаційної складової дає підстави стверджувати, що найкращою для запропонованого методу, як і раніше є топологія гіперкуб. Для порівняння паралельних алгоритмів різних засобів оцінки локальної апостеріорної похибки на основі багатоточкових методів оцінимо їх динамічні характеристики (рис. 3.16-3.17). Встановлено, що найбільш простими і найменш витратними є методи вкладених форм для систем будь-якої архітектури, різних параметрів задачі і методу. Це має особливе значення, оскільки, не дивлячись на трудомісткість, теоретично немає жодних обмежень для здобуття блокових методів вищих порядків.

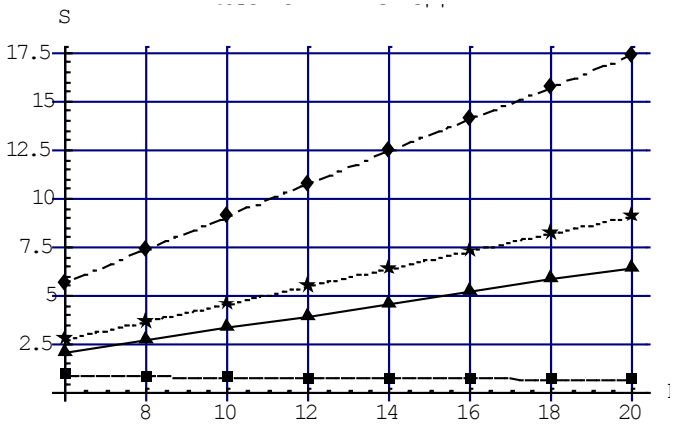


Рисунок 3.16 – Коефіцієнт прискорення блокових методів із різними засобами оцінки локальної похибки:
правило Рунге, БМ1,БМ2, ЛЕР,
$p = k$

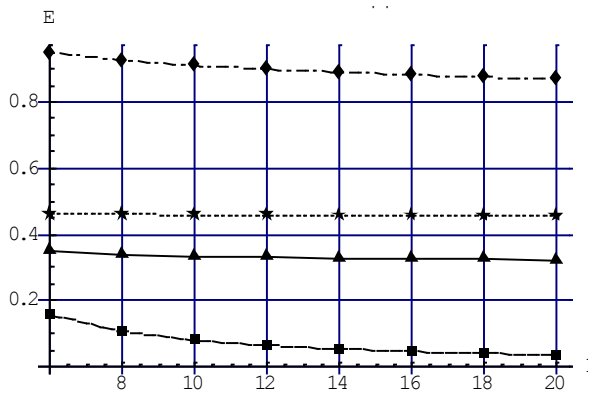


Рисунок 3.17 – Коефіцієнт ефективності блокових методів із різними засобами оцінки локальної похибки: правило Рунге, БМ1,БМ2, ЛЕР, $p = k$

Проте, вкладений блоковий метод на основі комбінації незалежних формул, як показали дослідження, має практично такий порядок складності, що і метод з використанням правила Рунге. Паралельний алгоритм технології локальної екстраполяції має гірші характеристики паралелізму, сферою його застосування, як і раніше залишаються високоточні розв’язки.

3.5 Особливості інтегрування систем звичайних диференційних рівнянь на базі вкладених блокових методів

Розроблені і обґрунтовані в попередніх підрозділах однокрокові блокові методи розв’язання задачі Коші для звичайного диференційного рівняння з оцінкою локальної похибки можна перенести на випадок системи рівнянь. При переході від одного рівняння до системи з'являється можливість використовувати системний паралелізм, який, як правило, значно більше паралелізму методу.

Блоковий однокроковий k-точковий метод для СЗДР має вигляд:

$$\bar{y}_{n,i} = \bar{y}_{n,0} + ih\left[b_i F_{n,0} + \sum_{j=1}^{k} a_{i,j} F_{n,j} \right]; i = \overline{1,k}; n = \overline{1,N}, \qquad (3.32)$$

та вимагає вирішення системи алгебраїчних нелінійних рівнянь розміру $m \times k$:

$$\begin{cases} y_{n,q,i;0} = y_{n,q,0} + ihf_q(x_{n,0}; \bar{y}_{n,0}), \quad i = \overline{1,k}, \ n = \overline{1,N}, \ q = \overline{1,m}, \\ y_{n,q,i;l+1} = y_{n,q,0} + ih[b_i f_q(x_{n,0}; \bar{y}_{n,0}) + \sum_{j=1}^{k} a_{i,j} \cdot f_q(x_{n,j}; \bar{y}_{n,j,l})], \quad l = \overline{0,L-1}, \end{cases} \tag{3.33}$$

де $F_{n,q,j} = f_q[x_{n,j}; y_1(x_{n,j}), y_2(x_{n,j}), ..., y_m(x_{n,j})], \ q = \overline{1,m}$ є q-та компонента вектору правої частини СЗДР.

Особливості паралельного інтегрування систем ЗДР розгледимо на основі вкладених блокових методів, як найбільш ефективних з існуючих засобів оцінки локальної апостеріорної похибки:

$$\begin{cases} y_{n,q,i;0} = y_{n,q,0} + ihF_{n,q,0}, \ i = \overline{1,k}, \\ y_{n,q,i} = y_{n,q,i;l+1} = y_{n,q,0} + ih(b_i F_{n,q,0} + \sum_{j=1}^{k} a_{i,j} F_{n,q,j;l}), \quad l = \overline{0,L-2}, \\ \hat{y}_{n,q,i} = y_{n,q,i;L} = y_{n,q,0} + ih(b_i F_{n,q,0} + \sum_{j=1}^{k} a_{i,j} F_{n,q,j;L-1}). \end{cases} \tag{3.34}$$

Послідовний алгоритм інтегрування за чисельною схемою має наступну обчислювальну складність:

$$T_1 = (Lk+1)\sum_{i=1}^{m} T_{f_i} + m[2L(k^2+2k)+3k]t_{op}. \tag{3.35}$$

Ітераційний процес, що описується обчислювальною схемою (3.34), може бути реалізований лише послідовно. Проте він дозволяє розподілити обчислення поточної l-тої ітерації векторів розв'язку наступними способами:

1) на k незалежних процесів за кількістю точок у блоці ($Dop_1 = k$);

2) на m за кількістю компонент у системі ЗДР($Dop_2 = m$);

3) на mk процесів за максимальною можливою шириною паралельного методу і системи ($Dop_3 = mk$).

Обчислювальні схеми паралельного вкладеного блокового методу для СЗДР, що використовують описані способи розпаралелювання, приведені на рисунках 3.18-3.20. Для скорочення запису введені наступні позначення.

Вектор розв'язків $y_{n,q,i;l}$ має чотири індекси, n – номер блоку, q – номер компоненти початкової СЗДР, i – номер точки у блоці, l – номер ітерації при розв'язанні відповідної СНАР.

Наявність прочерку для одного з індексів означає, що береться відповідний вектор значень. Так, $y_{n,\_,i;l}$ – позначає вектор розв'язків розміру m для СЗДР у i-й точці n-го блоку, обчислений на l-тій ітерації: $(y_{n,1,i;l}, y_{n,2,i;l}, \ldots, y_{n,m,i;l})$, а $y_{n,q,\_;l}$ – вектор розв'язку розмірності k для q-тої компоненти СЗДР, обчислений в усіх точках n-го блоку на l-тій ітерації: $(y_{n,q,1;l}, y_{n,q,2;l}, \ldots, y_{n,q,k;l})$.

Оцінимо якість отриманих паралельних алгоритмів і визначимо пріоритетні області використання кожного з них:

$$1)\ T^1_{p,comp} = (L+1)\cdot \sum_{i=1}^{m} T_{f_i} + m[2Lk + 4L + 3]\cdot t_{op}, \tag{3.36}$$

$$T^1_{p,comm} = (L+1)\cdot T_{all-to-all}(m,p) = (L+1)\cdot T_{all-to-all}(m,k), \tag{3.37}$$

$$2)\ T^2_{p,comp} = (Lk+1)T_{F_{max}} + [2L(k^2+2k)+3k]t_{op}, \tag{3.38}$$

$$T^2_{p,comm} = (L+1)\cdot T_{all-to-all}(k,p) = (L+1)\cdot T_{all-to-all}(k,m). \tag{3.39}$$

За першою схемою (рис. 3.18) кожен процесор обчислює всі компоненти вектору розв'язку деякої точки блоку на кожному кроці ітераційного процесу.

За другою схемою (рис. 3.19) кожен процесор обчислює одну компоненту вектору розв'язку для всіх точок блоку на кожному кроці ітераційного процесу.

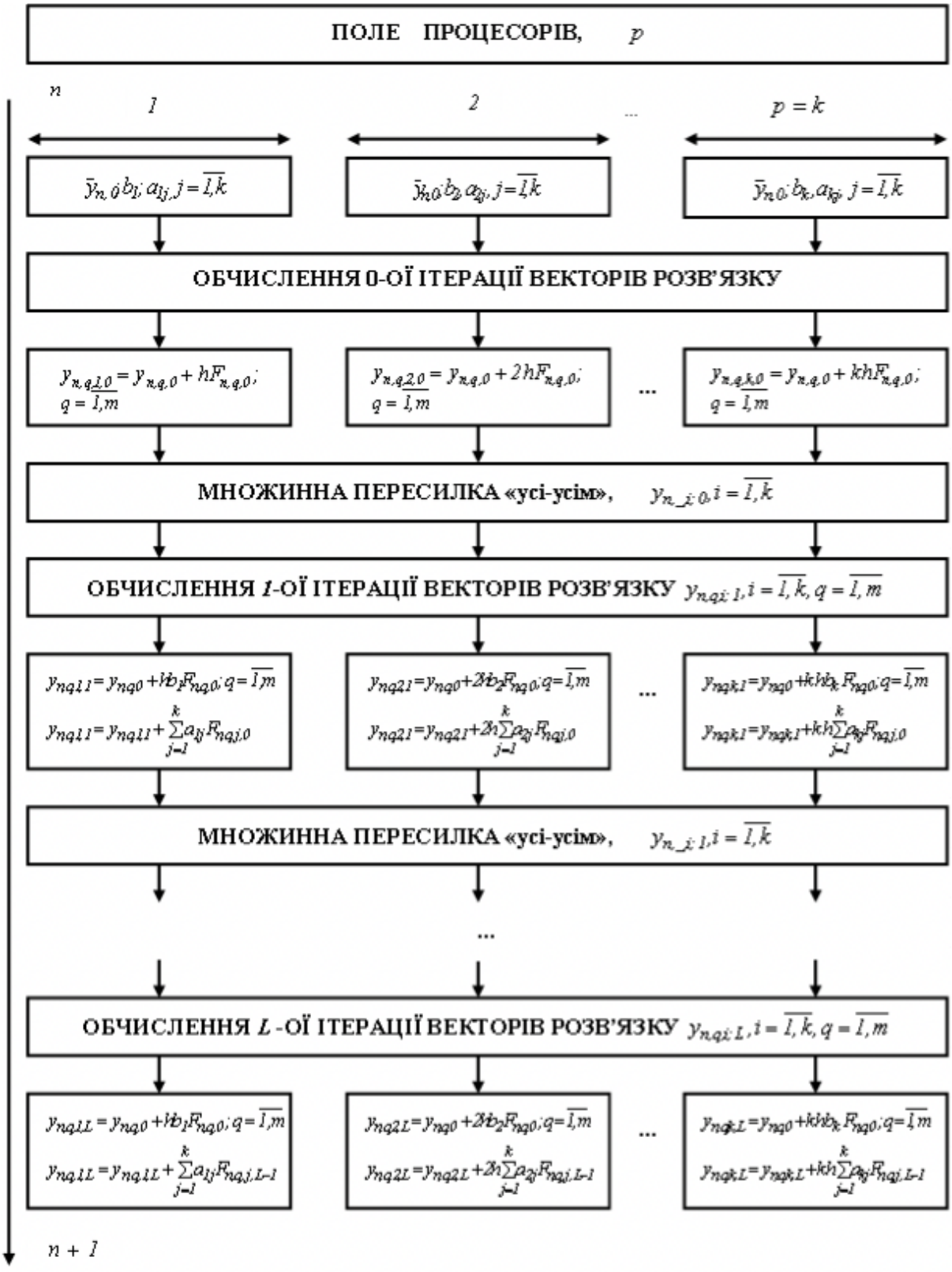


Рисунок 3.18 – Обчислювальна схема №1 паралельного алгоритму вкладеного однокрокового k -точкового блокового методу інтегрування систем звичайних диференційних рівнянь

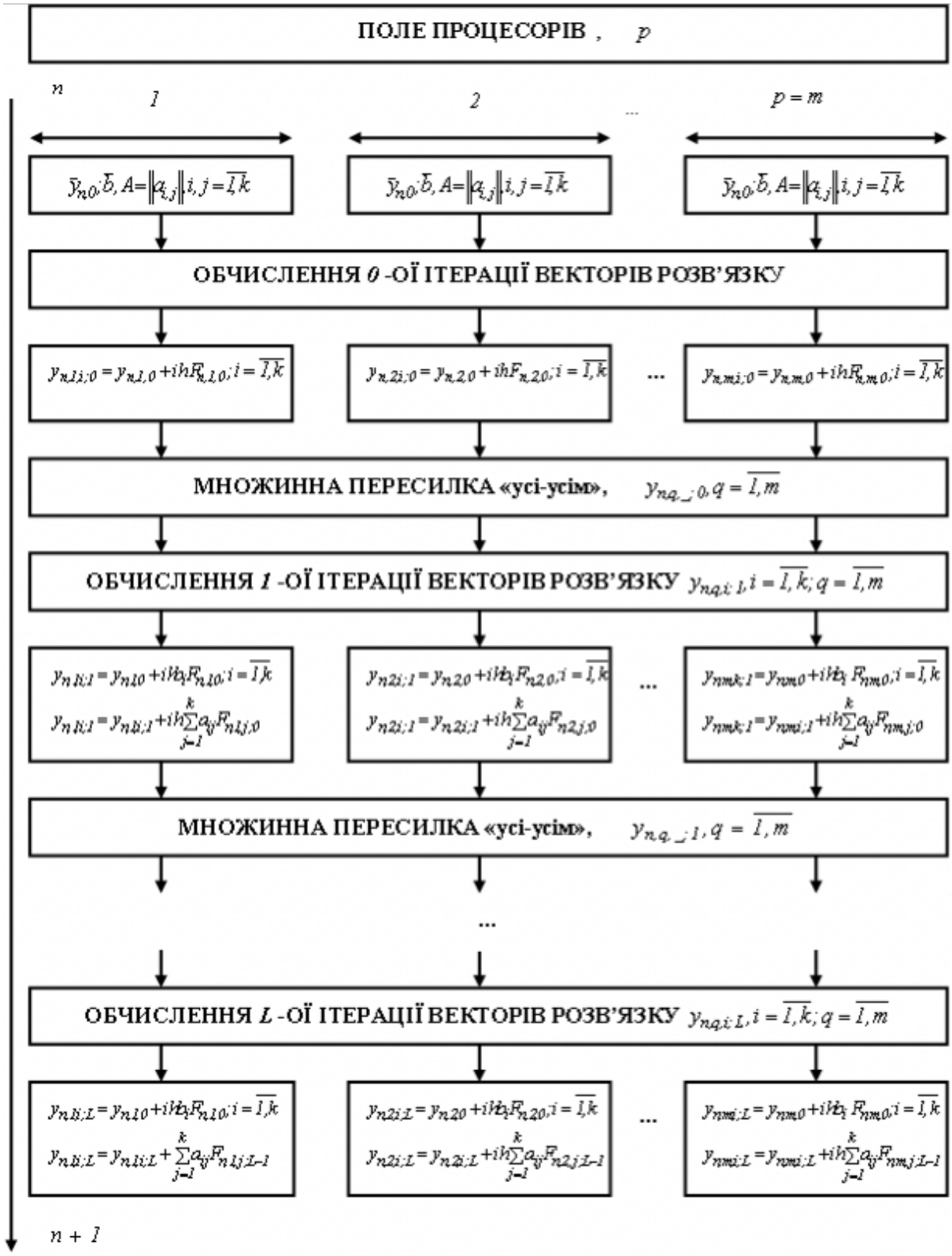


Рисунок 3.19 – Обчислювальна схема №2 паралельного алгоритму вкладеного однокрокового k -точкового блокового методу інтегрування СЗДР

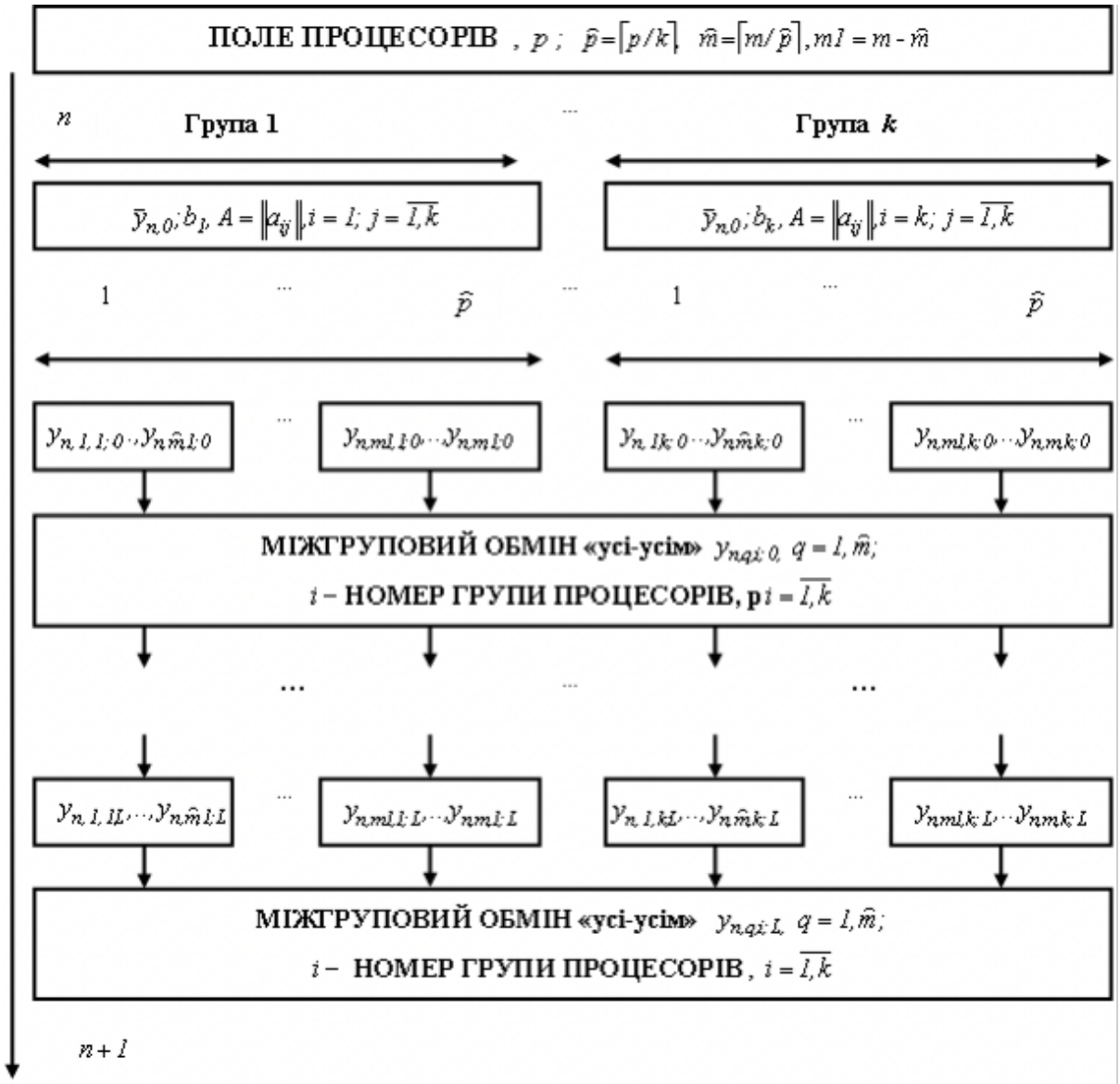


Рисунок 3.20 – Обчислювальна схема №3 паралельного алгоритму вкладеного однокрокового k-точкового блокового методу інтегрування СЗДР

Визначимо аналітичні вирази для розрахунку часу обміну між процесорами або комунікаційну складову із урахуванням якнайкращої для даного типу операцій топології гіперкуб:

$$T^1_{p,comm} = (L+1)\cdot[t_s \cdot log_2 k + m(k-1)\cdot t_w],$$

$$T^2_{p,comm} = (L+1)\cdot[t_s \cdot log_2 m + k(m-1)\cdot t_w].$$

Друга обчислювальна схема має час обчислень і загальний час виконання на багатопроцесорній ОС менші, ніж перша. Числові значення коефіцієнтів прискорення варіюються при зміні машинно-залежних комунікаційних констант та складності правої частини, але при цьому коефіцієнт прискорення обчислень для першої обчислювальної схеми завжди нижчий, ніж для другої.

Досліджені обчислювальні схеми блокового вкладеного методу вирішення СЗДР мають обмеження на кількість використаних процесорів. Розробимо паралельний алгоритм, що не містить таких обмежень і що враховує як паралелізм методу, так і системний (рис. 3.20).

Нехай є паралельна ОС із p процесорами; розіб'ємо процесорне поле на k груп по кількості точок у блоці. Кількість процесорів в одній групі дорівнює $\hat{p} = \lceil p/k \rceil$, кількість компонент системи, обчислюваних одним процесором, складає $\hat{m} = \lceil m/\hat{p} \rceil$. Кожна група відповідає за обчислення розв'язку в одній точці блоку для всієї СЗДР на одній ітерації. Час послідовної реалізації цього алгоритму обчислюється за формулами (3.38), час паралельної реалізації для топології гіперкуб визначається, як:

$$T^3_{p,comp} = (L+1) \cdot \sum_{i=1}^{\lceil mk/p \rceil} T_{f_i} + \lceil mk/p \rceil \cdot [2Lk + 4L + 3] \cdot t_{op}, \qquad (3.40)$$

$$T^3_{p,comm} = (L+1) \cdot [t_s \cdot \log_2 mk + \lceil mk/p \rceil (mk-1) \cdot t_w] \qquad (3.41)$$

Потенційні характеристики всіх трьох алгоритмів при довільних правих частинах близькі до ідеальних (лінійне прискорення і одинична ефективність) за умови, що кількість процесорів дорівнює максимальному ступеню паралелізму кожного методу.

Через різну складність обмінних операцій і максимально можливу кількість процесорів: перша обчислювальна схема, що використовує тільки паралелізм методу, має найгірші характеристики реального паралелізму.

Третя обчислювальна схема володіє безперечною перевагою перед двома іншими (рис. 3.21-3.22).

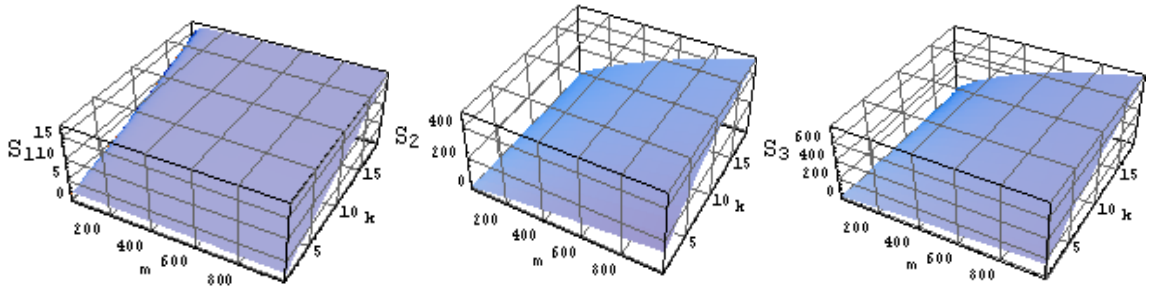


Рисунок 3.21– Коефіцієнт прискорення для паралельної реалізації трьох обчислювальних схем від кількості точок у блоці і розмірності задачі

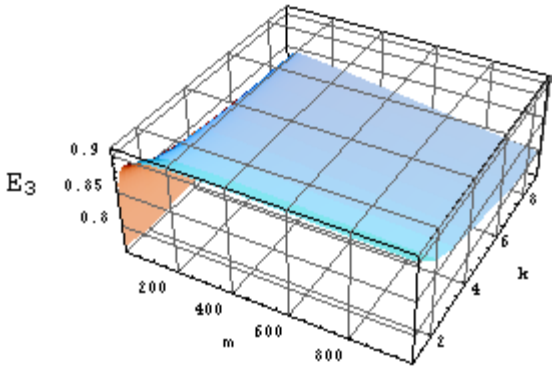


Рисунок 3.22– Коефіцієнт ефективності паралельної реалізації обчислювальної схеми №3

Вочевидь, що якість розроблених алгоритмів розв’язання СЗДР істотно залежить від декількох параметрів, в першу чергу від співвідношення розміру задачі, властивостей чисельного методу та паралельної багатопроцесорної системи. Таким чином, якнайкращі характеристики паралелізму при розв’язанні систем ЗДР блоковими вкладеними методами мають місце при виконанні декількох умов:

- багатовимірність задач,
- складність правих частин,

- наявність високошвидкісних мереж передачі даних із топологією гіперкуб,

- застосування обчислювальної схеми №3, що використовує як системний паралелізм, так і внутрішній паралелізм методу.

3.6 Порівняльний аналіз неявних однокрокових методів розв’язання задачі Коші для ЗДР

Аналіз ефективності визначення апостеріорної локальної похибки в даному розділі базується на вживанні наступних неявних однокрокових чисельних схем:

1) розпаралелених повністю неявних методів типу Рунге-Кутти;

2) паралельних блокових багатоточкових методів.

Проведемо порівняння двох класів неявних методів при послідовній і паралельній реалізаціях на основі найбільш ефективного способу оцінки локальної похибки, а, саме, вкладених формул. Основними параметрами, які характеризують ПНМРК, є взаємозв'язані величини – порядок методу r та кількість стадій s. Крім того, при вирішенні систем нелінійних алгебраїчних рівнянь для визначення стадійних коефіцієнтів з'являється такий параметр, як необхідна кількість ітерацій L.

Обчислювальна складність блокових методів залежить від порядку методу r, кількості точок у блоці k і, а також від кількості ітерацій $\hat{L}$ для здобуття розв’язку СНАР необхідної точності. Тимчасові витрати на обчислення, а також обмінні операції при паралельній реалізації описаних методів приведені в таблиці 3.1. Для того, щоб порівняння чисельних методів на базі неявних однокрокових схем було коректним, необхідно:

1) забезпечити один і той же порядок точності r;

2) отримати рішення в ідентичній кількості нових точок k;

3) породжувані ітераційні процеси повинні реалізовувати граничну кількість ітерацій, передбачену кожним з даних методів: $L = 2s$ і $\hat{L} = k$.

Для широко вживаних ПНМРК за теоремами Батчера кількість стадій пов'язана з порядком методу одним з наступних співвідношень [94]:

1) метод типу Гауса $r = 2s$;

2) методи Радо $r = 2s - 1$;

3) методи типу Лобатто $r = 2s - 2$.

Візьмемо співвідношення $r = 2s$, тим самим навмисно вибираючи кращий варіант для ПНМРК.

Таблиця 3.1 –Тимчасові характеристики s-стадійних ПНМРК та k-точкових блокових однокрокових методів

Методи		Коефіцієнт при T_F	Коефіцієнт при t_{op}	Коефіцієнт при $T_{all-to-all}$
ПНМРК (k - точок)	T_1	$ks(L+1)$	$2k(Ls^2 + 2s + 1)$	
	T_p	$k(L+1)$	$2k(Ls + s + 2)$	$k(L+1)$
Блокові методи	T_1	$\hat{L}k + 1$	$2\hat{L}(k^2 + 2k) + 3k$	
	T_p	$\hat{L} + 1$	$2\hat{L}k + 4k) + 3$	$\hat{L} + 1$

У той же час для блокових однокрокових методів порядок може бути визначений через кількість точок одного блоку: $r = k + 1$. Тоді кількість стадій ПНМРК і кількість точок в блоці багатоточкового методу зв'язані наступним співвідношенням: $r = 2s = k + 1 \Rightarrow k = 2s - 1$. Використовуючи отримані співвідношення, приведемо всі тимчасові характеристики до одного параметра, нехай це буде кількість стадій s (табл. 3.2).

Відмітимо, що серед множини неявних методів типу Рунге-Кутти для порівняння вибрані саме повністю неявні методи, оскільки вони володіють

ідентичними характеристиками стійкості, що й блокові однокрокові методи [4-5], а також через оптимальне співвідношення між порядком та кількістю стадій методів.

Таблиця 3.2 – Тимчасові характеристики ПНМРК і блокових однокрокових методів, приведені до одного параметру s

Методи		Коефіцієнт при T_F	Коефіцієнт При t_{op}	Коефіцієнт при $T_{all-to-all}$
ПНМРК (k - точок)	T_1	$4s^3 - s$	$8s^4 - 4s^3 + 8s^2 - 2$	
	T_p	$4s^2 - 1$	$8s^3 + 6s - 4$	$4s^2 - 1$
Блокові методи	T_1	$4s^2 - 4s + 2$	$16s^3 - 8s^2 + 2s - 1$	
	T_p	$2s$	$8s^2 + 1$	$2s$

Порівняємо час виконання послідовних алгоритмів даних методів. При домінуванні в обчисленнях часу звернення до правої частини ЗДР $T_F >> t_{op}$ обсяг обчислень для ПНМРК в s разів більший, ніж для блокових однокрокових методів:

$$T_1^{RK} / T_1^{BM} \approx \frac{ks(L+1)\cdot T_F}{(\hat{L}k+1)\cdot T_F} \approx \frac{s(4s^2-1)}{k^2+1} = \frac{4s^3 - s}{4s^2 - 4s + 2} \approx s \ .$$

Аналогічно, для паралельної реалізації ПНМРК та блокових багатоточкових однокрокових алгоритмів маємо:

$$\frac{T_p^{RK}}{T_p^{BM}} \approx \frac{k(L+1)\cdot T_F}{(\hat{L}+1)\cdot T_F} \approx \frac{4s^2-1}{k+1} = \frac{4s^2-1}{2s} \approx 2s.$$

Для тривіальної правої частини виходить аналогічний результат. Послідовна і паралельна реалізації обчислення розв'язку у нових k точках сітки інтегрування вимагають більших накладних витрат для неявних методів Рунге-Кутти у порівнянні з блоковими методами (рис. 3.28).

Відмітимо, що для паралельних алгоритмів ця різниця збільшується майже у 2 рази, що підтверджується експериментом на тестових задачах.

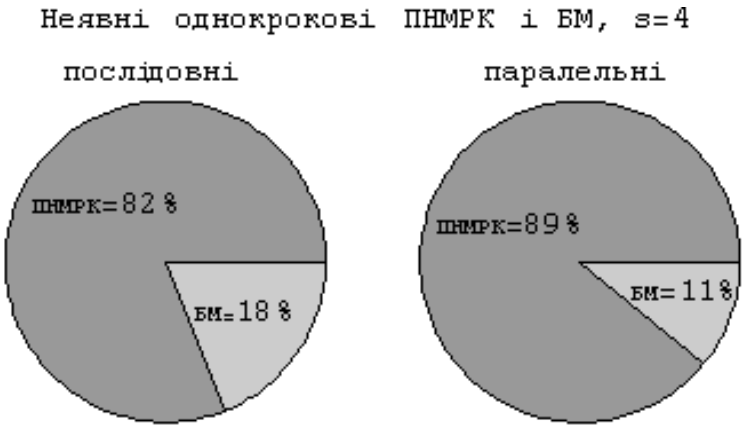


Рисунок 3.28 – Співвідношення часу виконання s-стадійних ПНМРК і k-точкових блокових вкладених однокрокових методів

Оцінка ефективності розпаралелювання проводиться на основі показників прискорення обчислень (рис. 3.29-3.30) у порівнянні зі своїм послідовним алгоритмом та у порівнянні з якнайкращим з послідовних алгоритмів, що були розглянуті.

$T_F \gg t_{op}$ $T_F \sim t_{op}$

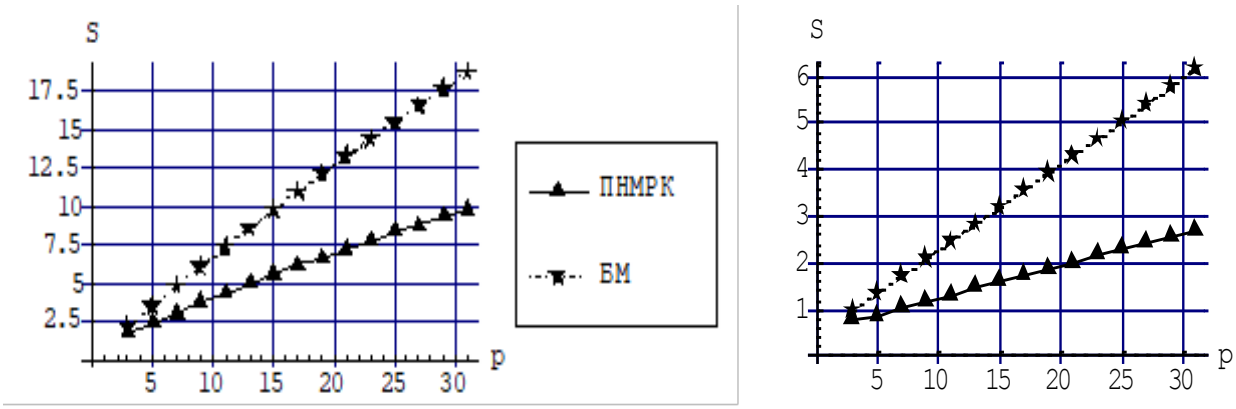


Рисунок 3.29 – Абсолютні коефіцієнти прискорення вкладених повністю неявних методів Рунге-Кутти та блокових однокрокових методів

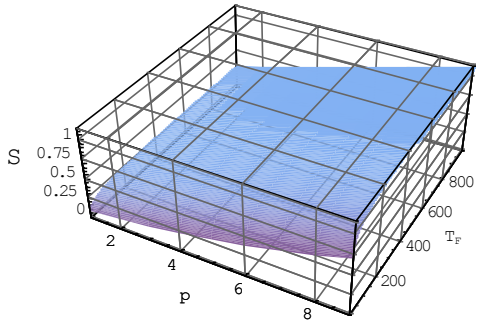


Рисунок 3.30 – Відносні коефіцієнти прискорення вкладених ПНМРК

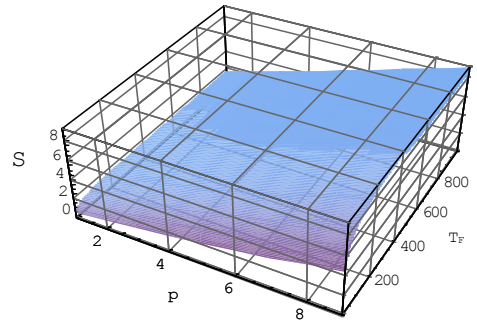


Рисунок 3.31 – Відносні коефіцієнти прискорення вкладених блокових однокрокових методів

На основі теоретичного аналізу виконання та аналогічних емпіричних досліджень можна зробити висновок, що для будь-яких засобів оцінки локальної похибки блокові багатоточкові однокрокові методи є найбільш ефективними з точки зору обчислювальних витрат у порівнянні з повністю неявними методами Рунге-Кутти того ж порядку точності. Таким чином, встановлено, що динамічні характеристики якості блокових багатоточкових методів перевершують відповідні характеристики багатостадійних однокрокових методів практично в s раз для послідовної та в $2s$ для паралельної реалізації. Ідея вкладених форм для НМРК може реалізована безпосередньо шляхом обчислення нової апроксимації рішення з використанням тих самих або кількох нових крокових коефіцієнтів. З іншого боку, оцінка локальної апостеріорної похибки може бути виконана шляхом порівняння двох апроксимацій рішення отриманих при виконанні

відповідної ітераційної схеми, що вирішує систему нелінійних рівнянь для визначення тих крокових коефіцієнтів.

3.7 Висновки за розділом

1. Запропоновано та обґрунтовано паралельні методи оцінки локальної апостеріорної похибки чисельного вирішення жорстких завдань Коші для одного диференціального рівняння на основі блокових неявних однокрокових різницевих схем:

1) блоковий k-точковий метод із правилом дублювання кроку;

2) вкладені форми на основі k та $(k+1)$-точкових блокових методів та з використанням методу послідовного підвищення порядку точності;

3) локальна екстраполяція з блоковим опорним методом.

2. Здійснено модифікацію алгоритмів, що реалізують різницеві блокові методи розпаралелювання для вирішення систем звичайних диференціальних рівнянь, яка з точки зору часових витрат дозволяє з більшою ефективністю порівняно з існуючими алгоритмами знаходити чисельне рішення задачі Коші.

3. Побудовано ітераційні паралельні алгоритми чисельного розв'язання нелінійного різницевого завдання Коші, що дозволяють отримувати результати із заданим ступенем точності. Наведено порівняльні характеристики чисельного вирішення тестових завдань та оцінки паралелізму. Рішення тестових завдань показало, що експериментальні оцінки прискорення та ефективності близькі до потенційних.

4. Наведено схеми відображення алгоритмів на паралельні обчислювальні структури, що ілюструють процеси скорочення обсягу послідовних обчислень. Для кожного з наведених алгоритмів розраховано аналітичні трудомісткості чисельного рішення, отримано експериментальні характеристики паралелізму, які свідчать про високу ефективність ($E > 0.7$) запропонованих методів.

РОЗДІЛ 4
РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ ТА ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1Модулі програмної системи

Розроблений програмний комплекс складається з двох незалежних модулів: Модуля Bevin_Masters_Block для виконання обчислень блоковим методом з паралельним алгоритмом та модуля Bevin_Masters_RK для виконання обчислень повністю неявним методом Рунге-Кутти з паралельним алгоритмом. Для виконання обчислень за повністю замкненим методом Рунге-Кутти.

Підсистема для обчислень за допомогою блочного методу, що має назву Bevin_Masters_Block, написана у вигляді консольного додатку на мові C+ для платформ Microsoft Windows з використанням принципів об'єктно-орієнтованого програмування. Такий підхід дозволяє легко запускати підсистему з інтерпретатора командного рядка CMD за допомогою інструкцій MPI.

Для запуску підсистеми Bervin_Masters_Block за технологією MPI використовується утиліта mpiexec. Формат запуску обчислень на чотирьох операціях показано на рисунку 4.1. Утиліта mpiexec виконує вказану кількість операцій перед запуском програми.

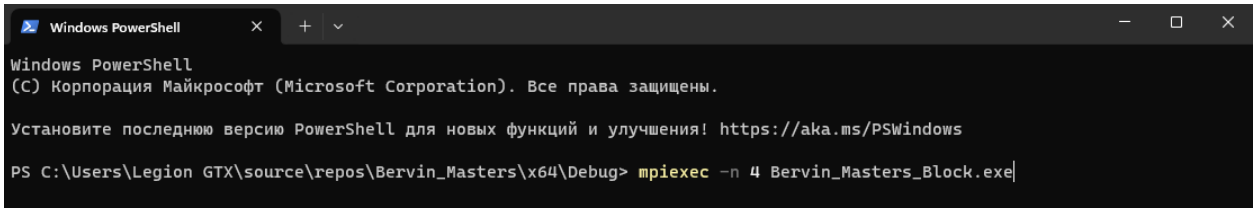


Рисунок 4.1 – Приклад запуску програми Bervin_Masters_Block на 4 процесах

Після завершення розрахунку головний процес записує результати у файл, а інші процеси завершують виконання.

Другий модуль програмного комплексу, котрий називається Bevin_Masters_RK, робить розрахунки за ПНМРК і реалізований аналогічним чином.

4.2 Опис розробленої програмної системи

Під час розробки програмного комплексу була розроблена бібліотека Function, яка містить класи з функціями та атрибутами, спільними для обох методів.

Клас Function є спільним для підсистем Bevin_Masters_Block та Bevin_Masters_RK, містить спільні методи, що використовуються для обчислень як у блочному, так і в паралельному алгоритмах ПНМРК. Опис полів наведено у таблиці 4.1.

Таблиця 4.1 – Опис основних полів класу Functions

Поле	Тип	Опис
AlphaQ	int	Коефіцієнт жорсткості
h1, h2	double	Розмір кроку для першого та другого блоків
t_0	double	Початкова умова розв’язання системи
inerc	int	Лічильник числа постійних кроків
U1, U2	double[]	Наближені значення розв’язання системи
badstep	int	Число відкинутих кроків
nstep	int	Число прийнятих кроків
maxnr	double	Максимальна локальна похибка

Продовження таблиці 4.1

Поле	Тип	Опис
w	double[]	Основна таблиця з результатами обчислень
wvsp	double[]	Допоміжна таблиця з результатами обчислень
Eps	double	Задана глобальна похибка
Iter	int	Максимальне число ітерацій
Fi00, Fi01, Fi10, Fi11, Fi20, Fi22	double[]	Стадійні коефіцієнти

Загальний опис методів класу Functions наведено у таблиці 4.2, а деталі методів sync () та send () описано у наступних підрозділах.

Таблиця 4.2 – Опис методів класу Functions

Назва методу	Призначення методу
ArraySubtraction()	Віднімання матриць
genB()	Генерація стадійних коефіцієнтів
MatrixMult(+1 перевантаження)	Перемноження матриці на матрицю, та перемноження матриці на число
MatrixPlus()	Складання матриць
Pow()	Зведення числа e в ступінь
F_()	Знаходження чисельного рішення системи рівнянь
Y_()	Знаходження точного рішення системи рівнянь
send ()	Пересилання даних від одного процесора іншім процесам однієї групи, або процесам обох груп

Продовження таблиці 4.2

Назва методу	Призначення методу
sync ()	Синхронізація даних між процесами однієї або двох груп, які приймали участь у розподільному обчисленні

Підсистеми Bevin_Masters_Block та Bevin_Masters_RK є основними системами, які збирають вихідні дані для обчислення, передають дані до класу Function, виконують алгоритм відповідного методу та зберігають результати у файлі.

4.3 Модифікований алгоритм розподілу процесів на дві групи

Для поділу процесорів на дві групи було використано алгоритм, який розділяє процесори на дві групи.

На рисунку 4.2 показано фрагмент коду реалізованого алгоритму.

```
if (size / 2 <= max(s_1, m_1))
{
    end_1_Group = (size / 2);

    if ((size / 2) <= max(s_2, m_1))
    {
        end_2_Group = (size / 2) * 2;
    }
    else {
        end_2_Group = max(s_2, m_1);
    }
}
else
{
    end_1_Group = max(s_1, m_1);

    if (max(s_2, m_1) < (size - (max(s_1, m_1))))
    {
        end_2_Group = ((max(s_1, m_1)) + max(s_2, m_1));
    }
    else {
        end_2_Group = size;
    }
}
```

Рисунок 4.2 – Фрагмент коду, який розподіляє процеси на дві групи за модифікованою схемою

Спочатку оголошуються додаткові змінні end_1_Group та end_2_Group, які вказують на порядок останнього процесу в першій та другій групах відповідно. Потім отримується загальна кількість доступних процесів і ділиться на два. Отримане значення порівнюється з кількістю балів, розрахованою в першому блоці, і якщо воно менше, ніж кількість етапів, то перша група процесів складається з половини доступних процесів і змінній end_1_Group присвоюється половина кількості процесів. Потім обчислюється необхідна кількість процесів у другій групі, і якщо кількість етапів у другому блоці більша за кількість процесів, що залишилися, то друга група складається з другої половини всіх процесів і змінній end_2_Group присвоюється число, що дорівнює загальній кількості доступних процесів. Однак, якщо кількість етапів у другому блоці менше, ніж кількість процесів, що залишилися, друга група складається з необхідної кількості процесів, що визначається кількістю етапів у другому блоці. Якщо кількість точок у першому блоці розрахунку менше половини наявних процесів, то перша група складається з такої ж кількості процесів, як і кількість етапів у першому блоці розрахунку. Потім визначається необхідна кількість операцій для другої групи, і якщо кількість етапів у другому блоці менше, ніж кількість операцій, що залишилися, то друга група складається з необхідної кількості операцій, що визначається кількістю етапів у другому блоці. Якщо ж кількість етапів у другому блоці більша за кількість процесів, що залишилися, то до другої групи потрапляють усі процеси, що залишилися.

Виходячи з того, що не всі процеси можуть брати участь в обчисленні, не можна використовувати метод Broadcast класу MPI. Цей метод перериває виконання програми всіх процесів і чекає, поки всі процеси отримають повідомлення. Тому в програмній системі реалізовано методи send() та sync() у класі Functions, які дозволяють здійснювати операції розподілу даних від одного процесу до іншого, який фактично бере участь в обчисленнях.

На рисунку 4.3 показано фрагмент коду методу send(). Цей метод надсилає повідомлення між паралельними процесами за наступними

принципами. Кожен процес чекає на повідомлення від процесу, який має надсилати повідомлення, і вказує в аргументі методу порядок, у якому цей процес має надсилати повідомлення. Аргумент методу також містить наступні дані: порядок першого та останнього процесу в інтервалі, в якому були визначені процеси, що беруть участь у передачі повідомлення.

```
void Function::send(MPI_Comm comm, int Rank, double* arr1, int N, int firstRank, int sendRank, int endRank)
{
    if (Rank == sendRank)
    {
        for (int i = firstRank; i < endRank; i++)
        {
            if (sendRank != i)
                MPI_Send(&arr1, N, MPI_DOUBLE, i, 0, MPI_COMM_WORLD);
        }

    }
    else
    {
        if (Rank != sendRank)
            MPI_Recv(&arr1, N, MPI_DOUBLE, sendRank, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
}
```

Рисунок 4.3 – Код методу send() для передачі повідомлень між паралельними процесами

Метод sync, код якого наведено у додатку В, виконує синхронізацію даних, що обчислюються паралельно. Іншими словами, якщо обчислення масиву виконується декількома паралельними процесами, і кожен процес обчислює тільки ту частину масиву, яка відповідає порядку обчислення, метод синхронізації виконує операцію об'єднання часток масиву і надсилає об'єднаний масив усім процесам, які брали участь у його формуванні.

4.4 Аналіз ефективності моделювання паралельних алгоритмів

У тесті блочного методу було розв'язано задачу Капса [47] за допомогою блочного методу.

$$x_1'(t) = -(\lambda + 2)x_1(t) + \lambda x_2^2(t), \;\; x_1(0) = 0, \;\; 0 \le t \le 10,$$

$$x_2'(t) = x_1(t) - x_2(t)\big(1 + x_2(t)\big), \;\; x_2(0), \qquad (4.1)$$

З відомими точними рішеннями

$$x_1(t) = e^{-2t},\ \ x_2(t) = e^{-t} \qquad (4.2)$$

Перед початком розрахунків було створено розрахункові коефіцієнти для еталонного та розрахункового блоків, а їхні значення записано в програмний код. На рисунках 4.4 та 4.5 показано фрагменти коду програмного комплексу, що реалізують чисельний та точний методи розв'язання задачі.

```
double* Function::F_(double AlphaQ, double t_, double* X0_, int koef_weight)
{
    double* f = new double[2];
    f[0] = -(AlphaQ + 2) * X0_[0] + AlphaQ * pow(X0_[1], 2);
    f[1] = X0_[0] - X0_[1] - Pow(X0_[1], 2);
    for (int i = 0; i < koef_weight; i++)
    {
        f[0] = -(AlphaQ + 2) * X0_[0] + AlphaQ * pow(X0_[1], 2);
        f[1] = X0_[0] - X0_[1] - Pow(X0_[1], 2);
    }
    return f;
}
```

Рисунок 4.4 – Реалізація методу чисельного розв’язання задачі Капса

До рівнянь були внесені дві додаткові модифікації: додатковий цикл для перерахунку рівнянь і додатковий аргумент для коефіцієнта складності правих частин системи. Додаткові цикли обчислень штучно збільшують розмір системи рівнянь для збільшення часу обчислень. Це необхідно для того, щоб час розв'язання одночасних рівнянь перевищував час пересилання повідомлень між паралельними процесами, а отже, щоб зручніше було аналізувати ефективність операцій зв'язку з покроковим керуванням.

```
double* Function::Y_(double t_)
{
    double* y = new double[2];
    y[0] = exp(-2 * t_);
    y[1] = exp(-t_);
    return y;
}
```

Рисунок 4.5 – Програмна реалізація точного розв’язку задачі Капса

Для порівняння паралельних алгоритмів ПНМРК та блочного методу в табл. 4.3 наведено результати обчислень обох методів з використанням чотирьох та восьми операцій.

Таблиця 4.3 – Зведені дані результатів обчислень паралельними алгоритмами з використанням 4 та 8 процесів

Коефіцієнт навантаження	ПНМРК		Блоковий	
	4 процесів	8 процесів	4 процесів	8 процесів
100	88,786	94,64	59,191	61,122
200	90,067	94,344	59,578	61,955
300	90,842	95,195	60,2	62,714
400	90,471	97,434	60,232	64,841
500	91,629	96,635	60,406	64,221
800	92,560	97,064	61,365	64,729
1000	94,872	97,882	63,248	65,321
5000	114,631	110,555	75,021	78,108
8000	126,798	122,485	83,532	85,673
10000	138,604	129,043	89,536	91,01

На рисунках 4.6 та 4.7 показано залежність часу обчислень від складності реалізації правої частини системи з використанням чотирьох та восьми процесів з різними алгоритмами паралельних обчислень.

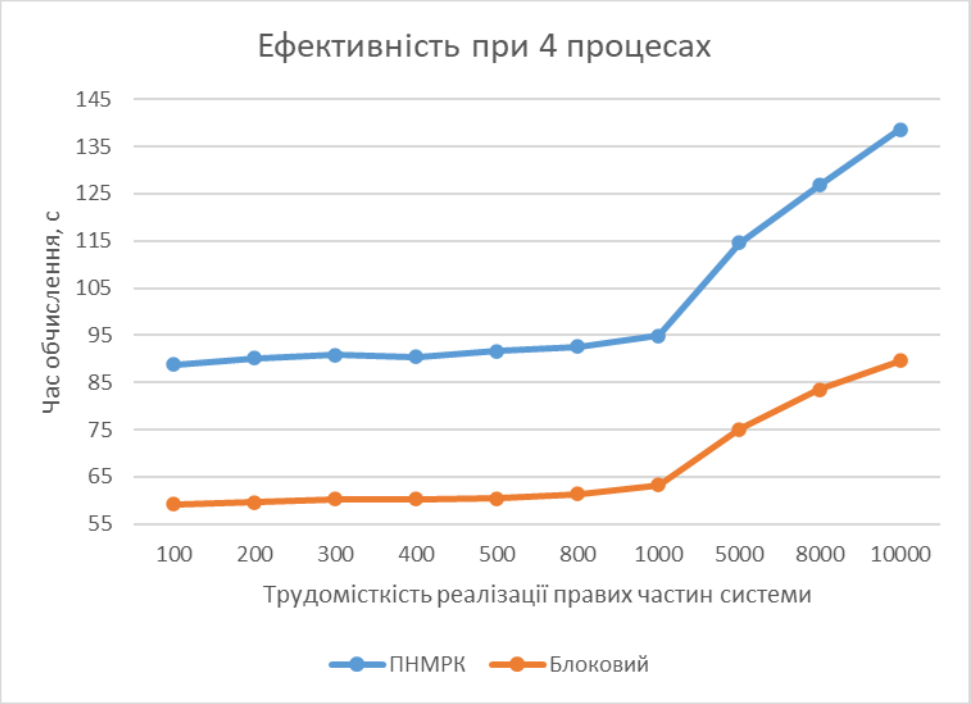


Рисунок 4.6 – Графік залежності часу обчислення від трудомісткості реалізації правих частин системи рівнянь на 4 процесах

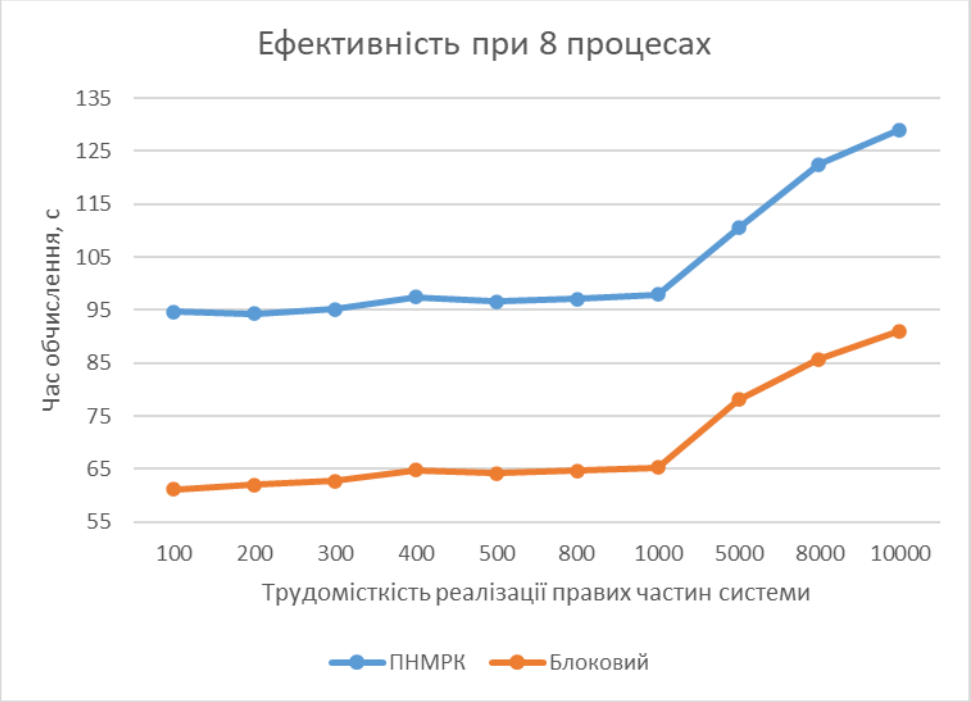


Рисунок 4.7 – Графік залежності часу обчислення від трудомісткості реалізації правих частин системи рівнянь на 8 процесах

Аналізуючи результати обчислень, наведені в таблиці 4.3 та зображені графічно на рисунках 4.6 та 4.7, можна зробити висновок, що блоковий метод на багатопроцесорному комп'ютері має кращі показники продуктивності, ніж паралельний алгоритм ПНМРК при розв'язанні задачі Коші для СЗДР. Аналізуючи графіки 4.6 та 4.7, можна зробити висновок, що при розв'язанні задачі Коші для СЗДР блоковий метод на багатопроцесорному комп'ютері має кращі показники продуктивності, ніж паралельний алгоритм повністю замкненого методу Рунге-Кутти.

4.5 Висновки за розділом

У розділі 4 описано модулі, класи та методи програмного комплексу, розробленого для розв'язання задачі Коші для СЗДР з використанням блочного методу та ПНМРК. Перший модуль відповідає за обчислення блоковим методом з використанням паралельного алгоритму, а другий - за обчислення паралельним алгоритмом ПНМРК. Бібліотека класів Function є спільною для обох модулів і реалізує спільні функції для обох паралельних алгоритмів; було виконано розбиття на дві групи, а операції зв'язку виконуються методами send() та sync() класу Functions.

Доведено, що використання блокових методів для побудови паралельних алгоритмів є більш ефективним, ніж використання ПНМРК.

ВИСНОВКИ

В роботі були розглянуто сучасні методи чисельного розв’язання задачі Коші для одного та системи звичайних диференційних рівнянь.

Визначено класи методів, які можуть бути використані для вирішення жорстких задач Коші: методи, що базуються на неявних чисельних схемах і тому мають достатні характеристики стійкості для розв’язання задач з особливостями.

Проведено дослідження ефективності використання блокових та повністю неявних багатостадійних методів типу Рунге-Кути методів на паралельних архітектурах з розподіленою пам’яттю та топологіями: кільце, 2D-тор та гіперкуб. Побудовані схеми відображення методів на паралельні архітектури заданих топологій та отримані динамічні характеристики якості для розроблених алгоритмів паралельних алгоритмів, а саме процеси комунікаційних операцій управління кроком і порядком інтегрування при вирішенні жорстких задач.

Була спроектована та розроблена програмна система з реалізацією запропонованих паралельних алгоритмів для проведення експериментів з метою їх дослідження та оцінки ефективності. Для перевірки отриманих теоретичних оцінок складності паралельних алгоритмів проведено експерименти на модельній задачі Капса з відомим точним розв’язком.

Розглянуто способи оцінки локальної похибки апостеріорної похибки: дублювання кроку, ЛЕР та вкладеність. Проведено порівняння обчислювальної складності паралельних реалізацій цих методів. Методи вкладених форм мають найменшу обчислювальну складність як для ПНМРК так і для БМ.

Проведено порівняння методів ПНМРК та БМ на основі ідеї вкладеності двох незалежних формул та ідеї послідовного підвищення порядку ітераційного процес, що породжується неявними схемами. Проаналізовано отримані результати експериментальних обчислень, які дозволили зробити висновок, що алгоритми, котрі мають у своїй основі повністю неявні методи Рунге-Кутти, є менш ефективними, ніж алгоритми, які розроблені за допомогою блокових методів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Бервін М.С. Дослідження ефективності паралельних методів розв’язання жорстких задач Коші / М.С. Бервін. – «ТАК»: телекомунікації, автоматика, комп’ютерно-інтегровані технології: зб. доповідей Всеукр. наук.-практ. конф. молодих вчених, Грудень 2023 р. // ДВНЗ «ДонНТУ; відп. ред. Г.В. Ступак. – Луцьк: ДВНЗ «ДонНТУ», 2022. – С. 4-6.

2. Grand Challenges: High performance computing and communications // A report by the Committee on Physical, Mathematical and Engineering Science, NSF/CISE, 1800 G. Street NW, Washington, DC 20550, 2001.

3. Огляд MPI. URL: https://www.mpich.org/about/overview/

4. Hairer E., Nørsett S.P., Wanner G. Solving Ordinary Differential Equations I: Nonstiff Problems. 2nd edition. Springer, 2011. 528 p.

5. Hairer E., Wanner G. Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems. Springer, 2010. 614p.

6. Фельдман Л.П., Назарова И.А. Современные параллельные методы численного решения задачи Коши. Донецк: ГВУЗ "ДонНТУ", 2013. 206с.

7. Паралельні однокрокові методи чисельного розв’язання задачі Коші: монографія / Л.П. Фельдман, І.А. Назарова. Донецьк: "ДВНЗ" ДонНТУ, 2011. 185 с.

8. Franklin M.A. Parallel solution of Ordinary Differential Equations. IEEE Transactions on Computer. 1978. Vol. 27, No. 5. P.413-420.

9. Stoer J., Bulirsch R. Introduction to Numerical Analysis. New York. Springer, 2002. 609 p.

10. Flynn M. Very high-speed computing systems. Proceeding of the IEEE №54 (12), 1966. P.1901-1909.

11. TOP500 Supercomputer Home Page [Електронний ресурс]. URL: https://www.top500.org

12. Foster *I.* Designing and Building Parallel Programs. URL: https://www.mcs.anl.gov/~itf/dbpp/

13. Grama A., Gupta A., Karypis G., Kumar V. Introduction to Parallel Computing. Addison Wesley. 856p. URL: https://www-users.cs.umn.edu/~karypis/parbook/

14. Назарова І.А., Дмитрієва О.А. Паралельні обчислення. Покровськ: ДВНЗ "ДонНТУ", 2020. 246с.

15. Pacheco Peter S. An Introduction to Parallel Programming (2nd Edition). Morgan Kaufmann publications, 2020. 450 p. ISBN-10: 0-12-804605-8. URL: http://www.e-tahtam.com/~turgaybilgin/2013-2014-guz/ParalelProgramlama/ParallelProg.pdf

16. Schmidt B., Gonzalez-Dominguez J., Hundt C., Schlarb M. Parallel Programming: Concepts and Practice 1st Edition. Morgan Kaufmann publications, 2017. 416 p. ISBN-13: 978-0128498903

17. Луцків А.М. Паралельні та розподілені обчислення: навч. посіб. / А.М. Луцків, С.А. Лупенко, В.В. Пасічник. Львів: вид-во Магнолія, 2017. 566 с. ISBN 978-617-574-110-8

18. Основні поняття про паралельні обчислення / Укладач Є. Ваврук. Львів: Національний університет “Львівська політехніка”, 2015. 109 с.

19. Worland P.B. Parallel methods for the numerical solution of ordinary differential equations. IEEE Transactions on Computer. 1976. Vol. 25, No. 10. P.1045-1048.

20. Молчанов И.Н. Введение в алгоритмы параллельных вычислений. Киев: Наукова думка, 1990. 128с.

21. Числові методи моделювання динамічних об'єктів в мультипроцесорних системах / О.А. Дмитрієва, Н.Г. Гуськова, Є.О. Башков, І.А. Назарова: монографія. Покровськ: ДВНЗ "ДонНТУ", 2020. 268 с.

22. Назарова І.А. Паралельні неявні блокові методи чисельного розв’язання жорстких динамічних задач із зосередженими параметрами. Научно-

теоретический журнал ИПИИ НАН Украины "Искусственный интеллект". №3. 2009. Донецк: ИПИИ. С. 404-412.

23. Назарова І.А. Моделювання паралельних процесів при розв’язанні багатовимірних жорстких задач Коші. Матеріали XII Міжнародної науково-практичної конференції "Інформаційні технології і автоматизація". Одеса: ОНАХТ, 2019. С. 36-37.

24. Назарова І.А. Паралельне моделювання динамічних жорстких процесів на базі блокових методів. Актуальні питання розвитку інформаційних технологій: тези доповідей Всеукраїнської конференції молодих учених. Маріуполь: ПДТУ, 2019. С. 43-45.

25. Назарова І.А. Розробка та дослідження ефективності паралельних алгоритмів розв'язання жорстких задач Коші для мультикомп'ютерів. III Всеукраїнська науково-практична інтернет-конференція "Комп’ютерні технології в енергетиці, електромеханіці та системах управління". Бахмут: ННППІ УІПА, 2020. С.30-31.

Додаток А

Зауваження нормоконтролера

Таблиця А.1 – Зауваження нормоконтролера

Сторінка	Позначення	Зміст зауваження
Розділ 2		Два рисунки з номером 2.8
Розділ 3		Нещільне заповнення сторінок
ПЗ		Змінні в тексті та формулах мають подекуди різний шрифт

Додаток Б

Презентація результатів дипломної роботи

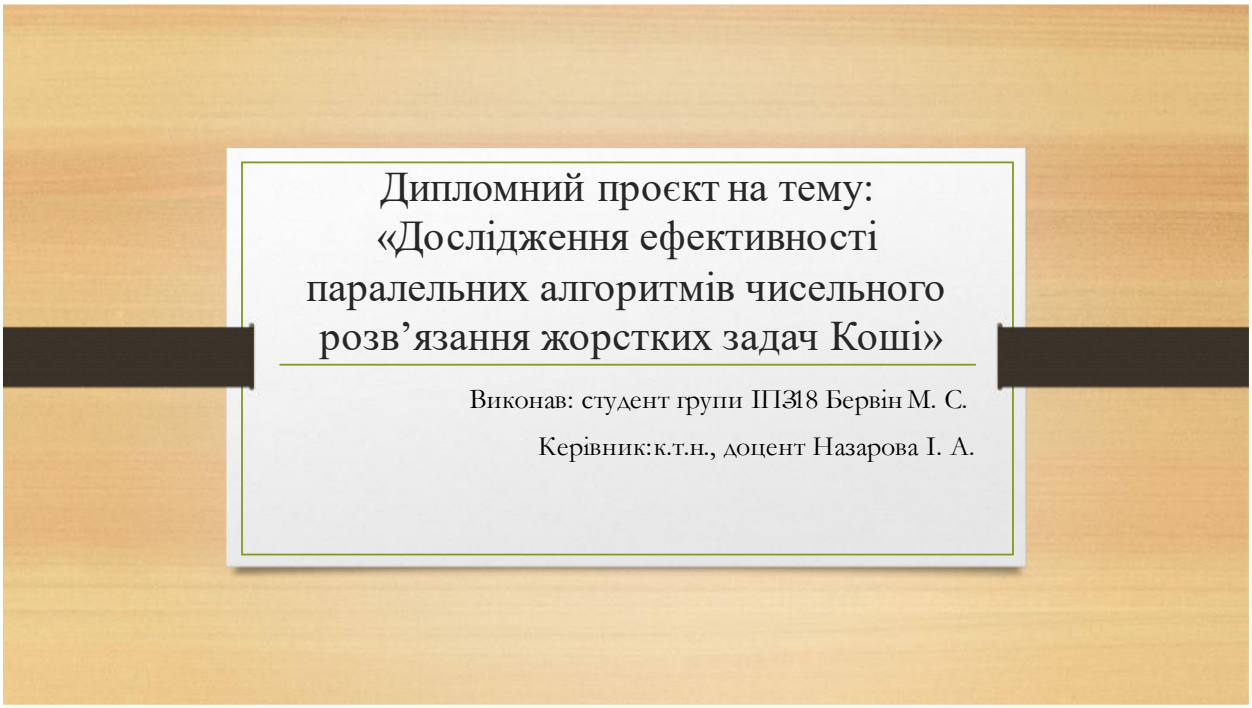


Рисунок Б.1 – Слайд 1

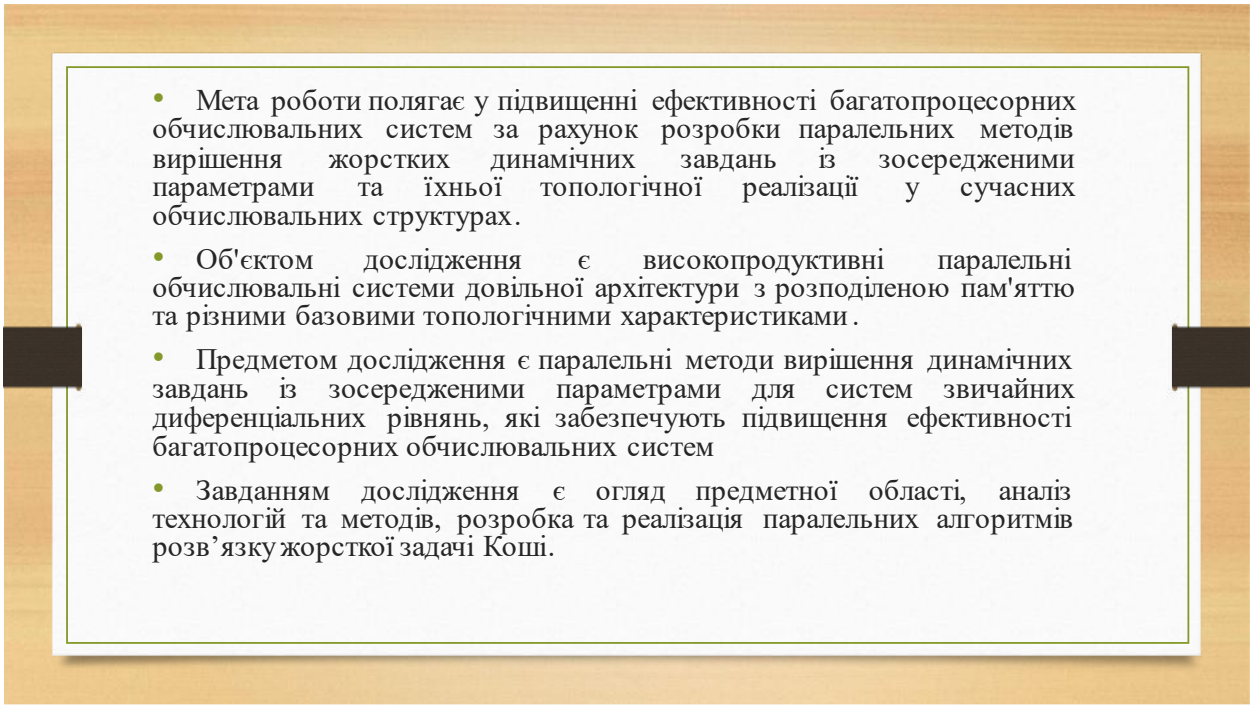


Рисунок Б.2 – Слайд 2

Постановка задачі

Чисельно розв'язується задача Коші для систем звичайних диференційних рівнянь першого порядку з відомими початковими умовами:

$$\frac{dY(x)}{dx} = F\big(x, Y(x)\big), Y : R \to R^m;\ F: R \times R^m \to R^m,$$

де:

- m — порядок,
- Y — вектор невідомих,
- F — права частина системи, котра у загальному випадку є нелінійною функцією, з заданими початковими умовами $Y(t_0) = Y_0$.

Рисунок Б.3 – Слайд 3

Топологічні аспекти моделювання паралельних обчислень

Основні комунікаційніоперації

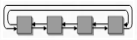
- передача даних між двома процесорами мережі – операція "point -point" ;
- передача даних від одного процесора всім іншим процесорам мережі – операція "one-to-all";
- передача даних від усіх процесорів мережі усім процесорам мережі – множинна пересилка "all-to-all".

Лінійна модель Хокні : $T_{p-p} = t_s + t_w \cdot V \cdot l, t_w = y/B$

t_s – латентність, тривалість підготовки повідомлення дляпередачі;
t_w – час передачі одного байту;
V – об'єм повідомлення в байтах;
l – довжина маршруту;
y – кількість байт у слові;
B – пропускна способність каналу передачі даних (байт/секунда);

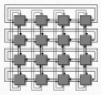
Топології

Кільце



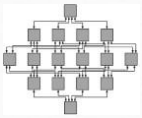
$T^R_{p-p} = t_s + V \cdot t_w \cdot \lfloor p/2 \rfloor$
$T^R_{one-to-all} = t_s + V \cdot t_w \cdot \lceil p/2 \rceil$
$T^R_{all-to-all} = (t_s + V \cdot t_w) \cdot (p-1)$

2D-тор



$T^M_{p-p} = t_s + 2V \cdot t_w \cdot \lfloor p/2 \rfloor$
$T^M_{one-to-all} = t_s + 2V \cdot t_w \cdot \lceil p/2 \rceil$
$T^M_{all-to-all} = 2t_s(\sqrt{p}-1) + V \cdot t_w \cdot \cdot (p-1)$

Гіперкуб



$T^H_{p-p} = t_s + V \cdot t_w \cdot \log_2 p$
$T^H_{one-to-all} = (t_s + V \cdot t_w) \cdot \log_2 p$
$T^H_{all-to-all} = t_s \cdot \log_2 p + t_w \cdot V \cdot \cdot (p-1)$

Рисунок Б.4 – Слайд 4

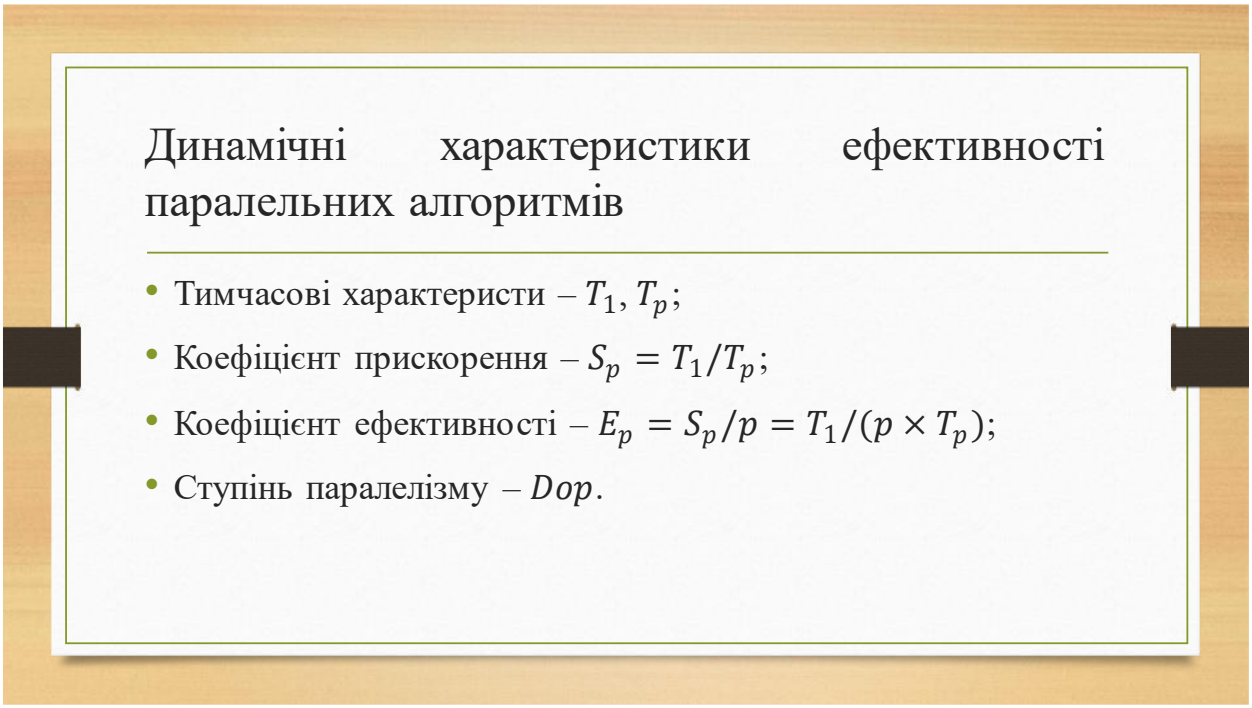


Рисунок Б.5 – Слайд 5

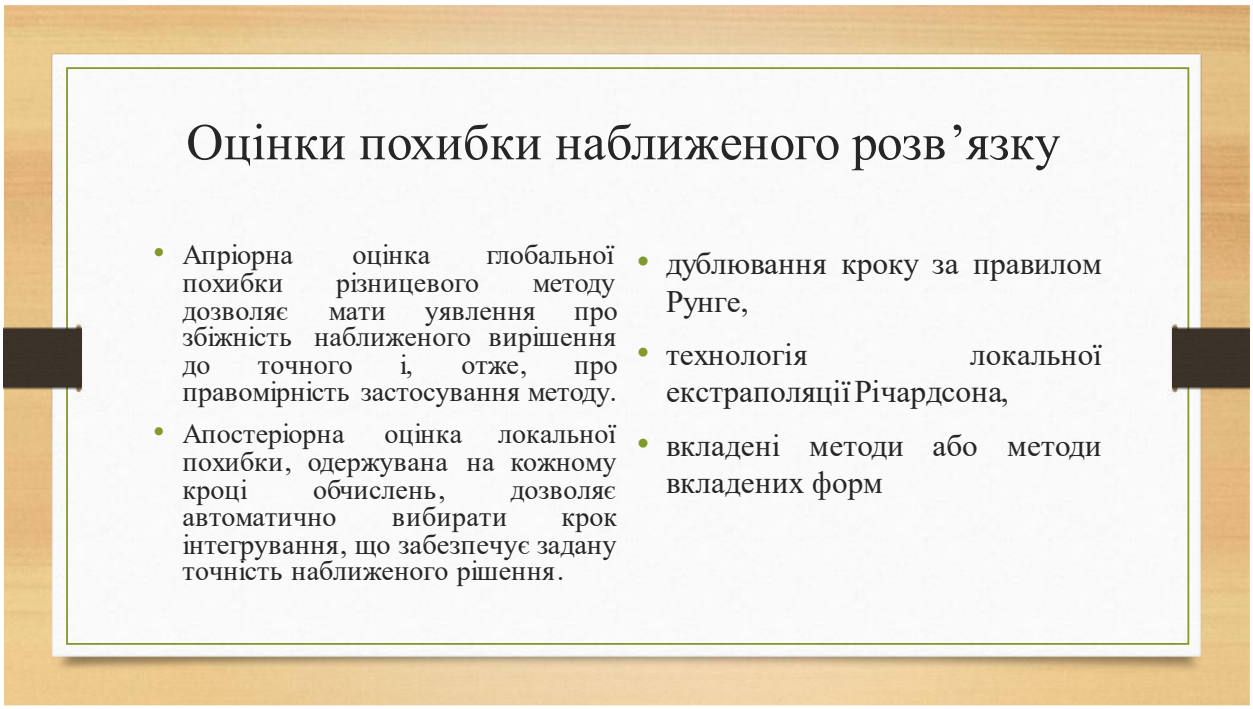


Рисунок Б.6 – Слайд 6

Повністю неявні методи Рунге-Кутти

Повністю неявний s-стадійний методтипа Рунге-Кутти (ПНМРК) описуєтьсянаступнимчином:

$$Y_{n,i} = Y_{n,0} + h \cdot \sum_{i=1}^{s} b_i K_i, i = \overline{1,s},$$

$$K_i = F\left[t_n + c_i h;\ Y_n + h \cdot \sum_{i=1}^{s} a_{ij} K_j\right],$$

де $C = (c_1, \dots, c_s)^T$, $B = (b_1, \dots, b_s)^T$ та $A = (a_{ij}), i, j = \overline{1,s}$ задають унікальний варіант методу й вибираються з міркувань точності;

$K_i, i = \overline{1,s}$ – вектор стадійних коефіцієнтів;

h – крок інтегрування;

s – кількість стадій методу, що визначає його порядок апроксимації;

Y_n – вектор наближеного розв'язку в точці n.

c_1	a_{11}	a_{12}	...	a_{1s-1}	a_{1s}
c_2	a_{21}	a_{22}	...	a_{2s-1}	a_{2s}
c_3	a_{31}	a_{32}	...	a_{3s-1}	a_{3s}
...	...	...	...	...	...
c_s	a_{s1}	a_{s2}	...	a_{ss-1}	a_{ss}
	b_1	b_2	...	b_{s-1}	b_s

Матриця Батчера

Рисунок Б.7 – Слайд 7

Повністю неявні методи Рунге-Кутти

До переваг повністю неявних однокрокових методів слід віднести добрі характеристики стійкості, достатні для розв'язання жорстких задач Коші. Недоліками цих методів є висока обчислювальна складність, обумовлена необхідністю розв'язання систем нелінійних алгебраїчних рівнянь на базі деякого ітераційного процесу для визначення стадійних коефіцієнтів :

$$\begin{cases} \bar{k}_1 = \bar{f}[x_n + c_1 h;\ \bar{y}_n + h(a_{11}\bar{k}_1 + a_{12}\bar{k}_2 + \dots + a_{1s}\bar{k}_s)] \\ \bar{k}_2 = \bar{f}[x_n + c_2 h;\ \bar{y}_n + h(a_{21}\bar{k}_1 + a_{22}\bar{k}_2 + \dots + a_{2s}\bar{k}_s)] \\ \dots \\ \bar{k}_s = \bar{f}[x_n + c_s h;\ \bar{y}_n + h(a_{s1}\bar{k}_1 + a_{s2}\bar{k}_2 + \dots + a_{ss}\bar{k}_s)]. \end{cases}$$

До методів розв'язання отриманої СНАР слід віднести схеми простої ітерації та підходи Ньютона. Наприклад, підчас вирішення СЗДР повністю неявними методами $m \times s$ стадійних коефіцієнтів можуть бути отриманими на основі наступної ітераційної схеми:

$$K_i^{(0)} = F\left[x_n + c_i h;\ G_i^{(0)}\right].$$

$$K_i^{(l+1)} = F\left[x_n + c_i h;\ G_i^{(l)}\right],$$

де $G_i^{(0)} = Y_0$, $G_i^{(l)} = Y_n + h \cdot \sum_{i=1}^{s} a_{ij} K_j^{(l)}$,

$i = \overline{1,s}, l = \overline{1,N}.$

Рисунок Б.8 – Слайд 8

Паралельні повністю неявні багатостадійні методи Рунге-Кутти

Розпаралелювання ПНМРК, як і інших однокрокових методів, базується на виконанні одного кроку інтегрування.

Процесори розбиваються на s груп за кількістю стадійних векторів, кожен процесор групи обчислює всього $\lceil m/p \rceil$ компонент таких векторів і потім передає їх всім процесорам групи.

Обчислення рішення у наступній точці вимагає реалізації операцій множинного пересилання даних за типом “all-to-all”.

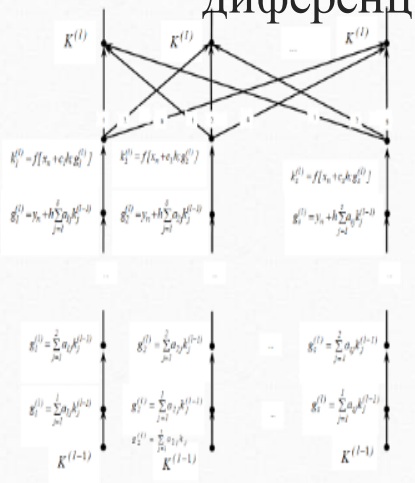
Міжгруповий обмін може бути здійснений двома способами:

1) кожний перший процесор у групі передає вектор у перший процесор кожної іншої групи з паралельним груповим обміном у кожній групі;

2) міжгрупова передача за типом “all-to-all”.

Рисунок Б.9 – Слайд 9

Паралельний ПНМРК для звичайного диференційного рівняння



Час послідовного виконання :

$$T_1^{\text{ПНМРК}} = \sum_{i=1}^{s} T(k_i^{(0)}) + \sum_{i=1}^{s} T(k_i^{(i)}) + T(y_{n+1})$$

$$T_1^{\text{ПНМРК}} = [illegible] + 2[illegible]) + [illegible][2[illegible] + [illegible]] + 2([illegible] + 1)[illegible]$$

Час обчислень правої частини:

$$T_{p,comp}^{\text{ПНМРК}} = (T_f + 2t_{op}) + N\left[2st_{op} + T_f\right] + 2(s+1)\,t_{op}$$

Коефіцієнт потенційного прискорення правої частини

$$S_{pot}^{\text{ПНМРК}} = T_1^{\text{ПНМРК}} \Big/ T_{p,comp}^{\text{ПНМРК}} \approx \frac{s(N+1)T_f}{(N+1)T_f} = s = p.$$

Час міжпроцесорного обміну:

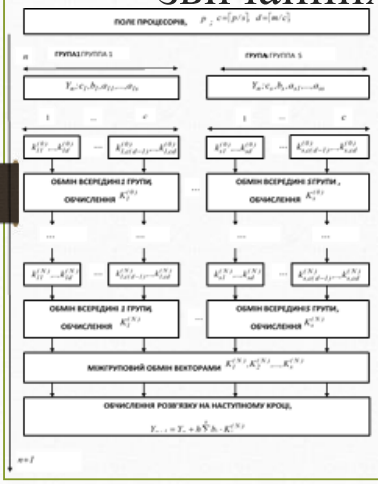
Кільце: $T_{p,comm}^{\text{ПНМРК},R} = (N+1)(t_s + t_w)\,(p-1)$;

2D-тор: $T_{p,comm}^{\text{ПНМРК},M} = (N+1)\left[2t_s\left(\sqrt{p}-1\right) + t_w(p-1)\right]$;

Гіперкуб: $T_{p,comm}^{\text{ПНМРК},H} = (N+1)[t_s log_2 p + t_w(p-1)]$.

Рисунок Б.10 – Слайд 10

Паралельний ПНМРК для системи звичайних диференційних рівнянь



Час послідовного виконання :

$$T_1^{\text{ПНМРК,СЗДР}} = (N+1)s \cdot \sum_{i=1}^{m} T_{f_i} + (2Nms^2 + 2ms + 2s + 2m) \cdot t_{op}.$$

Час обчислень компонент вектора:

SIMD: $T_F = \sum_{i=1}^{m} T_{f_i}$

MIMD:

$$T_{p,comp}^{\text{ПНМРК,СЗДР}} = \underbrace{t_{mul} + t_{ad} + \left\lceil \frac{ms}{p} \right\rceil T_F}_{\tilde{k}^{(0)}} + N \cdot \underbrace{\left\lceil \frac{ms}{p} \right\rceil [T_F + s(t_{mul} + t_{ad})]}_{\tilde{k}^{(1)}} + \underbrace{\left\lceil \frac{m}{p} \right\rceil (s+1)(t_{mul} + t_{ad})}_{\tilde{y}_{n+1}}, T = \max_{i=1}^{m} T_{f_i}$$

Час міжгрупового обміну:

Кільце: $T_{p,comm;2}^{\text{ПНМРК–СЗДР},R} = \left[2t_s + \left(\left\lceil \frac{m}{p} \right\rceil + \left\lceil \frac{ms}{p} \right\rceil\right) t_w\right](p-1);$

2D-тор $T_{p,comm;2}^{\text{ПНМРК–СЗДР},M} = 4\left(\sqrt{p}-1\right)t_s + \left(\left\lceil \frac{m}{p} \right\rceil + \left\lceil \frac{ms}{p} \right\rceil\right)(p-1)t_w;$

Гіперкуб $T_{p,comm;2}^{\text{ПНМРК–СЗДР},H} = 2log_2 p\, t_s + \left(\left\lceil \frac{m}{p} \right\rceil + \left\lceil \frac{ms}{p} \right\rceil\right)(p-1)t_w.$

Час внутрішньогрупового обміну:

Кільце: $T_{p,comm;1}^{\text{ПНМРК–СЗДР},R} = (N+1)\left(t_s + \left\lceil \frac{ms}{p} \right\rceil t_w\right)\left(\left\lceil \frac{p}{s} \right\rceil - 1\right);$

2D-тор $T_{p,comm;1}^{\text{ПНМРК–СЗДР},M} = (N+1)\left[2\left(\sqrt{\left\lceil \frac{p}{s} \right\rceil} - 1\right)t_s + \left\lceil \frac{ms}{p} \right\rceil\left(\left\lceil \frac{p}{s} \right\rceil - 1\right)t_w\right];$

Гіперкуб $T_{p,comm;1}^{\text{ПНМРК–СЗДР},H} = (N+1)\left[log_2\left(\left\lceil \frac{p}{s} \right\rceil\right)t_s + \left\lceil \frac{ms}{p} \right\rceil\left(\left\lceil \frac{p}{s} \right\rceil - 1\right)t_w\right].$

Рисунок Б.11 – Слайд 11

Блокові/багатоточкові методи

У випадку блокових методів розв'язання задачі Коші множина точок рівномірної сітки, отриманої з кроком інтегрування h:

$$\Omega_h : \{x_j\}, j = \overline{1,M}, N \le M$$

розбивається на N блоків. Кожен блок містить k точок і при цьому $N \le M$. Передбачається, що в межах блоку всі точки рівновіддалені одна від одної:

$$x_{n,i} = x_{n,0} + ih, i = \overline{1,k}$$

де: i – номер точки в блоці $i = \overline{1,k}$;

n – номер боку $n = \overline{1,N}$;

$x_{n,i}$ – точка з номером i, що належить блоку n;

$x_{n,0}$ – початкова точка n-го блоку;

$x_{n,k}$ – кінцева точка n-го блоку

Рівняння однокрокових блокових різницевих методів у вживанні до СЗДР для блоку з k-точок можуть бути записані і таким чином:

$$Y_{n\,i} = Y_{n,0} + ih\left[b_i F_{n,0} + \sum_{j=1}^{k} a_{ij} F_{n,j}\right];$$
$$i = \overline{1,k}; n = \overline{1,N},$$

де: N – кількість блоків сітки інтегрування.

Метод простої функціональної ітерації:

$$\begin{cases} y_{n,i,0} = y_{n,0} + ihF_{n,0}, i = \overline{1,k}, n = 1,2,\ldots,N, \\ y_{n,i,l+1} = y_{n,0} + ih\left(b_i F_{n,0} + \sum_{j=1}^{k} a_{i,j} F_{n,j,l}\right), \; l = \overline{0,L-1}, \end{cases}$$

l – номер поточної ітерації $l = \overline{0,L-1}$;

L – максимальне число ненульових ітерацій.

Схема розбиття на блоки для однокрокового k-точкового методу

Рисунок Б.12 – Слайд 12

Порівняльний аналіз неявних однокрокових методів розв'язання жорстких задач Коші

Повністю неявні багатостадійні методи типу Рунге-Кутти (ПНМРК)

Блокові/багаточкові неявні методи (БМ)

- s – кількість стадій
- L – кількість ітерацій СНАР ПНМРК
- $r = 2s$

r – порядок методу

- k – кількість точок в блоці
- $\hat{L}$ – кількість ітерацій СНАР БМ
- $r = k + 1$

Для ПНМРК за теоремами Батчера:

- метод типу Гауса $r = 2s$
- метод типу Радо $r = 2s - 1$
- метод типу Лобатто $r = 2s - 2$

ПНМРК мають ідентичні характеристики стійкості, що і БМ, а також оптимальне співвідношення між порядком і числом стадій

Умови порівняння ПНМРК та БМ:

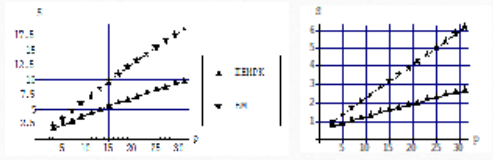
- один і той же порядок точності r
- розв'язок k нових точок
- породжувальні ітераційні процеси мають граничну кількість ітерацій : L = 2s, $\hat{L} = k$

Рисунок Б.13 – Слайд 13

Порівняльний аналіз неявних однокрокових методів розв'язання жорстких задач Коші

Методи		Коефіцієнт при T_F	Коефіцієнт При t_{op}	Коефіцієнт при $T_{all-to-all}$
ПНМРК (k-точок)	T_1	$4s^3 - s$	$8s^4 - 4s^3 + 8s^2 - 2$	
	T_p	$4s^2 - 1$	$8s^3 + 6s - 4$	$4s^2 - 1$
Блокові методи	T_1	$4s^2 - 4s + 2$	$16s^3 - 8s^2 + 2s - 1$	
	T_p	$2s$	$8s^2 + 1$	$2s$

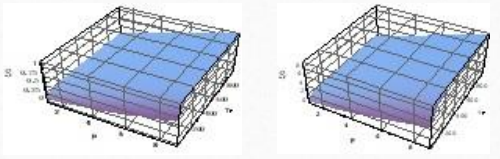
$T_F \gg t_{op}$ $T_F \sim t_{op}$



Коефіцієнти прискорення вкладених ПНМРК/БМ від p

Для одного процесору $\frac{T_1^{\text{ПНМРК}}}{T_1^{\text{БМ}}} \approx \frac{ks(L+1)T_F}{(k\hat{L}+1)T_F} \approx s$

Для p процесорів $\frac{T_p^{\text{ПНМРК}}}{T_p^{\text{БМ}}} \approx \frac{k(L+1)T_F}{(\hat{L}+1)T_F} \approx 2s,$



Коефіцієнти прискорення вкладених ПНМРК/БМ від T_F

Рисунок Б.14 – Слайд 14

Рисунок Б.15 – Слайд 15

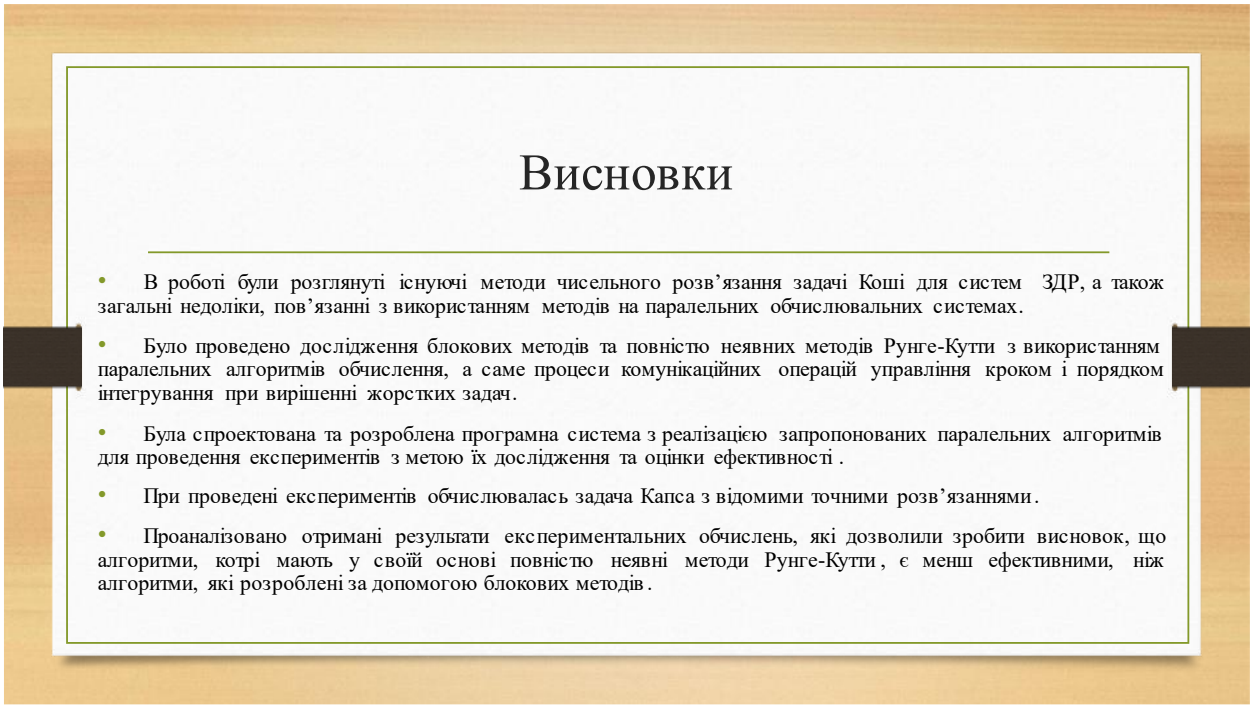


Рисунок Б.16 – Слайд 16

Додаток В

Лістинг модулей програмного продукту

B.1 Function

```
double* Function::MatrixMult(double* pAMatrix, int N, double* pBMatrix, int r1, int c1,
int r2, int c2)
{
    int Size = N;
    int col = Size / 2;
    double* pCMatrix = new double[r1 * c2];
    int i, j, k_;
    for (i = 0; i < r1; i++)
        for (j = 0; j < c2; j++)
            for (k_ = 0; k_ < c1; k_++)
                pCMatrix[i * c2 + j] += pAMatrix[i * c1 + k_] * pBMatrix[k_ * c2 + j];
    return pCMatrix;
}

double* Function::MatrixMult(double* pAMatrix, int N, double x)
{
    int Size = N;
    double* pCMatrix = new double[Size];
    for (int i = 0; i < Size; i++)
        pCMatrix[i] = pAMatrix[i] * x;
    return pCMatrix;
}

double* Function::MatrixPlus(double* AMatrix, int N, double* BMatrix)
{
    int Size = N;
    double* CMatrix = new double[Size];
    for (int i = 0; i < Size; i++)
        CMatrix[i] = AMatrix[i] + BMatrix[i];
    return CMatrix;
}

double* Function::ArraySubtraction(double* AMatrix, double* BMatrix, int s_1, int start)
{
    double* res = new double[2];
    res[0] = AMatrix[start + 0] - BMatrix[(2) + start * 2 + 0];
    res[1] = AMatrix[start + 1] - BMatrix[(2) + start * 2 + 1];
    return res;
}

double Function::Pow(double base, double exponent)
{
    if (exponent == 0.0) return 1.0;
    double result = base;
    for (double iteration = 1.0; iteration < exponent; iteration += 1.0)
    {
        result *= base;
    }
    return result;
}

double* Function::genB(int caseSwitch, int step, int s1)
{
    double* B_ = new double[1];
    switch (caseSwitch)//(2,2)_(2,4)
    {
    case 22:
        switch (step)
        {
        case 0:
            if (s1 == 2) //s2 = 4
                B_ = new double[] { -1 / 2.0, 3 / 2.0, -2.0, 4.0 };
```

```
            break;
        case 1:
            if (s1 == 2) //s2 = 4
                B_ = new double[] { -2.0, 4.0, -8.0, 12.0 };
            break;
        case -1:
            if (s1 == 2)
                B_ = new double[] { -1 / 8.0, 5 / 8.0, -1 / 2.0, 3 / 2.0 };
            break;
        }
        break;
    case 24:
        switch (step)
        {
        case 0:
            if (s1 == 2) //s2 = 4
                B_ = new double[] { -1 / 8.0, 5 / 8.0, -1 / 2.0, 3 / 2.0, -9 / 8.0, 21 /
8.0, -2.0, 4.0 };
            break;
        case 1:
            if (s1 == 2) //s2 = 4
                B_ = new double[] { -1 / 2.0, 3 / 2.0, -2.0, 4.0, -9 / 2.0, 15 / 2.0, -
8.0, 12.0 };
            break;
        case -1:
            if (s1 == 2)
                B_ = new double[] { -1 / 32.0, 9 / 32.0, -1 / 8.0, 5 / 8.0, -9 / 32.0, 33
/ 32.0, -1 / 2.0, 3 / 2.0 };
            break;
        }
        break;
    case 222:
        switch (step)
        {
        case 0:
            if (s1 == 2)
                B_ = new double[] { -1 / 24.0, 13 / 24.0, 0 / 3.0, 1 / 3.0 };
            break;
        case 1:
            if (s1 == 2)
                B_ = new double[] { -4 / 15.0, 20 / 15.0, 0 / 3.0, 2 / 3.0 };
            break;
        case -1:
            if (s1 == 2)
                B_ = new double[] { -1 / 192.0, 46 / 192.0, 0 / 6.0, 1 / 6.0 };
            break;
        }
        break;
    case 122:
        switch (step)
        {
        case 0:
            if (s1 == 2)
                B_ = new double[] { 13 / 24.0, -1 / 24.0, 4 / 3.0, 1 / 3.0 };
            break;
        case 1:
            if (s1 == 2)
                B_ = new double[] { 15 / 15.0, -1 / 15.0, 8 / 3.0, 2 / 3.0 };
break;
        case -1:
            if (s1 == 2)
                B_ = new double[] { 56 / 192.0, -5 / 192.0, 4 / 6.0, 1 / 6.0 };
break;
        }
```

```
        break;
    case 224:
        switch (step)
        {
        case 0:
            if (s1 == 2)
                B_ = new double[] { -9 / 5760.0, 1139 / 5760.0, -1 / 1080.0, 189 /
1080.0, -1 / 640.0, 123 / 640.0, 0 / 45.0, 7 / 45.0 };
            break;
        case 1:
            if (s1 == 2)
                B_ = new double[] { -27 / 1440.0, 637 / 1440.0, -1 / 90.0, 34 / 90.0, -3
/ 160.0, 3 * 23 / 160.0, 0 / 45.0, 2 * 7 / 45.0 };                              break;
        case -1:
            if (s1 == 2)
                B_ = new double[] { -27 / 322560.0, 30002 / 322560.0, -1 / 20160.0, 1694
/ 20160.0, -3 / 35840.0, 3 * 1078 / 35840.0, 0 / 90.0, 7 / 90.0 };
break;
        }
        break;
    case 124:
        switch (step)
        {
        case 0:
            if (s1 == 2) //s2 = 4
                B_ = new double[] { 2224 / 5760.0, -651 / 5760.0, 208 / 5760.0, -31 /
5760.0, 704 / 1080.0, 189 / 1080.0, 0 / 1080.0, -1 / 1080.0, 368 / 640.0, 333 / 640.0,
144 / 640.0, -7 / 640.0, 32 / 45.0, 12 / 45.0, 32 / 45.0, 7 / 45.0 };
            break;
        case 1:
            if (s1 == 2)
                B_ = new double[] { 1022 / 1440.0, -258 / 1440.0, 77 / 1440.0, -11 /
1440.0, 114 / 90.0, 34 / 90.0, -1 / 90.0, 0 / 90.0, 3 * 58 / 160.0, 3 * 58 / 160.0, 3 *
23 / 160.0, -3 / 160.0, 2 * 32 / 45.0, 2 * 12 / 45.0, 2 * 32 / 45.0, 2 * 7 / 45.0 };
break;
        case -1:
            if (s1 == 2)
                B_ = new double[] { 66304 / 322560.0, -22008 / 322560.0, 7552 / 322560.0,
-1183 / 322560.0, 6720 / 20160.0, 1624 / 20160.0, 64 / 20160.0, -21 / 20160.0, 3 * 3584 /
35840.0, 3 * 2968 / 35840.0, 3 * 1408 / 35840.0, 3 * (-77) / 35840.0, 32 / 90.0, 12 /
90.0, 32 / 90.0, 7 / 90.0 };                                  break;
        }
        break;
    }
    return B_;
}

double* Function::F_(double AlphaQ, double t_, double* X0_, int koef_weight)
{
    double* f = new double[2];
    f[0] = -(AlphaQ + 2) * X0_[0] + AlphaQ * pow(X0_[1], 2);
    f[1] = X0_[0] - X0_[1] - Pow(X0_[1], 2);
    for (int i = 0; i < koef_weight; i++)
    {
        f[0] = -(AlphaQ + 2) * X0_[0] + AlphaQ * pow(X0_[1], 2);
        f[1] = X0_[0] - X0_[1] - Pow(X0_[1], 2);
    }
    return f;
}

double* Function::Y_(double t_)
{
    double* y = new double[2];
    y[0] = exp(-2 * t_);
```

```
    y[1] = exp(-t_);
    return y;
}

void Function::send(MPI_Comm comm, int Rank, double* arr1, int N, int firstRank, int
sendRank, int endRank)
{
    if (Rank == sendRank)
    {
        for (int i = firstRank; i < endRank; i++)
        {
            if (sendRank != i)
                MPI_Send(&arr1, N, MPI_DOUBLE, i, 0, MPI_COMM_WORLD);
        }

    }
    else
    {
        if (Rank != sendRank)
            MPI_Recv(&arr1, N, MPI_DOUBLE, sendRank, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
    }
}

void Function::sync(MPI_Comm comm, int Rank, int startRank, int endRank, double* arr1,
int N, int col)
{
    int n = (int)ceil((double)N / col / (endRank - startRank));
    double* send = new double[n * col];
    for (int i = 0; i < n; i++)
    {
        if ((Rank - startRank) + i * (endRank - startRank) < N / col)
        {
            if (col >= 1)
                send[i * col + 0] = arr1[(Rank - startRank) * col + i * col * (endRank -
startRank) + 0];
            if (col >= 2)
                send[i * col + 1] = arr1[(Rank - startRank) * col + i * col * (endRank -
startRank) + 1];
            if (col >= 3)
                send[i * col + 2] = arr1[(Rank - startRank) * col + i * col * (endRank -
startRank) + 2];
            if (col >= 4)
                send[i * col + 3] = arr1[(Rank - startRank) * col + i * col * (endRank -
startRank) + 3];
            if (col >= 5)
                send[i * col + 4] = arr1[(Rank - startRank) * col + i * col * (endRank -
startRank) + 4];
        }
    }
    double** tmp_out = new double* [endRank];
    for (int i = 0; i < endRank; i++)
    {
        tmp_out[i] = new double[n * col];
    }
    if (Rank == startRank)
    {
        tmp_out[Rank] = send;
        for (int i = startRank + 1; i < endRank; i++)
        {
            MPI_Recv(&tmp_out[i], n * col, MPI_DOUBLE, i, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

        }
```

```
        for (int i = startRank + 1; i < endRank; i++)
        {
            MPI_Send(&tmp_out, endRank, MPI_DOUBLE, i, 0, MPI_COMM_WORLD);
        }
    }
    else
    {
        MPI_Send(&send, n * col, MPI_DOUBLE, startRank, 0, MPI_COMM_WORLD);
        MPI_Recv(&tmp_out, endRank, MPI_DOUBLE, startRank, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

    }
    for (int i = startRank; i < endRank; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if ((i - startRank) + j * (endRank - startRank) < N / col)
            {
                if (col >= 1)
                    arr1[(i - startRank) * col + j * col * (endRank - startRank) + 0] =
tmp_out[i][j * col + 0];
                if (col >= 2)
                    arr1[(i - startRank) * col + j * col * (endRank - startRank) + 1] =
tmp_out[i][j * col + 1];
                if (col >= 3)
                    arr1[(i - startRank) * col + j * col * (endRank - startRank) + 2] =
tmp_out[i][j * col + 2];
                if (col >= 4)
                    arr1[(i - startRank) * col + j * col * (endRank - startRank) + 3] =
tmp_out[i][j * col + 3];
                if (col >= 5)
                    arr1[(i - startRank) * col + j * col * (endRank - startRank) + 4] =
tmp_out[i][j * col + 4];
            }
        }
    }
}
```

B.2 Bervin_Masters_RK

```
int main(int argc, char** argv)

{

   setlocale(LC_CTYPE, "ukr");

   MPI_Init(&argc, &argv);

   int rank, size;

   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   MPI_Comm_size(MPI_COMM_WORLD, &size);

   MPI_Comm ring_comm;

   int dims[1] = { size };

   int periods[1] = { 1 };

   int coords[1];
```

```
int end_1_Group = 0;
int end_2_Group = 1;
MPI_Cart_create(MPI_COMM_WORLD, 1, dims, periods, 0, &ring_comm);
MPI_Cart_coords(ring_comm, rank, 1, coords);
int left_neighbor, right_neighbor;
MPI_Cart_shift(ring_comm, 0, 1, &left_neighbor, &right_neighbor);
int nstep = 0;
int badstep = 0;
int maxnr = 0;
int s_1 = 2;
int m_1 = 2;
int s_2 = 2 * s_1;
int m_2 = m_1;
int AlphaQ_ = 100;
double tau_ = 0.001;
double start, end, time, nrvsp;
double* X = new double[2]{ 1, 1 };
   int koef_weight = 0;
   string an_mode = "1";
   int inerc_stepUp = 30;
   int eps_stepUp = 100;
   double tau_stepUp = 0.05;
   if (rank == 0)
   {
      cout << "Альфа[1-100]: ";
      cin >> AlphaQ_;
      cout << "Параметр інерційності [1-15]: ";
      cin >> inerc_stepUp;
      cout << "Коефіцієнт похибки [10, 100]: ";
      cin >> eps_stepUp;
      cout << "Параметр розходження [0.001 - 0.5]: ";
      cin >> tau_stepUp;
```

```
    cout << "Складність: ";
    cin >> koef_weight;
    if (size / 2 <= max(s_1, m_1))
    {
        end_1_Group = (size / 2);

        if ((size / 2) <= max(s_2, m_1))
        {
            end_2_Group = (size / 2) * 2;
        }
        else {
            end_2_Group = max(s_2, m_1);
        }
    }
    else
    {
        end_1_Group = max(s_1, m_1);
        if (max(s_2, m_1) < (size - (max(s_1, m_1))))
        {
            end_2_Group = ((max(s_1, m_1)) + max(s_2, m_1));
        }
        else {
            end_2_Group = size;
        }
    }
}
MPI_Bcast(&end_1_Group, 1, MPI_INT, 0, ring_comm);
MPI_Bcast(&end_2_Group, 1, MPI_INT, 0, ring_comm);
MPI_Bcast(&koef_weight, 1, MPI_INT, 0, ring_comm);
MPI_Bcast(&inerc_stepUp, 1, MPI_INT, 0, ring_comm);
MPI_Bcast(&eps_stepUp, 1, MPI_INT, 0, ring_comm);
MPI_Bcast(&tau_stepUp, 1, MPI_DOUBLE, 0, ring_comm);
```

```
if (rank < end_2_Group)
{
   double Eps = 0.00000001; //e-8
   int t0 = 0;
   double tau = tau_;
   double tau_1 = tau;
   double tau_2 = tau_1 / 2;
   int s_2 = s_1 * 2;
   int m_2 = m_1;
   int Iter = s_2 * 2;
   int tend = 10;
   int m = m_1;
   double* w = new double[(s_1 * 5)];
   double* W = new double[(s_1 * 5)];
   double* wvsp = new double[s_2 * 5];
   double tvsp;
   int tn = t0;
   double t;
   double* Y = new double[2];
      start = omp_get_wtime();
      if (rank < end_1_Group)
      {
         for (int i = rank; i < s_1; i += end_1_Group)
         {
            t = t0 + (i)*tau_1;
            Y = Y_(t);
            w[i * 5 + 0] = t;
            w[i * 5 + 1] = Y[0];
            w[i * 5 + 2] = Y[1];
            w[i * 5 + 3] = 0;
            w[i * 5 + 4] = 0;
         }
```

```
    Function.sync(ring_comm, rank, 0, end_1_Group, w, s_2 * 5, 5);
    W = w;
}
if (rank >= end_1_Group && rank < end_2_Group)
{
    for (int i = rank - end_1_Group; i < s_2; i += (end_2_Group - end_1_Group))
    {
        tvsp = t0 + (i)*tau_2;
        Y = Function.Y_(tvsp);
        wvsp[i * 5 + 0] = tvsp;
        wvsp[i * 5 + 1] = Y[0];
        wvsp[i * 5 + 2] = Y[1];
        wvsp[i * 5 + 3] = 0;
        wvsp[i * 5 + 4] = 0;
    }
    Function.sync(ring_comm, rank, end_1_Group, end_2_Group, wvsp, s_2 * 5, 5);
    double* Wvsp = wvsp;
}
double* U = new double[m_1 * 2];
double* U1 = new double[s_1 * 2];
double* U2 = new double[s_2 * 2];
double* Array = new double[s_1 * 2];
int k_ = 1, j_ = 0;
if (rank == 0)
{
    for (int i = 0; i < m_1 * 5; i++)
    {
        if (k_ >= 2 && k_ <= 3 && i != 0)
        {
            U[j_] = w[i];
            j_++;
        }
```

```
      k_++;
      if (k_ == 6)
        k_ = 1;
    }
  }
  Function.send(ring_comm, rank, U, m_1*2, 0, 0, end_2_Group);
  double* Uvsp = new double[(s_2 - s_1) * 2];
  if (rank == end_1_Group)
  {
    k_ = 1;
    j_ = 0;
    for (int i = (s_1) * 5; i < s_2 * 5; i++)
    {
      if (k_ >= 2 && k_ <= 3 && i != 0)
      {
        Uvsp[j_] = wvsp[i];
        j_++;
      }
      k_++;
      if (k_ == 6)
        k_ = 1;
    }
  }
  Function.send(ring_comm, rank, Uvsp, (s_2 - s_1) * 2, 0, end_1_Group, end_2_Group);
  int fl0, fl1, fl2, fl3, fl4;
  double mark;
  int nr = 0;
  tn = t0 + tau * (s_1 - 1);
  fl4 = 0;
  int NIter = 0;
  int inerc = 0;
  int r1 = s_1;
```

```
mark = 1;
double* Fi00 = 0, *Fi01 = 0, *Fi10 = 0, *Fi11 = 0, *Fi20 = 0, *Fi22 = 0;
while (tn <= tend)
{
   fl0 = 0;
   fl1 = 0;
   fl2 = 0;
   fl3 = 0;
   mark = 1;
   if (rank == 0)
      Fi00 = Function.genB(22, 0, s_1);
   if (rank <= 1)
      Fi10 = Function.genB(222, 0, s_1);
   if (rank == 0)
      Fi20 = Function.genB(122, 0, s_1);
   if (rank == end_1_Group)
      Fi01 = Function.genB(24, 0, s_1);
   if (rank >= end_1_Group && rank <= end_1_Group + 1)
      Fi11 = Function.genB(224, 0, s_1);
   if (rank == end_1_Group)
      Fi22 = Function.genB(124, 0, s_1);
   if (fl4 == 1)
   {
      mark = 2;
      if (rank == 0)
         Fi00 = Function.genB(22, 1, s_1);
      if (rank <= 1)
         Fi10 = Function.genB(222, 1, s_1);
      if (rank == 0)
         Fi20 = Function.genB(122, 1, s_1);
      if (rank == end_1_Group)
         Fi01 = Function.genB(24, 1, s_1);
```

```
    if (rank >= end_1_Group && rank <= end_1_Group + 1)
        Fi11 = Function.genB(224, 1, s_1);
    if (rank == end_1_Group)
        Fi22 = Function.genB(124, 1, s_1);
    fl4 = 0;
}
while (fl2 == 0 && fl1 <= 1 && fl3 == 0)
{
    nstep += 1;
    double *F1_ = new double[m * 2];
    double *u = new double[2];
    if (rank < end_1_Group)
    {
        for (int i = rank; i < m; i += end_1_Group)
        {
            u[0] = U[i * 2];
            u[1] = U[i * 2 + 1];
            u = Function.F_(AlphaQ_, (tn + (i - m) * tau), u, koef_weight);
            F1_[i * 2] = u[0];
            F1_[i * 2 + 1] = u[1];
        }
        Function.sync(ring_comm, rank, 0, end_1_Group, F1_, m * 2, 2);
    }
    Function.send(ring_comm, rank, F1_, m*2, 0, 0, end_2_Group);
    double* u_copy;
    u_copy = U;
    double* U_tmp;
    double *V1 = new double[s_1 * 2];
    double *R1 = new double[s_1 * 2];
    Array = U;
    if (rank < end_1_Group)
    {
```

```
                U_tmp = new double[s_1 * 2]; ;
                for (int i_ = 0; i_ < s_1 * 2; i_ += 2)
                {
                    U_tmp[i_] = u_copy[s_1 * 2 - 2];
                    U_tmp[i_ + 1] = u_copy[s_1 * 2 - 1];
                }
                if (end_1_Group > 1)
                {
                    if (rank == 0)
                    {
                        V1 = Function.MatrixPlus(U_tmp, s_1 * 2,
Function.MatrixMult(Function.MatrixMult(Fi00, s_1 * 2, F1_, s_1, m, m, 2), s_1 * 2, tau));
                    }
                    if (rank == 1)
                    {
                        R1 = Function.MatrixPlus(U_tmp, s_1 * 2,
Function.MatrixMult(Function.MatrixMult(Fi10, s_1 * 2, F1_, s_1, m, m, 2), s_1 * 2, tau));
                    }
                    Function.send (ring_comm, rank, V1, s_1 * 2, 0, 0, end_1_Group);
                    send_data_in_group(ring_comm, rank, R1, s_1 * 2, 0, 1, end_1_Group);
                }
                else
                {
                    V1 = Function.MatrixPlus(U_tmp, s_1 * 2,
Function.MatrixMult(Function.MatrixMult(Fi00, s_1 * 2, F1_, s_1, m, m, 2), s_1 * 2, tau));
                    R1 = Function.MatrixPlus(U_tmp, s_1 * 2,
Function.MatrixMult(Function.MatrixMult(Fi10, s_1 * 2, F1_, s_1, m, m, 2), s_1 * 2, tau));
                }
            }
            double *V2 = new double[s_2 * 2];
            double *R2 = new double[s_2 * 2];
            if (rank >= end_1_Group && rank < end_2_Group)
            {
```

```
                U_tmp = new double[s_2 * 2];
                for (int i_ = 0; i_ < s_2 * 2; i_ += 2)
                {
                    U_tmp[i_] = u_copy[s_2 * 2 - 2];
                    U_tmp[i_ + 1] = u_copy[s_2 * 2 - 1];
                }
                if ((end_2_Group - end_1_Group) > 1)
                {
                    if (rank == end_1_Group)
                        V2 = Function.MatrixPlus(U_tmp, s_2 * 2,
Function.MatrixMult(Function.MatrixMult(Fi01, s_2 * 2, F1_, s_2, m, m, 2), s_2 * 2, tau));
                    if (rank == end_1_Group + 1)
                        R2 = Function.MatrixPlus(U_tmp, s_2 * 2,
Function.MatrixMult(Function.MatrixMult(Fi11, s_2 * 2, F1_, s_2, m, m, 2), s_2 * 2, tau));
                    Function.send(ring_comm, rank, V2, s_1 * 2, end_1_Group, end_1_Group,
end_2_Group);
                    Function.send(ring_comm, rank, R2, s_1 * 2, end_1_Group, end_1_Group + 1,
end_2_Group);
                }
                else
                {
                    V2 = Function.MatrixPlus(U_tmp, s_2 * 2,
Function.MatrixMult(Function.MatrixMult(Fi01, s_2 * 2, F1_, s_2, m, m, 2), s_2 * 2, tau));
                    R2 = Function.MatrixPlus(U_tmp, s_2 * 2,
Function.MatrixMult(Function.MatrixMult(Fi11, s_2 * 2, F1_, s_2, m, m, 2), s_2 * 2, tau));
                }
            }
            nr = 10;
            int p = 0;
            double tn_1 = tn + tau_1 * mark;
            double tn_2 = tn - tau_2 * mark;
            while (p < s_2 || (nr > Eps && p <= Iter))
            {
                double *v = new double[2];
```

```
if (rank < end_1_Group)
{
   double *G1 = new double[s_1 * 2];
   for (int i = rank; i < s_1; i += end_1_Group)
   {
      v[0] = V1[i * 2];
      v[1] = V1[i * 2 + 1];
      v = Function.F_(AlphaQ_, 1, v, koef_weight);
      G1[i * 2] = v[0];
      G1[i * 2 + 1] = v[1];
   }
   Function.sync(ring_comm, rank, 0, end_1_Group, G1, s_1 * 2, 2);
   if (rank == 0)
   {
      V1 = Function.MatrixPlus(R1, s_1 * 2,
Function.MatrixMult(Function.MatrixMult(Fi20, s_1 * 2, G1, s_1, s_1, s_1, 2), s_1 * 2, tau_1));
      U1 = V1;
   }
   Function.send (ring_comm, rank, V1, s_1 * 2, 0, 0, end_1_Group);
}
Function.send(ring_comm, rank, U1, s_1 * 2, 0, 0, end_2_Group);
if (rank >= end_1_Group && rank < end_2_Group)
{
   double *G2 = new double[s_2 * 2];
   for (int i = rank - end_1_Group; i < s_2; i += (end_2_Group - end_1_Group))
   {
      v[0] = V2[i * 2];
      v[1] = V2[i * 2 + 1];
      v = Function.F_(AlphaQ_, 1, v, koef_weight);
      G2[i * 2] = v[0];
      G2[i * 2 + 1] = v[1];
   }
```

```
                Function.sync(ring_comm, rank, end_1_Group, end_2_Group, G2, s_2 * 2, 2);
                if (rank == end_1_Group)
                {
                    V2 = Function.MatrixPlus(R2, s_2 * 2,
Function.MatrixMult(Function.MatrixMult(Fi22, s_2 * 2, G2, s_2, s_2, s_2, 2), s_2 * 2, tau_1));
                    U2 = V2;
                }
                Function.send (ring_comm, rank, V2, s_1 * 2, end_1_Group, end_1_Group,
end_2_Group);
            }
            Function.send (ring_comm, rank, U2, s_1 * 2, 0, end_1_Group, end_2_Group);
            nr = 0.0;
            p += 1;
            double *nr_arr = new double[s_1 * 2];
            if ((s_1 - end_1_Group) < (end_2_Group - end_1_Group))
            {
                if (rank < end_1_Group)
                {
                    for (int i = rank; i < s_1; i += end_1_Group)
                    {
                        double *res = ArraySubtraction(U1, U2, s_1, 2 * i);
                            nr_arr[i * 2 + 0] = abs(res[0]);
                            nr_arr[i * 2 + 1] = abs(res[1]);
                    }
                    Function.sync(ring_comm, rank, 0, end_1_Group, nr_arr, s_1 * 2, 2);
                    if (rank == 0)
                    {
                        nr - nr_arr[0];
                        for (int i = 1; i < s_1 * 2; i++)
                            if (nr_arr[i] > nr)
                                nr = nr_arr[i];
                    }
                }
```

```
                }
                else if ((s_1 - end_1_Group) >= (end_2_Group - end_1_Group))
                {
                    if (rank < end_1_Group)
                    {
                        for (int i = rank; i < s_1 / 2; i += end_1_Group)
                        {
                            double *res = Function.ArraySubtraction(U1, U2, s_1, 2 * i);
                            nr_arr[i * 2 + 0] = abs(res[0]);
                            nr_arr[i * 2 + 1] = abs(res[1]);
                        }
                    }
                    if (rank >= end_1_Group)
                    {
                        for (int i = rank + (s_1 / 2 - end_1_Group); i < s_1; i += (end_2_Group -
end_1_Group))
                        {
                            double *res = Function.ArraySubtraction(U1, U2, s_1, 2 * i);
                            nr_arr[i * 2 + 0] = abs(res[0]);
                            nr_arr[i * 2 + 1] = abs(res[1]);
                        }
                    }
                    Function.sync(ring_comm, rank, 0, end_2_Group, nr_arr, s_1 * 2, 2);
                    if (rank == 0)
                    {
                        nr - nr_arr[0];
                        for (int i = 1; i < s_1 * 2; i++)
                            if (nr_arr[i] > nr)
                                nr = nr_arr[i];
                    }
                }
                Function.send(ring_comm, rank, nr_arr, s_1 * 2, 0, 0, end_2_Group);
```

```
    if (rank == 0)
    {
        nr - nr_arr[0];
        for (int i = 1; i < s_1 * 2; i++)
            if (nr_arr[i] > nr)
                nr = nr_arr[i];
    }
}
if (p > NIter)
    NIter = p;
if (p > Iter && nr > Eps)
{
    fl2 = 1;
}
else
{
    fl3 = 1;
    inerc += 1;
}
if (fl2 == 1 && fl1 == 0)
{
    mark = mark / 2.0;
    if (rank == 0)
        Fi00 = Function.genB(22, -1, s_1);
    if (rank <= 1)
        Fi10 = Function.genB(222, -1, s_1);
    if (rank == 0)
        Fi20 = Function.genB(122, -1, s_1);
    if (rank == end_1_Group)
        Fi01 = Function.genB(24, -1, s_1);
    if (rank >= end_1_Group && rank <= end_1_Group + 1)
        Fi11 = Function.genB(224, -1, s_1);
```

```
    if (rank == end_1_Group)
        Fi22 = Function.genB(124, -1, s_1);
    fl1++;
    fl2 = 0;
    inerc = 0;
    badstep++;
}
else if (fl2 == 1 && fl1 == 1 && fl0 == 0)
{
    if (rank == 0)
        Fi00 = Function.genB(22, 0, s_1);
    if (rank <= 1)
        Fi10 = Function.genB(222, 0, s_1);
    if (rank == 0)
        Fi20 = Function.genB(122, 0, s_1);
    if (rank == end_1_Group)
        Fi01 = Function.genB(24, 0, s_1);
    if (rank >= end_1_Group && rank <= end_1_Group + 1)
        Fi11 = Function.genB(224, 0, s_1);
    if (rank == end_1_Group)
        Fi22 = Function.genB(124, 0, s_1);
    mark = 1.0;
    tau /= 2;
    tau_1 = tau;
    tau_2 = tau_1 / 2;
    U = Uvsp;
    fl0 = 1;
    fl1 = 0;
    fl2 = 0;
    inerc = 0;
    badstep++;
}
```

```
}
double *w_old = w;
Array = w;
double *nrvsp_arr = new double[s_1 * 2];
if (rank == 0)
{
   for (int i = 0; i < s_1; i++)
   {
      w[(r1 * 5) + (i * 5) + 0] = tn + (i + 1) * (s_1 - 1) * tau_1 * mark;
      w[(r1 * 5) + (i * 5) + 1] = U2[(2 * 2) * (i + 1) - 2 + 0];
      w[(r1 * 5) + (i * 5) + 2] = U2[(2 * 2) * (i + 1) - 2 + 1];
      w[(r1 * 5) + (i * 5) + 3] = (U1[i * 2 + 0] - U2[(2 * 2) * (i + 1) - 2 + 0]);
      w[(r1 * 5) + (i * 5) + 4] = (U1[i * 2 + 1] - U2[(2 * 2) * (i + 1) - 2 + 1]);
   }
}
if (rank == end_1_Group)
{
   for (int i = 0; i < s_1; i++)
   {
      nrvsp = abs(max((U1[i * 2 + 0] - U2[(2 * 2) * (i + 1) - 2 + 0]), (U1[i * 2 + 1] - U2[(2 * 2) * (i +
1) - 2 + 1])));
      nrvsp_arr[i * 2] = nrvsp;
      nrvsp_arr[i * 2 + 1] = 0;
      U[i * 2 + 0] = U2[(2 * 2) * (i + 1) - 2 + 0];
      U[i * 2 + 1] = U2[(2 * 2) * (i + 1) - 2 + 1];
      Uvsp[i * 2 + 0] = U2[(s_1 * 2) + i * 2 + 0];
      Uvsp[i * 2 + 1] = U2[(s_1 * 2) + i * 2 + 1];
   }
   maxnr - nrvsp_arr[0];
   for (int i = 1; i < s_1 * 2; i++)
      if (nrvsp_arr[i] > maxnr)
         maxnr = nrvsp_arr[i];
```

```
            }
            Function.send(ring_comm, rank, U, m_1 * 2, 0, end_1_Group, end_2_Group);
            Function.send(ring_comm, rank, Uvsp, (s_2 - s_1) * 2, 0, end_1_Group, end_2_Group);
            Function.send(ring_comm, rank, nrvsp_arr, s_1 * 2, 0, end_1_Group, end_2_Group);
            maxnr - nrvsp_arr[0];
            for (int i = 1; i < s_1 * 2; i++)
               if (nrvsp_arr[i] > maxnr)
                  maxnr = nrvsp_arr[i];
            r1 += s_1;
            tn = tn + s_1 * tau_1 * mark;
            tau_1 *= mark;
            tau_2 *= mark;
            tau = tau_1;
            if (inerc > inerc_stepUp && nr < (eps_stepUp * Eps) && tau_1 < tau_stepUp)
            {
               inerc = 0;
               fl3 = 1;
               fl4 = 1;
            }
         }
         if (rank == 0)
         {
            end = omp_get_wtime();
            time = end - start;
         }
      }
      if (rank == 0)
      {
         cout << "Запис у файл...";
         ofstream out;
         string name = "Result(" + to_string(size) + ")_"+ to_string(time) + "_BM).txt";
         out.open(name);
```

```
        if (out.is_open())
        {
           out << "Режим: " << "БМ" << endl << " Похибка: " << maxnr << endl << "Час: " << time << <<
endl "Погані кроки: " << badstep << endl << "Всього кроків: " << nstep;
        }
        out.close();
        cout << "Закінчено запис";
    }
    if (rank == 0)
        cout << "Робота закінчена";
    return 0;
  }
```

B.3 Bervin_Masters_Block

```
int main(int argc, char** argv)
{
    setlocale(LC_CTYPE, "ukr");
    MPI_Init(&argc, &argv);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm ring_comm;
    int dims[1] = { size };
    int periods[1] = { 1 };
    int coords[1];
    int end_1_Group = 0;
    int end_2_Group = 1;
    MPI_Cart_create(MPI_COMM_WORLD, 1, dims, periods, 0, &ring_comm);
    MPI_Cart_coords(ring_comm, rank, 1, coords);
    int left_neighbor, right_neighbor;
    MPI_Cart_shift(ring_comm, 0, 1, &left_neighbor, &right_neighbor);
    int nstep = 0;
    int badstep = 0;
    int maxnr = 0;
    int s_1 = 2;
    int m_1 = 2;
    int s_2 = 2 * s_1;
    int m_2 = m_1;
    int AlphaQ_ = 100;
    double tau_ = 0.001;
    double start, end, time, nrvsp;
    double* X = new double[2]{ 1, 1 };
        int koef_weight = 0;
        string an_mode = "1";
        int inerc_stepUp = 30;
```

```
int eps_stepUp = 100;
double tau_stepUp = 0.05;
if (rank == 0)
{
   cout << "Альфа[1-100]: ";
   cin >> AlphaQ_;
   cout << "Параметр інерційності [1-15]: ";
   cin >> inerc_stepUp;
   cout << "Коефіцієнт похибки [10, 100]: ";
   cin >> eps_stepUp;
   cout << "Параметр розходження [0.001 - 0.5]: ";
   cin >> tau_stepUp;
   cout << "Складність: ";
   cin >> koef_weight;
   if (size / 2 <= max(s_1, m_1))
   {
      end_1_Group = (size / 2);

      if ((size / 2) <= max(s_2, m_1))
      {
         end_2_Group = (size / 2) * 2;
      }
      else {
         end_2_Group = max(s_2, m_1);
      }
   }
   else
   {
      end_1_Group = max(s_1, m_1);
      if (max(s_2, m_1) < (size - (max(s_1, m_1))))
      {
         end_2_Group = ((max(s_1, m_1)) + max(s_2, m_1));
```

```
      }
      else {
         end_2_Group = size;
      }
   }
}
MPI_Bcast(&end_1_Group, 1, MPI_INT, 0, ring_comm);
MPI_Bcast(&end_2_Group, 1, MPI_INT, 0, ring_comm);
MPI_Bcast(&koef_weight, 1, MPI_INT, 0, ring_comm);
MPI_Bcast(&inerc_stepUp, 1, MPI_INT, 0, ring_comm);
MPI_Bcast(&eps_stepUp, 1, MPI_INT, 0, ring_comm);
MPI_Bcast(&tau_stepUp, 1, MPI_DOUBLE, 0, ring_comm);
if (rank < end_2_Group)
{
   double Eps = 0.00000001; //e-8
   int t0 = 0;
   double tau = tau_;
   double tau_1 = tau;
   double tau_2 = tau_1 / 2;
   int s_2 = s_1 * 2;
   int m_2 = m_1;
   int Iter = s_2 * 2;
   int tend = 10;
   int m = m_1;
   double* w = new double[(s_1 * 5)];
   double* W = new double[(s_1 * 5)];
   double* wvsp = new double[s_2 * 5];
   double tvsp;
   int tn = t0;
   double t;
   double* Y = new double[2];
      start = omp_get_wtime();
```

```
if (rank < end_1_Group)
{
   for (int i = rank; i < s_1; i += end_1_Group)
   {
      t = t0 + (i)*tau_1;
      Y = Y_(t);
      w[i * 5 + 0] = t;
      w[i * 5 + 1] = Y[0];
      w[i * 5 + 2] = Y[1];
      w[i * 5 + 3] = 0;
      w[i * 5 + 4] = 0;
   }
   Function.sync(ring_comm, rank, 0, end_1_Group, w, s_2 * 5, 5);
   W = w;
}
if (rank >= end_1_Group && rank < end_2_Group)
{
   for (int i = rank - end_1_Group; i < s_2; i += (end_2_Group - end_1_Group))
   {
      tvsp = t0 + (i)*tau_2;
      Y = Function.Y_(tvsp);
      wvsp[i * 5 + 0] = tvsp;
      wvsp[i * 5 + 1] = Y[0];
      wvsp[i * 5 + 2] = Y[1];
      wvsp[i * 5 + 3] = 0;
      wvsp[i * 5 + 4] = 0;
   }
   Function.sync(ring_comm, rank, end_1_Group, end_2_Group, wvsp, s_2 * 5, 5);
   double* Wvsp = wvsp;
}
double* U = new double[m_1 * 2];
double* U1 = new double[s_1 * 2];
```

```
double* U2 = new double[s_2 * 2];
double* Array = new double[s_1 * 2];
int k_ = 1, j_ = 0;
if (rank == 0)
{
   for (int i = 0; i < m_1 * 5; i++)
   {
      if (k_ >= 2 && k_ <= 3 && i != 0)
      {
         U[j_] = w[i];
         j_++;
      }
      k_++;
      if (k_ == 6)
         k_ = 1;
   }
}
Function.send(ring_comm, rank, U, m_1*2, 0, 0, end_2_Group);
double* Uvsp = new double[(s_2 - s_1) * 2];
if (rank == end_1_Group)
{
   k_ = 1;
   j_ = 0;
   for (int i = (s_1) * 5; i < s_2 * 5; i++)
   {
      if (k_ >= 2 && k_ <= 3 && i != 0)
      {
         Uvsp[j_] = wvsp[i];
         j_++;
      }
      k_++;
      if (k_ == 6)
```

```
            k_ = 1;
        }
    }
    Function.send(ring_comm, rank, Uvsp, (s_2 - s_1) * 2, 0, end_1_Group, end_2_Group);
    int fl0, fl1, fl2, fl3, fl4;
    double mark;
    int nr = 0;
    tn = t0 + tau * (s_1 - 1);
    fl4 = 0;
    int NIter = 0;
    int inerc = 0;
    int r1 = s_1;
    mark = 1;
    double* Fi00 = 0, *Fi01 = 0, *Fi10 = 0, *Fi11 = 0, *Fi20 = 0, *Fi22 = 0;
    while (tn <= tend)
    {
        fl0 = 0;
        fl1 = 0;
        fl2 = 0;
        fl3 = 0;
        mark = 1;
        if (rank == 0)
            Fi00 = Function.genB(22, 0, s_1);
        if (rank <= 1)
            Fi10 = Function.genB(222, 0, s_1);
        if (rank == 0)
            Fi20 = Function.genB(122, 0, s_1);
        if (rank == end_1_Group)
            Fi01 = Function.genB(24, 0, s_1);
        if (rank >= end_1_Group && rank <= end_1_Group + 1)
            Fi11 = Function.genB(224, 0, s_1);
        if (rank == end_1_Group)
```

```
    Fi22 = Function.genB(124, 0, s_1);
if (fl4 == 1)
{
    mark = 2;
    if (rank == 0)
        Fi00 = Function.genB(22, 1, s_1);
    if (rank <= 1)
        Fi10 = Function.genB(222, 1, s_1);
    if (rank == 0)
        Fi20 = Function.genB(122, 1, s_1);
    if (rank == end_1_Group)
        Fi01 = Function.genB(24, 1, s_1);
    if (rank >= end_1_Group && rank <= end_1_Group + 1)
        Fi11 = Function.genB(224, 1, s_1);
    if (rank == end_1_Group)
        Fi22 = Function.genB(124, 1, s_1);
    fl4 = 0;
}
while (fl2 == 0 && fl1 <= 1 && fl3 == 0)
{
    nstep += 1;
    double *F1_ = new double[m * 2];
    double *u = new double[2];
    if (rank < end_1_Group)
    {
        for (int i = rank; i < m; i += end_1_Group)
        {
            u[0] = U[i * 2];
            u[1] = U[i * 2 + 1];
            u = Function.F_(AlphaQ_, (tn + (i - m) * tau), u, koef_weight);
            F1_[i * 2] = u[0];
            F1_[i * 2 + 1] = u[1];
```

```
            }
            Function.sync(ring_comm, rank, 0, end_1_Group, F1_, m * 2, 2);
        }
        Function.send(ring_comm, rank, F1_, m*2, 0, 0, end_2_Group);
        double* u_copy;
        u_copy = U;
        double* U_tmp;
        double *V1 = new double[s_1 * 2];
        double *R1 = new double[s_1 * 2];
        Array = U;
        if (rank < end_1_Group)
        {
            U_tmp = new double[s_1 * 2]; ;
            for (int i_ = 0; i_ < s_1 * 2; i_ += 2)
            {
                U_tmp[i_] = u_copy[s_1 * 2 - 2];
                U_tmp[i_ + 1] = u_copy[s_1 * 2 - 1];
            }
            if (end_1_Group > 1)
            {
                if (rank == 0)
                {
                    V1 = Function.MatrixPlus(U_tmp, s_1 * 2,
Function.MatrixMult(Function.MatrixMult(Fi00, s_1 * 2, F1_, s_1, m, m, 2), s_1 * 2, tau));
                }
                if (rank == 1)
                {
                    R1 = Function.MatrixPlus(U_tmp, s_1 * 2,
Function.MatrixMult(Function.MatrixMult(Fi10, s_1 * 2, F1_, s_1, m, m, 2), s_1 * 2, tau));
                }
                Function.send (ring_comm, rank, V1, s_1 * 2, 0, 0, end_1_Group);
                send_data_in_group(ring_comm, rank, R1, s_1 * 2, 0, 1, end_1_Group);
            }
```

```
                else
                {
                    V1 = Function.MatrixPlus(U_tmp, s_1 * 2,
Function.MatrixMult(Function.MatrixMult(Fi00, s_1 * 2, F1_, s_1, m, m, 2), s_1 * 2, tau));
                    R1 = Function.MatrixPlus(U_tmp, s_1 * 2,
Function.MatrixMult(Function.MatrixMult(Fi10, s_1 * 2, F1_, s_1, m, m, 2), s_1 * 2, tau));
                }
            }
            double *V2 = new double[s_2 * 2];
            double *R2 = new double[s_2 * 2];
            if (rank >= end_1_Group && rank < end_2_Group)
            {
                U_tmp = new double[s_2 * 2];
                for (int i_ = 0; i_ < s_2 * 2; i_ += 2)
                {
                    U_tmp[i_] = u_copy[s_2 * 2 - 2];
                    U_tmp[i_ + 1] = u_copy[s_2 * 2 - 1];
                }
                if ((end_2_Group - end_1_Group) > 1)
                {
                    if (rank == end_1_Group)
                        V2 = Function.MatrixPlus(U_tmp, s_2 * 2,
Function.MatrixMult(Function.MatrixMult(Fi01, s_2 * 2, F1_, s_2, m, m, 2), s_2 * 2, tau));
                    if (rank == end_1_Group + 1)
                        R2 = Function.MatrixPlus(U_tmp, s_2 * 2,
Function.MatrixMult(Function.MatrixMult(Fi11, s_2 * 2, F1_, s_2, m, m, 2), s_2 * 2, tau));
                    Function.send(ring_comm, rank, V2, s_1 * 2, end_1_Group, end_1_Group,
end_2_Group);
                    Function.send(ring_comm, rank, R2, s_1 * 2, end_1_Group, end_1_Group + 1,
end_2_Group);
                }
                else
                {
```

```
                V2 = Function.MatrixPlus(U_tmp, s_2 * 2,
Function.MatrixMult(Function.MatrixMult(Fi01, s_2 * 2, F1_, s_2, m, m, 2), s_2 * 2, tau));
                R2 = Function.MatrixPlus(U_tmp, s_2 * 2,
Function.MatrixMult(Function.MatrixMult(Fi11, s_2 * 2, F1_, s_2, m, m, 2), s_2 * 2, tau));
              }
            }
            nr = 10;
            int p = 0;
            double tn_1 = tn + tau_1 * mark;
            double tn_2 = tn - tau_2 * mark;
            while (p < s_2 || (nr > Eps && p <= Iter))
            {
              double *v = new double[2];
              if (rank < end_1_Group)
              {
                double *G1 = new double[s_1 * 2];
                for (int i = rank; i < s_1; i += end_1_Group)
                {
                  v[0] = V1[i * 2];
                  v[1] = V1[i * 2 + 1];
                  v = Function.F_(AlphaQ_, 1, v, koef_weight);
                  G1[i * 2] = v[0];
                  G1[i * 2 + 1] = v[1];
                }
                Function.sync(ring_comm, rank, 0, end_1_Group, G1, s_1 * 2, 2);
                if (rank == 0)
                {
                  V1 = Function.MatrixPlus(R1, s_1 * 2,
Function.MatrixMult(Function.MatrixMult(Fi20, s_1 * 2, G1, s_1, s_1, s_1, 2), s_1 * 2, tau_1));
                  U1 = V1;
                }
                Function.send (ring_comm, rank, V1, s_1 * 2, 0, 0, end_1_Group);
              }
```

```
                Function.send(ring_comm, rank, U1, s_1 * 2, 0, 0, end_2_Group);
                if (rank >= end_1_Group && rank < end_2_Group)
                {
                   double *G2 = new double[s_2 * 2];
                   for (int i = rank - end_1_Group; i < s_2; i += (end_2_Group - end_1_Group))
                   {
                      v[0] = V2[i * 2];
                      v[1] = V2[i * 2 + 1];
                      v = Function.F_(AlphaQ_, 1, v, koef_weight);
                      G2[i * 2] = v[0];
                      G2[i * 2 + 1] = v[1];
                   }
                   Function.sync(ring_comm, rank, end_1_Group, end_2_Group, G2, s_2 * 2, 2);
                   if (rank == end_1_Group)
                   {
                      V2 = Function.MatrixPlus(R2, s_2 * 2,
Function.MatrixMult(Function.MatrixMult(Fi22, s_2 * 2, G2, s_2, s_2, s_2, 2), s_2 * 2, tau_1));
                      U2 = V2;
                   }
                   Function.send (ring_comm, rank, V2, s_1 * 2, end_1_Group, end_1_Group,
end_2_Group);
                }
                Function.send (ring_comm, rank, U2, s_1 * 2, 0, end_1_Group, end_2_Group);
                nr = 0.0;
                p += 1;
                double *nr_arr = new double[s_1 * 2];
                if ((s_1 - end_1_Group) < (end_2_Group - end_1_Group))
                {
                   if (rank < end_1_Group)
                   {
                      for (int i = rank; i < s_1; i += end_1_Group)
                      {
                         double *res = ArraySubtraction(U1, U2, s_1, 2 * i);
```

```
                    nr_arr[i * 2 + 0] = abs(res[0]);
                    nr_arr[i * 2 + 1] = abs(res[1]);
                }
                Function.sync(ring_comm, rank, 0, end_1_Group, nr_arr, s_1 * 2, 2);
                if (rank == 0)
                {
                    nr - nr_arr[0];
                    for (int i = 1; i < s_1 * 2; i++)
                        if (nr_arr[i] > nr)
                            nr = nr_arr[i];
                }
            }
        }
        else if ((s_1 - end_1_Group) >= (end_2_Group - end_1_Group))
        {
            if (rank < end_1_Group)
            {
                for (int i = rank; i < s_1 / 2; i += end_1_Group)
                {
                    double *res = Function.ArraySubtraction(U1, U2, s_1, 2 * i);
                    nr_arr[i * 2 + 0] = abs(res[0]);
                    nr_arr[i * 2 + 1] = abs(res[1]);
                }
            }
            if (rank >= end_1_Group)
            {
                for (int i = rank + (s_1 / 2 - end_1_Group); i < s_1; i += (end_2_Group -
end_1_Group))
                {
                    double *res = Function.ArraySubtraction(U1, U2, s_1, 2 * i);
                    nr_arr[i * 2 + 0] = abs(res[0]);
                    nr_arr[i * 2 + 1] = abs(res[1]);
```

```
            }
        }
        Function.sync(ring_comm, rank, 0, end_2_Group, nr_arr, s_1 * 2, 2);
        if (rank == 0)
        {
            nr - nr_arr[0];
            for (int i = 1; i < s_1 * 2; i++)
                if (nr_arr[i] > nr)
                    nr = nr_arr[i];
        }
    }
    Function.send(ring_comm, rank, nr_arr, s_1 * 2, 0, 0, end_2_Group);
    if (rank == 0)
    {
        nr - nr_arr[0];
        for (int i = 1; i < s_1 * 2; i++)
            if (nr_arr[i] > nr)
                nr = nr_arr[i];
    }
}
if (p > NIter)
    NIter = p;
if (p > Iter && nr > Eps)
{
    fl2 = 1;
}
else
{
    fl3 = 1;
    inerc += 1;
}
if (fl2 == 1 && fl1 == 0)
```

```
{
   mark = mark / 2.0;
   if (rank == 0)
      Fi00 = Function.genB(22, -1, s_1);
   if (rank <= 1)
      Fi10 = Function.genB(222, -1, s_1);
   if (rank == 0)
      Fi20 = Function.genB(122, -1, s_1);
   if (rank == end_1_Group)
      Fi01 = Function.genB(24, -1, s_1);
   if (rank >= end_1_Group && rank <= end_1_Group + 1)
      Fi11 = Function.genB(224, -1, s_1);
   if (rank == end_1_Group)
      Fi22 = Function.genB(124, -1, s_1);
   fl1++;
   fl2 = 0;
   inerc = 0;
   badstep++;
}
else if (fl2 == 1 && fl1 == 1 && fl0 == 0)
{
   if (rank == 0)
      Fi00 = Function.genB(22, 0, s_1);
   if (rank <= 1)
      Fi10 = Function.genB(222, 0, s_1);
   if (rank == 0)
      Fi20 = Function.genB(122, 0, s_1);
   if (rank == end_1_Group)
      Fi01 = Function.genB(24, 0, s_1);
   if (rank >= end_1_Group && rank <= end_1_Group + 1)
      Fi11 = Function.genB(224, 0, s_1);
   if (rank == end_1_Group)
```

```
            Fi22 = Function.genB(124, 0, s_1);
        mark = 1.0;
        tau /= 2;
        tau_1 = tau;
        tau_2 = tau_1 / 2;
        U = Uvsp;
        fl0 = 1;
        fl1 = 0;
        fl2 = 0;
        inerc = 0;
        badstep++;
    }
}
double *w_old = w;
Array = w;
double *nrvsp_arr = new double[s_1 * 2];
if (rank == 0)
{
    for (int i = 0; i < s_1; i++)
    {
        w[(r1 * 5) + (i * 5) + 0] = tn + (i + 1) * (s_1 - 1) * tau_1 * mark;
        w[(r1 * 5) + (i * 5) + 1] = U2[(2 * 2) * (i + 1) - 2 + 0];
        w[(r1 * 5) + (i * 5) + 2] = U2[(2 * 2) * (i + 1) - 2 + 1];
        w[(r1 * 5) + (i * 5) + 3] = (U1[i * 2 + 0] - U2[(2 * 2) * (i + 1) - 2 + 0]);
        w[(r1 * 5) + (i * 5) + 4] = (U1[i * 2 + 1] - U2[(2 * 2) * (i + 1) - 2 + 1]);
    }
}
if (rank == end_1_Group)
{
    for (int i = 0; i < s_1; i++)
    {
```

```
                nrvsp = abs(max((U1[i * 2 + 0] - U2[(2 * 2) * (i + 1) - 2 + 0]), (U1[i * 2 + 1] - U2[(2 * 2) * (i + 1) - 2 + 1])));
                nrvsp_arr[i * 2] = nrvsp;
                nrvsp_arr[i * 2 + 1] = 0;
                U[i * 2 + 0] = U2[(2 * 2) * (i + 1) - 2 + 0];
                U[i * 2 + 1] = U2[(2 * 2) * (i + 1) - 2 + 1];
                Uvsp[i * 2 + 0] = U2[(s_1 * 2) + i * 2 + 0];
                Uvsp[i * 2 + 1] = U2[(s_1 * 2) + i * 2 + 1];
            }
            maxnr - nrvsp_arr[0];
            for (int i = 1; i < s_1 * 2; i++)
                if (nrvsp_arr[i] > maxnr)
                    maxnr = nrvsp_arr[i];
        }
        Function.send(ring_comm, rank, U, m_1 * 2, 0, end_1_Group, end_2_Group);
        Function.send(ring_comm, rank, Uvsp, (s_2 - s_1) * 2, 0, end_1_Group, end_2_Group);
        Function.send(ring_comm, rank, nrvsp_arr, s_1 * 2, 0, end_1_Group, end_2_Group);
        maxnr - nrvsp_arr[0];
        for (int i = 1; i < s_1 * 2; i++)
            if (nrvsp_arr[i] > maxnr)
                maxnr = nrvsp_arr[i];
        r1 += s_1;
        tn = tn + s_1 * tau_1 * mark;
        tau_1 *= mark;
        tau_2 *= mark;
        tau = tau_1;
        if (inerc > inerc_stepUp && nr < (eps_stepUp * Eps) && tau_1 < tau_stepUp)
        {
            inerc = 0;
            fl3 = 1;
            fl4 = 1;
        }
```

```
            }
            if (rank == 0)
            {
               end = omp_get_wtime();
               time = end - start;
            }
      }
      if (rank == 0)
      {
         cout << "Запис у файл...";
         ofstream out;
         string name = "Result(" + to_string(size) + ")_"+ to_string(time) + "_BM).txt";
         out.open(name);
         if (out.is_open())
         {
            out << "Режим: " << "БМ" << endl << " Похибка: " << maxnr << endl << "Час: " << time << <<
endl "Погані кроки: " << badstep << endl << "Всього кроків: " << nstep;
         }
         out.close();
         cout << "Закінчено запис";
      }
      if (rank == 0)
         cout << "Робота закінчена";
      return 0;
   }
```