ДВНЗ «Донецький національний технічний університет»

Факультет комп'ютерно–інформаційних технологій та автоматизації

(повна назва факультету)

Кафедра електронної техніки

(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри електронної техніки
_____ Сергій КОВАЛЬОВ
(підпис) (Ім'я та ПРІЗВИЩЕ)
«___» _____ 2023 р.

Випускна кваліфікаційна робота

магістр
(освітній ступінь)

на тему «Підсистема обміну інформацією проблемно орієнтованого паралельного моделюючого середовища»
Виконав студент 2 курсу, групи КІм–22
(шифр групи)

спеціальності 123 Комп'ютерна інженерія
(код і назва спеціальності)

освітньої програми Комп'ютерна інженерія
(назва освітньої програми)

Корнілов Олександр Олексійович
(Прізвище, Ім'я, По–батькові) (підпис)

Керівник Проф. каф. ЕТ, д.т.н., проф. _____ Володимир СВЯТНИЙ
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Рецензент проф. каф. ПМІ, д.т.н. проф. _____ Євген БАШКОВ
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Рецензент зав. каф. ЕІ, к. т. н.,доц. _____ Олександр КОЛЛАРОВ
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Нормоконтроль
асистент кафедри ЕТ
_____ Артем ЛЮБИМОВ
(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
(повна назва факультету)

Кафедра електронної техніки
(повна назва кафедри)

Захист відбувся _____
(дата)
з
оцінкою_____
Секретар ЕК
_____
(підпис)

Випускна кваліфікаційна робота

магістр
(освітній ступінь)

Тема: «Підсистема обміну інформацією проблемно орієнтованого паралельного моделюючого середовища»

Виконав, студент
гр. КІм – 22 _____ Олександр Корнілов
(підпис, дата, Ім'я та ПРІЗВИЩЕ

Керівник _____ Володимир СВЯТНИЙ
(підпис, дата, Ім'я та ПРІЗВИЩЕ)

Консультанти: _____
(підпис, дата, Ім'я та ПРІЗВИЩЕ)

Луцьк – 2023 р.

Державний вищий навчальний заклад
"Донецький національний технічний університет"

Факультет Комп'ютерно–інформаційних технологій та автоматизації
Кафедра Електронна техніка
Освітній ступінь магістр
Спеціальність 123 Комп'ютерна інженерія
(код та назва спеціальності)
Освітня програма Комп'ютерна інженерія
(назва освітньої програми)

ЗАТВЕРДЖУЮ
Завідувач кафедри електронної техніки
Сергій КОВАЛЬОВ
«____» _____2023 року

З А В Д А Н Н Я

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА
Корнілов Олександр Олексійович
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Підсистема обміну інформацією проблемно орієнтованого паралельного моделюючого середовища
керівник проєкту (роботи) Святний Володимир Андрійович, д.т.н., проф.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом від 22.09.2023 року № 478
2. Строк подання студентом проєкту (роботи) 8 грудня 2023 року
3. Вихідні дані до проєкту (роботи)
Результати переддипломної практики та самостійної роботи
4. Зміст розрахунково–пояснювальної записки (перелік питань, які потрібно розробити) аналіз робіт, релевантних темі, задачі розробок і досліджень, функціональність і структура підсистеми обміну інформацією, розробка кільцевого тестеру динаміки обміну між двома процесорами, алгоритми функціонування підсистеми обміну інформацією.

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1			
2			
3			
4			

7. Дата видачі завдання 27.09.2023 р

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Проблемна орієнтація РПМС	28.09.23–01.10.23	
2	Структура і функції підсистеми	02.10.23–10.10.23	
3	Розробка алгоритмів	11.10.23–31.10.23	
4	Програмна реалізація	01.11.23–17.11.23	
5	Відлагодження та аналіз результатів	18.11.23–25.11.23	
6	Оформлення пояснювальної записки	26.11.23–07.12.23	
7	Нормоконтроль пояснювальної записки	08.12.23–13.12.23	
8	Рецензування роботи	14.12.23–16.12.23	
9	Захист роботи	22.12.23	

Студент _____ **Олександр КОРНІЛОВ**
(підпис) (Ім'я та ПРІЗВИЩЕ)

Керівник роботи _____ **Володимир Святний**
(підпис) (Ім'я та ПРІЗВИЩЕ)

ЛИСТ ЗАУВАЖЕНЬ

Посада, Ім’я та ПРІЗВИЩЕ	Суть зауваження, оцінка та підпис

АНОТАЦІЯ

Корнілов О. О. Підсистема обміну інформацією проблемно орієнтованого паралельного моделюючого середовища / Випускна кваліфікаційна робота на здобуття освітнього ступеня «магістр» за спеціальністю 123 «Комп'ютерна інженерія» – ДВНЗ ДонНТУ, Покровськ/Луцьк, 2023.

Пояснювальна записка: 148 стор., 48 рис., 5 табл., 23 посилання

Об'єктом дослідження є підсистема обміну інформацією проблемно орієнтованого розподіленого паралельного моделюючого середовища (ПОРПМС).

Предметом дослідження виступають алгоритмічні й апаратно–програмні засоби, які забезпечують функціонування підсистеми обміну інформацією середовища.

Наукова новизна: отримав подальший розвиток принцип декомпозиції РПМС шляхом побудови підсистеми обміну інформацією, що враховує проблемну орієнтацію середовища.

Практична цінність: розроблена підсистема РПМС впроваджується в практику роботи ParSimTech–центру ДонНТУ та в навчальний процес за спеціальністю «Комп'ютерна інженерія».

Для розробки підсистеми РПМС було використано мову програмування C, MPI– та OpenMP–стандарти паралельного програмування, mpich реалізацію стандарту MPI.

Ключові слова: проблемно орієнтоване паралельне моделююче середовище, мережевий динамічний об'єкт з розподіленими параметрами, модель, simulation–модель, симулятор, паралельне програмування, обмін даними.

ANNOTATION

Kornilov O. O. Datenaustauschsubsystem der poblemorientierten parallelen Simulationsumgebung / Final qualification work for the degree of "master" in the specialty 123 "Computer Engineering" – SHEI DonNTU, Pokrovsk/Lutsk, 2023.

Objekt der Untersuchungen ist das Datenaustauschsubsystem der problemorientierten verteilten parallelen Simulationsumgebung (POVPSU).

Die Gegenstände der Untersuchungen und Entwicklungen sind die algorithmischen und HW/SW–Mittel, welche die Funktionierung des Subsystems von Datenaustausch verwirklichen.

Wissenschaftliche Modernität: die erhaltene Ergebnisse beitragen zur weiteren Entwicklung der Dekomposition von VPSU durch Aufbau des Datenaustauschsubsystems mit der Berücksichtigung der VPSU–Problemorientierung.

Der Nutzwert: vorgeschlagenes Subsystem der VPSU wird in Praxis des ParSimTech–Zentrums der DonNTU und in der Lehre des Fachgebiets „Computer Engineering“ eingesetzt.

Keywords: problemorientierte parallele simulationsumgebung, dynamisches netzobjekt mit verteilten parametern, modell, simulationsmodell, simulator, parallele programmierung, datenaustausch.

ЗМІСТ

ВСТУП 10
1 АНАЛІЗ РОБІТ, РЕЛЕВАНТНИХ ТЕМІ, ЗАДАЧІ РОЗРОБОК І ДОСЛІДЖЕНЬ 14
1.1 Технологічна характеристика і математична модель мережевого динамічного об’єкту з розподіленими параметрами 14
1.2 Структура проблемно орієнтованого РПМС 19
1.3 Характеристика засобів обміну даними Infiniband 23
1.4 Задачі розробок і досліджень 24
2 ФУНКЦІОНАЛЬНІСТЬ І СТРУКТУРА ПІДСИСТЕМИ ОБМІНУ ІНФОРМАЦІЄЮ 25
2.1 Підходи до розпаралелювання МДОРП і віртуальні рівні паралельності 25
2.2 РП–1 28
2.2.1 БОМ–операції обчислень в Q– і P–процесах для j–гілки і МДОРП 28
2.2.2 Порівняння об’єму операцій обміну даними з об’ємом обчислень, вивід формул для оцінки віртуального прискорення 31
2.2.3 Зв’язки між віртуальними процесами, віртуальна мережа обміну інформацією 38
2.3 РП–2 40
2.3.1 БОМ–операції обчислень в $(QP)_{jv}$–БОМ–процесах для j–гілки і МДОРП 41
2.3.2 Порівняння об’ємів операцій обміну даними і обчислювальних операцій, вивід формул для оцінки віртуального прискорення 44
2.3.3 Схеми зв’язку між віртуальними процесами, віртуальна мережа зв’язку52
2.4 РП–3 53
2.4.1 Структура віртуальних QP–процесів з підпорядкуванням „Система рівнянь j–гілки – процес“. 53
2.4.2 Порівняння об’ємів операцій обміну даними і обчислень, оцінка віртуального прискорення 54

2.4.3 Схема з’єднань між віртуальними процесами та віртуальна мережа зв’язку. 57
2.5 РП–4 59
2.5.1 Порівняння об’ємів операцій обміну даними і обчислень, оцінка віртуального прискорення 59
2.6 Визначення підсистеми та її основні функції 62
3 РОЗРОБКА КІЛЬЦЕВОГО ТЕСТЕРУ ДИНАМІКИ ОБМІНУ МІЖ ДВОМА ПРОЦЕСОРАМИ 64
3.1 Постановка задачі 64
3.2 Simulation–модель і структура тестера, схема обміну даними між процесорами 64
3.3 Алгоритми вирішувачів рівнянь тестера, обчислювальні методи 65
3.4 Програмна реалізація 68
3.5 Експериментальні дослідження 74
3.6 Висновки 75
4 АЛГОРИТМИ ФУНКЦІОНУВАННЯ ПІДСИСТЕМИ ОБМІНУ ІНФОРМАЦІЄЮ 76
4.1 Презентація підсистеми 76
4.1.1 Апаратні ресурси MIMD–кластера ДонНТУ 76
4.1.2 Програмні ресурси MPI та OpenMP 77
4.2 Організація обміну даними в паралельному симуляторі мережевого динамічного об’єкту з зосередженими параметрами (МДОЗП) 80
4.3 Організація обміну даними в паралельних симуляторах мережевого динамічного об’єкту з розподіленими параметрами (МДОРП) 86
4.3.1 Орієнтація на MIMD–системи з розподіленим адресним простором і MPI–засобами паралельного програмування 86
4.3.2 Орієнтація на MIMD–системи з загальною пам’яттю і OpenMP–засобами паралельного програмування 89
4.3.3 Орієнтація на MIMD–системи з комбінованою організацією пам’яті і MPI/OpenMP–засобами паралельного програмування 101

4.3.4 Перспективна архітектура і розвиток підсистеми 103
4.4 Висновки 107
ВИСНОВКИ 108
СПИСОК ЛІТЕРАТУРИ 110
ДОДАТОК А – Заява на затвердження теми і керівника 113
ДОДАТОК Б – Лістинги 114
ДОДАТОК В – Охорона праці ..... **Ошибка! Закладка не определена.**

ВСТУП

Актуальність теми. Розробка проблемно орієнтованих розподілених паралельних моделюючих середовищ (ПОРПМС) належить до проблематики технологій паралельного моделювання складних динамічних систем (Parallel Simulation Technology, Parallele Simulationstechnik – ParSimTech–проблеми). Серед об'єктів моделювання, що потребують застосування паралельних обчислювальних систем, актуальними є мережеві динамічні об'єкти з розподіленими параметрами (МДОРП), що широко розповсюджені в різних предметних областях (галузях промисловості) – в гірництві, гідравліці, аеромеханіці, газодинаміці, електротехніці, хімічних та біохімічних технологіях, машинобудуванні, електроніці, в міждисциплінарних проєктах з захисту довкілля. Вони належать до складних динамічних систем (СДС), що характеризуються великою кількістю одночасно функціонуючих компонент, нелінійністю залежностей між параметрами, різними за фізичним змістом і одночасно працюючими динамічними процесами, ієрархічністю розташування засобів регулювання та керування, залежними від навантажень характеристиками джерел енергії (активні елементи – генератори, вентилятори, насоси, компресори, акумулятори і ін.). Динаміка процесів в МДОРП є предметом активної модельної підтримки інноваційних проєктів як вирішального фактору правильності проєктних рішень на всіх етапах розробок і, як наслідок, конкурентоспроможності об'єктів нової техніки та розвинутих технологій (Model Drive Engineering, Modellgetriebene Entwicklung – MDE/MGE–Problem). При вирішенні цих задач стосовно МДОРП реальної промислової розмірності виникає певний поріг складності математичних моделей, який може бути подоланим шляхом застосування паралельних обчислювальних систем (ПОС), що є основою симуляторів як апаратно–програмних реалізацій Simulation–моделей МДОРП. Наукова школа ДонНТУ з ParSimTech–проблем співпрацює з надпотужним обчислювальним центром

та інститутами Штутгартського університету (HLRS – Höchstleistungsrechenzentrum Stuttgart, IPVS – Institut für parallele und verteilte Systeme, IAS – Institut für Automatisierungs– und Softwaresysteme). Запропоновано проблемну орієнтацію розподіленого паралельного моделюючого середовища (РПМС) на вирішення задач моделювання МДОРП. Центральною функціональною складовою середовища має бути підсистема обміну інформацією між процесорами цільової паралельної обчислювальної системи ПОРПМС. Розробці цієї підсистеми з врахуванням актуальних архітектурних і програмних особливостей сучасних MIMD–систем присвячено дану кваліфікаційну роботу. Її виконано в рамках проєкту з побудови Українського науково–навчального ParSimTech–центру за сприяння стипендіальної DAAD–програми ім. Леонарда Ейлера.

Мета роботи – розробити підсистему обміну інформацією, що зумовить орієнтацію паралельного моделюючого середовища на вирішення задач активної модельної підтримки МДОРП–проєктів і сприятиме підвищенню рівня дружності РПМС до користувачів.

Для досягнення цієї мети в роботі вирішуються наступні задачі розробок і досліджень:

1) Дано технологічну характеристику, розглянуто математичну модель і Simulation–модель мережевого динамічного об’єкту з розподіленими параметрами (МДОРП), на який орієнтовано РПМС.
2) Визначено функціональність підсистеми обміну інформацією РПМС.
3) Розроблено структуру підсистеми.
4) Запропоновано й реалізовано тестер динаміки обміну даними між двома процесорами цільової паралельної обчислювальної системи .
5) Виконано експерименти і запропоновано використати результати розробок в ParSimTech–центрі ДонНТУ.

Об’єктом дослідження є підсистема обміну інформацією проблемно орієнтованого розподіленого паралельного моделюючого середовища (ПОРПМС).

Предметом дослідження виступають алгоритмічні й апаратно–програмні засоби, які забезпечують функціонування підсистеми обміну інформацією середовища.

Наукова новизна: отримав подальший розвиток принцип декомпозиції РПМС шляхом побудови підсистеми обміну інформацією, що враховує проблемну орієнтацію середовища.

Практична цінність: розроблена підсистема ПОРПМС впроваджується в практику роботи ParSimTech–центру ДонНТУ та в навчальний процес за спеціальністю «Комп’ютерна інженерія».

1 АНАЛІЗ РОБІТ, РЕЛЕВАНТНИХ ТЕМІ, ЗАДАЧІ РОЗРОБОК І ДОСЛІДЖЕНЬ

1.1 Технологічна характеристика і математична модель мережевого динамічного об’єкту з розподіленими параметрами

В якості прикладу мережевого динамічного об’єкту з розподіленими параметрами було обрано модель системи вентиляції шахти.

У системах вентиляції шахт важливо розрізняти аеродинамічні процеси, пов’язані з повітрям і газом [1]. Для вивчення та розв’язання завдань з питань безпеки розглядаються наступні аеродинамічні процеси:

– Аеродинамічні процеси фільтрації повітря в пористих середовищах (вироблені простори, шляхи втрат повітря);

– Аеродинамічні процеси подачі повітря на робочі місця підготовчих виробок, де встановлені місцеві вентилятори;

– Динаміка розподілу повітряних потоків в шахтній вентиляційній мережі (ШВМ) без регуляторів повітряного потоку;

– Динамічна поведінка вентиляторів;

– Регульована динаміка повітряних потоків в автоматизованій системі вентиляції шахт;

– Динаміка автоматизованих систем шлюзів в ШВМ;

– Динаміка переключення системи вентиляції шахт в реверсивний режим роботи та критичних щодо безпеки ситуацій (відключення вентиляторів, коротке замикання, місцеві обвали вугільних пластів тощо).

Події, пов’язані з питаннями безпеки та станом системи вентиляції шахт, залежать від наступних газодинамічних процесів:

– Динаміка газових потоків з джерел метану; ми будемо розглядати лише метан і передбачати, що всі джерела метану працюють асинхронно та незалежно одне від одного;

– Динаміка потоку метану в фільтраційних середовищах "Старі робочі місця" (СРМ, використані тривимірні простори, де раніше були вугільні пласти);

– Динаміка процесів змішування на межевих ділянках "СРМ–шахта";

– Динаміка концентрацій метану в критично важливих пунктах моніторингу;

– Динаміка концентрацій метану в лавах, залежно від робочого плану та фаз роботи;

– Оптимальне управління рівнем концентрації метану шляхом зміни величини повітряного потоку;

– Динаміка концентрації метану під час переключення (реверсу) напрямку повітряних потоків як часова та просторова проблема;

– Динаміка концентрацій метану в тупикових робочих просторах без протікання повітряних потоків (місцева подача повітря за допомогою спеціальних місцевих вентиляторів).

Аерогазодинамічні процеси в лавах, штреках, вироблених просторах схем провітрювання виробничих дільниць та стволах ШВМ мають описуватися диференціальними рівняннями в частинних похідних як об’єктах з розподіленими параметрами. Існують два види фрагментів ШВМ, які беруться до уваги при математичному описі аерогазодинамічних процесів [1].

1) Складні елементи з двома лавами, двома виробленими просторами та трьома штреками, що включають всі можливі варіанти взаємозв’язків між повітряними потоками (рис. 1.1).

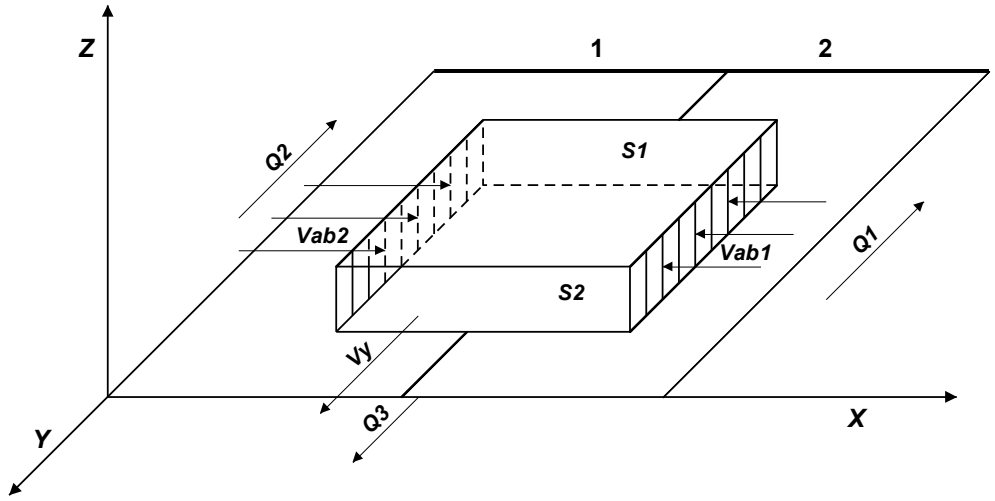


Рисунок 1.1 – Схема складного фрагменту ШВМ (виробнича дільниця зі спареними лавами)

Математичний опис процесів на рис.1.1 представлений системою диференціальних рівнянь у частинних похідних [1]:

$$-\frac{\partial P_i}{\partial \varepsilon} = \frac{2\rho}{F_i^2} Q_i \frac{\partial Q_i}{\partial \varepsilon} + \frac{\rho}{F_i} \frac{\partial Q_i}{\partial t} + r_i Q_i^2 + r'_i Q_i^2 \quad (1.1)$$

$$-\frac{\partial P_i}{\partial t} = \frac{\rho a^2}{F_i} \frac{\partial Q_i}{\partial \varepsilon} + \frac{\rho a^2}{S_i} q_i \quad (1.2)$$

$$-\frac{\partial P_3}{\partial y} = -\frac{2\rho}{F_3^2} Q_3 \frac{\partial Q_3}{\partial \varepsilon} + \frac{\rho}{F_3} \frac{\partial Q_3}{\partial t} + r_3 Q_3^2 + r'_3 Q_3^2 \quad (1.3)$$

$$-\frac{\partial P_3}{\partial t} = \frac{\rho a^2}{F_3} \frac{\partial Q_3}{\partial y} + \frac{\rho a^2}{F_3} (q_1 + q_2) \quad (1.4)$$

$$-\nabla P_{1i} = \frac{\rho}{m_i} \frac{\partial v_i}{\partial t} + p(\frac{\rho v}{k_i} + \frac{v_i}{l_i}) v_i \quad (1.5)$$

$$-\frac{\partial P_{1i}}{\partial t} = \frac{\rho a^2}{m_i} \nabla v_i \quad (1.6)$$

де P_i, P_{1i} – відповідно: повітряний тиск у штреках та в порожнинах вироблених просторів, Н/м²;

Q_i – повітряний потік у штреках, м³/с;

t – час, сек;

$0 < \xi < l$ – координата, виміряна від початку гірничої виробки до її кінця вздовж лінії фільтрації, м;

ρ – густина повітря, кг/м³;

F_i – площа поперечного розрізу штреку, м².

Граничні умови:

$$P_i \mid_{\xi = 0} = f_i (Q_i) \mid_{\xi = 0} \quad (1.7)$$

$$P_{1i} \mid_{L} = P_3 (\xi) \quad (1.8)$$

$$q_i = k_i \frac{\partial P_3}{\partial n} | L \quad (1.9)$$

2) Прості повітропроводи з глухими стінами без втрат фільтрації. Це окремі гірничі виробки, що є гілками ШВМ. Ці елементи можна описати наступною системою рівнянь [1]:

$$\begin{cases} -\frac{\partial P_i}{\partial \varepsilon} = \frac{\rho}{F_i} \frac{\partial Q_i}{\partial t} + r_i Q_i^2 + r_i' Q_i^2 \\ -\frac{\partial P_i}{\partial t} = \frac{\rho a^2}{F_i} \frac{\partial Q_i}{\partial \varepsilon}, \end{cases} \quad (1.10)$$

Граничні умови – це функції тиску в початковому і кінцевому вузлах елементу мережі.

Аналіз експериментальних вимірювань повітряних потоків в різних точках гірничих робіт та результати моделювання фільтраційних процесів показують, що просторово розподілений фільтраційний потік в шахті може бути апроксимований з достатньою точністю паралельними, концентрованими потоками (рис. 1.2).

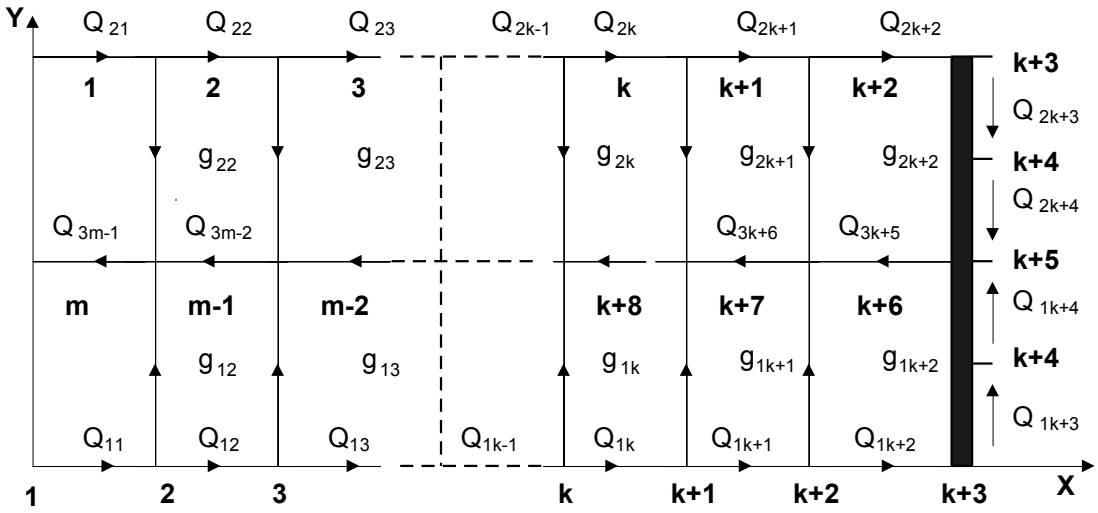


Рисунок 1.2 – Апроксимація розподіленого фільтраційного потоку

Ці потоки ділять відстань на елементи відповідно до різницевого методу розв'язування рівнянь у частинних похідних. Запишемо рівняння (1.1) … (1.6) у різницевій формі для k–го елемента штрека з врахуванням напрямку потоку повітря:

$$-\frac{P_{ik}-P_{ik+1}}{\Delta\varepsilon}=\frac{2\rho}{F_i^2}Q_{ik}\frac{Q_{ik}-Q_{ik+1}}{\Delta\varepsilon}+\frac{\rho}{F_i}\frac{\partial Q_{ik}}{\partial t}+r_iQ_{ik}^2+r'_iQ_{ik}^2 \qquad (1.11)$$

$$-\frac{P_{ik+1}}{dt}=\frac{\rho a^2}{F_i}\frac{Q_{ik}-Q_{ik+1}}{\Delta\varepsilon}+\frac{\rho a^2}{S_i}\frac{g_{ik}}{\partial\varepsilon},i=1,2 \qquad (1.12)$$

$$\frac{P_{3l}-P_{3l+1}}{\Delta x}=\frac{2\rho}{F_3^2}Q_{3l}\frac{Q_{3l}-Q_{3l+1}}{\Delta x}+\frac{\rho}{F_3}\frac{\partial Q_{3l}}{\partial\varepsilon}+r_3Q_{3l}^2+r'_3Q_{3l}^2 \qquad (1.13)$$

$$-\frac{P_{3l+1}}{dt}=\frac{\rho a^2}{F_3}\frac{Q_{3l}-Q_{3l+1}}{\Delta x}+\frac{\rho a^2}{S_3}\frac{g_{1k}+g_{2k}}{\Delta x},l=k+8 \qquad (1.14)$$

$$-\frac{P_{1ik}^{(S)}-P_{1ik}^{(S+!)}}{\Delta y}=\frac{\rho}{m_iF_k}\frac{dg_{ik}^{(S)}}{dt}+\frac{\rho v}{k_iF_k}g_{ik}^{(S)}+\frac{\rho}{l_iF_k^2}\left|g_{ik}^{(S)}\right|g_{ik}^{(S)} \qquad (1.15)$$

$$\frac{dP_{1ik}^{(S+!)}}{dt}=\frac{\rho a^2}{m_iF_k}\frac{g_{ik}^{(S)}-g_{ik}^{(S+1)}}{\Delta y} \qquad (1.16)$$

де s = 1, 2, … , N – номер елемента повітряного потоку g_{ik};

$\mathbf{P_{1ik}^{(1)}=P_{ik+1}}$, $\mathbf{P_{1k}^{(N+1)}=P_{k+8}}$ – граничні умови для потоку повітря g_{ik};

k = 1, 2, …, м; g_{ik} – величини концентрованих потоків повітря, м3/сек.

Опір r’(t) враховується в рівнянні елемента, в якому розташований регулюючий орган (шибер, вентиляційні двері та ін.). Рівняння (1.11), (1.12) для точки g_{ik}=0 наступне (1.17).

$$\frac{2\rho}{F_i^2} Q_{ik} \frac{Q_{ik}-Q_{ik+1}}{\Delta\varepsilon} = 0 \qquad (1.17)$$

Отже, маємо систему звичайних диференціальних рівнянь (1.11) ... (1.16), які апроксимують систему (1.1) – (1.6). Граничні та початкові умови залишилися незмінними. Для апроксимації рівнянь (1.10) використовуються рівняння (1.11), (1.12) з урахуванням (1.17) та g_{ik}=0.

Моделювання процесів в окремих гілках ШВМ та в простих мережах показує, що число елементів апроксимації N повинно бути в діапазоні від 6 до 10 з точки зору точності. Для реальних мереж кількість гілок m > 100 .

1.2 Структура проблемно орієнтованого РПМС

Проблемно–орієнтоване моделююче середовище з розподіленими параметрами (ПОРПМС) має структуру, яка включає різні компоненти для ефективного вирішення завдань моделювання з розподіленими параметрами. Основні складові цієї структури включають програмне забезпечення моделювання та оптимізації, системне програмне забезпечення та обладнання (рис. 1.3).

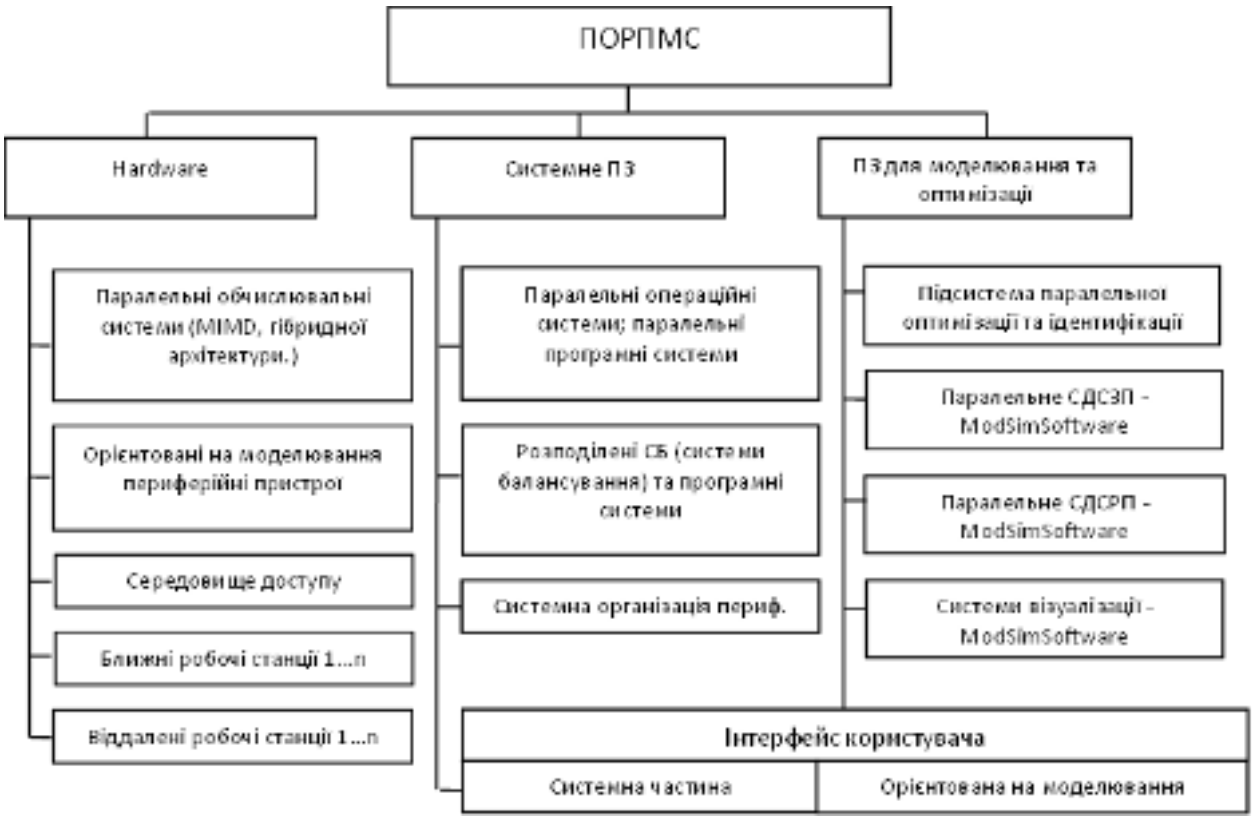


Рисунок 1.3 – Структура ПОРПМС

Для підсистеми обміну даними важливими є апаратні та програмні складові, які забезпечують передачу і обмін інформацією між різними компонентами MIMD–системи.

Основними програмними частинами системи обміну даними є наступні:

Протоколи обміну даними. Протоколи визначають правила та структуру обміну даними між компонентами системи. Вони визначають, як дані повинні бути упаковані, передані, розгорнуті та підтверджені. В контексті підсистеми обміну даними важливими є протоколи, які знаходяться на «нижчих» рівнях в ієрархії моделі OSI. До них відносяться такі протоколи як TCP, UDP, QUIC [2].

Бібліотеки програмування. До таких бібліотек відносять MPI та OpenMP. MPI є стандартом та бібліотекою для паралельного програмування, призначеною для розподілених обчислень на кластерах або суперкомп’ютерах. Вона дозволяє створювати програми, які виконуються на багатьох вузлах, обмінюючи даними через повідомлення між вузлами. OpenMP є стандартом та набором розширень для багатьох мов програмування, призначених для

паралельного програмування на обчислювальних вузлах з багатьма ядрами. Вона спрощує розпаралелювання коду та управління потоками.

До апаратної частини підсистеми обміну даними в комп'ютерних системах відносяться фізичні компоненти та пристрої, які забезпечують функціональність обміну даними між різними компонентами системи. Ось деякі з них:

Мережеві адаптери: Мережеві адаптери або мережеві карти – це пристрої, які дозволяють комп'ютерам та іншим пристроям приєднуватися до локальної мережі чи Інтернету. Вони можуть бути вбудованими на материнській платі або підключатися як окремі розширювальні карти.

Маршрутизатори: Маршрутизатори є ключовими пристроями для мережевого обміну даними. Вони визначають оптимальний маршрут для пакетів даних, які пересилаються між різними мережевими сегментами. Маршрутизатори можуть також виконувати функції фаєрволу для забезпечення безпеки мережі [3].

Комутатори: Комутатори використовуються для забезпечення комутації даних на локальному рівні мережі. Вони визначають, куди направити пакети даних в локальній мережі, щоб забезпечити ефективний обмін між пристроями [4].

Мережеві кабелі та з'єднання: Фізичні кабелі, такі як Ethernet–кабелі, оптоволоконні кабелі, бездротові адаптери тощо, є важливими компонентами для передачі даних в мережі. Правильний вибір та налаштування кабелів і з'єднань впливає на швидкість та надійність обміну даними.

На рисунку 1.4 знаходиться графік, який показує залежність між кількістю процесорів та часом виконання обчислень.

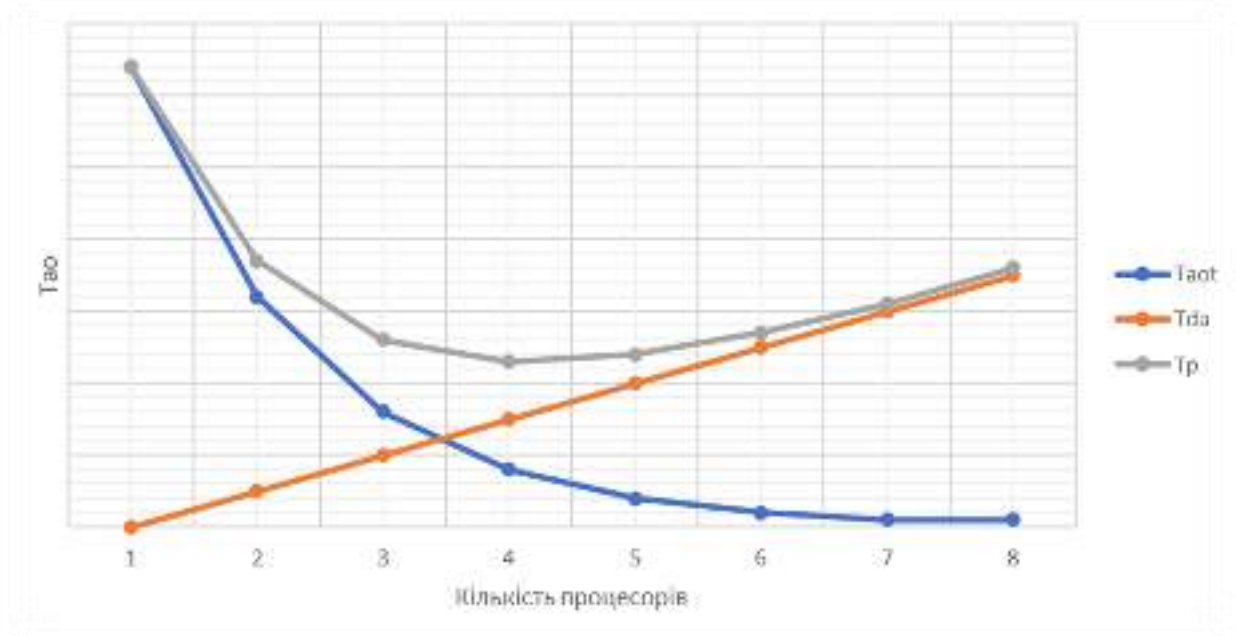


Рисунок 1.4 – Тривалість обчислень

де T_{AO} – тривалість виконання арифметичних операцій;

T_p – реальний час обчислення;

T_{AOT} – теоретична тривалість паралельних обчислень *N* процесорами;

T_{DA} – час для обміну інформацією між процесорами.

На даному графіку видно, що теоретична тривалість паралельних обчислень N процесорами експоненціально зменшується. Якщо прийняти тривалість обміну даними $T_{DA}(N)$ лінійною в залежності від кількості процесорів, то графік $T_{AOT}(N)$ зміщується в напрямку збільшення тривалості обчислень і відповідного зменшення можливого їх прискорення. З аналізу цих графіків випливають наступні формули:

Теоретичне прискорення

$$S_T = \frac{T(1)}{T(N)} \tag{1.18}$$

Реально досяжне прискорення

$$S_p = \frac{T(1)}{T_{AO}(N)+T_{DA}(N)} = \frac{T(1)}{T_{AO}(N)} * \frac{1}{1+\frac{T_{DA}(N)}{T_{AO}(N)}} \quad (1.19)$$

Для отримання по можливості позитивного й прийнятного прискорення треба забезпечити мінімізацію співвідношення

$$\frac{T_{DA}(N)}{T_{AO}(N)} \to min \quad (1.20)$$

В даній роботі будемо орієнтуватись на апаратно–програмні засоби обміну повідомленнями Infiniband, що використовується в сучасних MIMD–системах.

1.3 Характеристика засобів обміну даними Infiniband

InfiniBand (IB) — це стандарт зв'язку комп'ютерної мережі, який використовується у високопродуктивних обчисленнях і має дуже високу пропускну здатність і дуже низьку затримку [5].

InfiniBand використовує топологію комутованої мережі. Усі передачі починаються або закінчуються на адаптері каналу. Кожен процесор містить адаптер каналу хоста (HCA), а кожен периферійний пристрій має адаптер цільового каналу (TCA). Ці адаптери також можуть обмінюватися інформацією для безпеки або якості обслуговування (QoS).

InfiniBand на канальному рівні передає дані в пакетах розміром до 4 КБ, які разом утворюють повідомлення [6]. Повідомлення може бути:

- віддалений прямий доступ до пам'яті, читання або запис;
- канал надсилання або отримання;
- операція на основі транзакції (яка може бути скасована);
- багатоадресна передача;
- атомна операція.

1.4 Задачі розробок і досліджень

Мета роботи – розробити підсистему обміну інформацією, що зумовить орієнтацію паралельного моделюючого середовища на вирішення задач активної модельної підтримки МДОРП–проектів і сприятиме підвищенню рівня дружності РПМС до користувачів.

Для досягнення цієї мети в роботі мають бути вирішені наступні задачі розробок і досліджень:

1. Дати технологічну характеристику, розглянути математичну модель і Simulation–модель мережевого динамічного об’єкту з розподіленими параметрами (МДОРП), на який орієнтовано РПМС.
2. Визначити функціональність підсистеми обміну інформацією РПМС.
3. Розробити структуру підсистеми.
4. Запропонувати й реалізувати тестер динаміки обміну даними між двома процесорами цільової паралельної обчислювальної системи.
5. Виконати експерименти і запропонувати використання результатів розробок в ParSimTech–центрі ДонНТУ.

2 ФУНКЦІОНАЛЬНІСТЬ І СТРУКТУРА ПІДСИСТЕМИ ОБМІНУ ІНФОРМАЦІЄЮ

2.1 Підходи до розпаралелювання МДОРП і віртуальні рівні паралельності

Визначення мінімального кроку процесів у контексті декомпозиції та просторової дискретизації МДОРП призводить до віртуальної паралельної моделі симуляції МДОРП, яка включає наступний набір аеродинамічних рівнянь Q та P (процесів):

$$NQ_{Gl} = \sum M_j\,, j = 1{,}2, \dots, m;$$

$$NP_{Gl} = n + \sum \left(M_j - 2\right),$$

де m – множина ребер;

n – множина вузлів;

Mj – множина дискретних елементів j–го ребра графа МДОРП.

Значення Mj обчислюється відповідно до довжини ребра lj та обраного кроку Δx:

$$M_j = {}^{l_j}/_{\Delta x}$$

При цьому вибирається однаковий крок для всіх ребер МДОРП. Для створення віртуальних паралельних моделей симуляції за допомогою взаємодії з користувачем передбачається виконання відповідних обчислень параметрів. Пропонуються такі підходи до паралелізації моделей симуляції МДОРП:

1–й підхід: після просторової дискретизації отримали вказані елементарні повітряні шляхи ∑Mj з n + ∑(Mj – 2) граничними умовами вузлів. Відповідно призначаємо Q– та P–процесам (1 рівняння = 1 процес) відповідні Q– та P–рівняння. Цей підхід призводить до максимально можливої кількості мінімального кроку процесів, які відповідають за обчислення компонент векторів Q та P. Також віртуальна мережа зв'язку (комунікаційний граф) повинна бути побудована орієнтовно на ребра, щоб з'єднати вказані процеси сусідніми чином відповідно до просторової дискретизації. Ця віртуальна паралельна модель симуляції відноситься до першого рівня паралельності (РП–1). Згідно з обраним числовим методом отримуємо дискретну віртуальну паралельну модель симуляції (ДВПМС), напр. Eu–, RuKu–, AB– або БОМ–ДВПSM рівня паралельності РП–1.

2–й підхід: призначимо Q– та P–рівнянням відповідні QP–процеси (2 рівняння елемента = 1 процесу). Тоді віртуальна паралельна модель симуляції матиме зменшену кількість ∑Mj – nk QP–процесів, які відповідають за розв'язання систем рівнянь елементів ребра, включаючи граничні значення вузлів. Тут nk – це кількість процесів, які формально виникають для обчислення значень тисків на вузлах, які є інцидентними більш, ніж одному ребру потоку повітря. Згідно з обраним числовим методом отримуємо дискретну віртуальну паралельну модель симуляції (ДВПМС), напр. Eu–, RuKu–, AB– або БОМ–ДВПSM рівня паралельності РП–2.

3–й підхід: додаткове збільшення робочого обсягу процесів досягається призначенням "1 ребро –> 1 процес". Такий підхід надає нам точно m процесів, відповідальних за обчислення компонентів "мультивекторів" Q, P:

$$Q_j\left(Q_{j1}, Q_{j2}, \dots, Q_{jMj}\right), j = 1,2, \dots, m$$

$$P_j(P_{jAKJ} = P_{j1}, P_{j2}, \dots, P_{jMj}, P_{jMj+1} = P_{jEKJ})$$

Тут P_{jAKJ} та P_{jEKJ} відповідають початковим та кінцевим вузлам ребра Q_j. З кожного j–процесу вирішуються 2Mj задачі Коші, включаючи обчислення граничних значень вузлів P_{jAKJ}, P_{jEKJ} та зовнішніх граничних умов (підключення до атмосфери та до вентиляторів). З міркувань однорідності навантаження також розглядатимемо варіант РП–3 "1 ребро l_{min} = 1 процес". Згідно з обраним числовим методом отримуємо дискретну віртуальну паралельну модель симуляції (DVPSM), напр. Eu–, RuKu–, AB– або БОМ–ДВПSM цього рівня паралельності РП–3 в двох згаданих варіантах [7].

4–й підхід: наступна структурна одиниця в графі МДОРП – це часткова мережа. Розкладання графа на певну кількість часткових графів (ЧГ) формулюється як задача оптимізації мінімізації обсягу комунікації між частковими мережами. МДОРП–граф можна розкласти на множину ЧГ з урахуванням наступних міркувань:

• застосування формальних методів розкладання;

• використання об’єктно–специфічних властивостей об’єктів мережі, наприклад, мережі вентиляції шахт дозволяють нам: розділення мережі на v частин відповідно до кількості вентиляторів; локалізація в отриманих v частинах схем розподілу вентиляції; локалізація в схемах розділу середовищ фільтрації – з позначкою для старих робочих місць; визначення приєднаних вузлів як граничних умов та з’єднувальних ребер як компонентів мережі;

• комбінація вищезазначених методів для оптимізації розкладання МДОРП за допомогою діалогової підтримки.

Візьмемо відповідності "1 ЧГ –> 1 Процес" та отримаємо, відповідно до обраного числового методу, дискретну віртуальну паралельну модель симуляції (ДВПSM), напр. Eu–, RuKu–, AB– або БОМ–ДВПSM цього рівня паралельності РП–4.

2.2 РП–1

2.2.1 БОМ–операції обчислень в Q– і P–процесах для j–гілки і МДОРП

Структура віртуальної паралельної моделі симуляції (ВПСМ) для розподіленої за просторовою координатою кромки DNOVP з M елементами представлена на рис. 2.1.

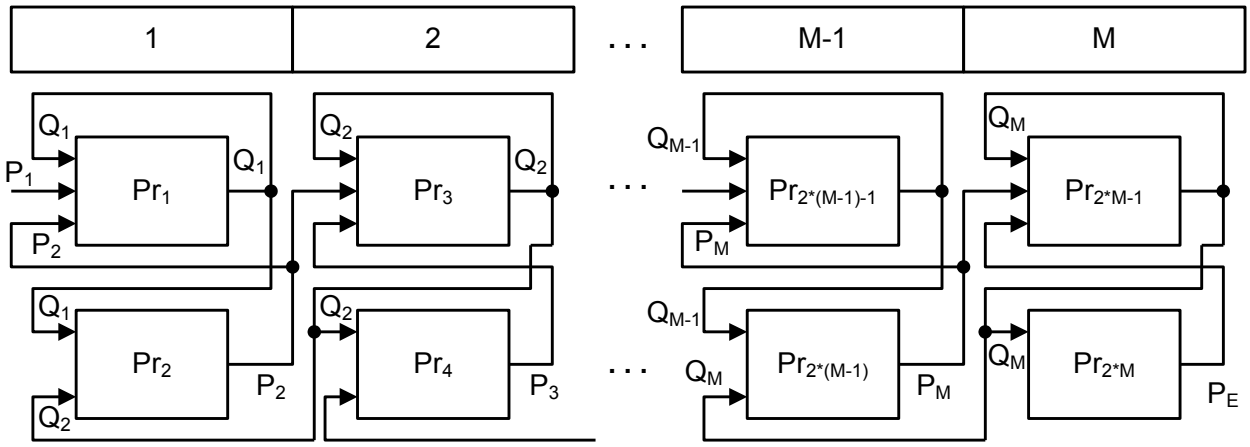


Рисунок 2.1 – ВПSM–структура дискретизованої гілки МДОРП

В рівняннях гілки

$$\begin{cases} \dot{Q}_{jv} = a_j\left(P_{jv} - P_{j,v+1}\right) - \beta_j Q_{jv}\left|Q_{jv}\right| \\ \dot{P}_{j,v+1} = g_j\left(Q_{jv} - Q_{j,v+1}\right) \end{cases} \tag{2.1}$$

де, M_j – множина елементів ребра;

j=1, 2,... m (m – множина ребер в DNOVP);

v = 1, 2, ... M_j – порядковий номер елемента ребра;

Q_{j1}, Q_{j2}, ..., Q_{jv-1}, Q_{jv}, Q_{jv+1}, ..., Q_{jMj} – порядкова нумерація повітряних потоків в елементах ребра;

P_{j1}, P_{j2}, ..., P_{jv-1}, P_{jv}, P_{jv+1}, ..., P_{jMj}, P_{jMj+1}– порядкова нумерація тисків в елементах ребра; при цьому P_{j1} – тиск на початку, а P_{jMj+1} – тиск в кінці ребра;

При цьому різниці тисків $P_{j1}-P_{j2}$, $P_{j2}-P_{j3}$, ..., $P_{jv-1}-P_{jv}$, $P_{jv}-P_{jv+1}$, $P_{jv+1}-P_{jv+2}$,,..., $P_{jMj}-P_{jMj+1}$, відповідають повітряним потокам Q_{j1}, Q_{j2}, ..., Q_{jv-1}, Q_{jv}, Q_{jv+1}, ..., Q_{jMj}.

Кожен елемент ребра j в DNOVP має P_{j1} як початковий вузол і P_{jMj+1}– як кінцевий вузол. З рисунку 2.1 і рівняння (2.1) випливає, що Q–процеси для елементів ребра j (v=1,2,..., M_{j-1}, M_j,) повинні виконувати наступні операції:

1) Введення значень тисків P_v, та P_{v+1}, від сусідніх процесів;

2) Обчислення значень Q_v,;

3) Виведення значень Q_v, сусідньому процесу.

З боку P–процесів виконуються аналогічні операції:

1) Введення значень повітряних потоків Q_v та $Q_{v+!}$ від сусідніх процесів;

2) Обчислення значень P_{v+1};

3) Виведення значень P_{v+1} сусідньому процесу.

Програмний зміст процесів повністю залежить від обраного числового методу.

Q–процес для елемента v ребра (рис. 2.1) має вирішити за БОМ перше рівняння з систем (2.1), тобто

$$\dot{Q}_{jv} = a_j\left(P_{jv} - P_{j,v+1}\right) - \beta_j Q_{jv}\left|Q_{jv}\right|$$

БОМ (m=1, k=4) надає наступні обчислювальні формули [1, 8, 9]:

$$U_{n,1} \Longrightarrow Q_{jvn,1} = \frac{\tau}{720}\left(251F_{n,0} + 646F_{n,1} - 264F_{n,2} + 106F_{n,3} - 19F_{n,4}\right)_{Qjv} + Q_{jvn,0};$$

$$U_{n,2} \Longrightarrow Q_{jvn,2} = \frac{\tau}{90}\left(29F_{n,0} + 124F_{n,1} + 24F_{n,2} + 4F_{n,3} - F_{n,4}\right)_{Qjv} + Q_{jvn,0};$$

$$U_{n,3} \Longrightarrow Q_{jvn,3} = \frac{3\tau}{80}\left(9F_{n,0} + 34F_{n,1} + 24F_{n,2} + 14F_{n,3} - F_{n,4}\right)_{Qjv} + Q_{jvn,0};$$

$$U_{n,4} \Longrightarrow Q_{jvn,4} = \frac{2\tau}{45}\left(7F_{n,0} + 32F_{n,1} + 12F_{n,2} + 32F_{n,3} + 7F_{n,4}\right)_{Qjv} + Q_{jvn,0};$$

$$\begin{aligned}
(F_{v,n,0})Q_{jv} &= \alpha_j(P_{jv,0} - P_{jv+1,n,0}) - \beta_j Q_{jv,n,0}|Q_{jv,n,0}| \\
\left(F_{v,n,1}\right)Q_{jv} &= \alpha_j\left(P_{jv,0} - P_{jv+1,n,1}\right) - \beta_j Q_{jv,n,1}\left|Q_{jv,n,1}\right| \\
\left(F_{v,n,2}\right)Q_{jv} &= \alpha_j\left(P_{jv,0} - P_{jv+1,n,2}\right) - \beta_j Q_{jv,n,2}\left|Q_{jv,n,2}\right| \\
\left(F_{v,n,3}\right)Q_{jv} &= \alpha_j\left(P_{jv,0} - P_{jv+1,n,3}\right) - \beta_j Q_{jv,n,3}\left|Q_{jv,n,3}\right| \\
(F_{v,n,4})Q_{jv} &= \alpha_j(P_{jv,0} - P_{jv+1,n,4}) - \beta_j Q_{jv,n,4}|Q_{jv,n,4}|
\end{aligned} \tag{2.2}$$

Р–процес елементу v системи (2.1) повинен, згідно з БОМ, відповідати другому рівнянню з системи (2.1), тобто

$$\dot{P}_{j,v+1} = g_j\left(Q_{jv} - Q_{j,v+1}\right)$$

БОМ (m=1, k=4) надає наступні формули для обчислення [8, 9]:

$$U_{n,1} \Longrightarrow P_{jvn,1} = \frac{\tau}{720}\left(251F_{n,0} + 646F_{n,1} - 264F_{n,2} + 106F_{n,3} - 19F_{n,4}\right)_{Pjv+1} + P_{jv+1,n,0};$$

$$U_{n,2} \Longrightarrow P_{jvn,2} = \frac{\tau}{90}\left(29F_{n,0} + 124F_{n,1} + 24F_{n,2} + 4F_{n,3} - F_{n,4}\right)_{Pjv+1} + P_{jv+1,n,0};$$

$$U_{n,3} \Longrightarrow P_{jvn,3} = \frac{3\tau}{80}\left(9F_{n,0} + 34F_{n,1} + 24F_{n,2} + 14F_{n,3} - F_{n,4}\right)_{Pjv+1} + P_{jv+1,n,0};$$

$$U_{n,4} \Longrightarrow P_{jvn,4} = \frac{2\tau}{45}\left(7F_{n,0} + 32F_{n,1} + 12F_{n,2} + 32F_{n,3} + 7F_{n,4}\right)_{Pjv+1} + P_{jv+1,n,0};$$

$$\begin{aligned}
(F_{v,n,0})_{Pjv+1} &= g_j(Q_{jv,n,0} - Q_{jv+1,n,0}) \\
(F_{v,n,1})_{Pjv+1} &= g_j(Q_{jv,n,1} - Q_{jv+1,n,1}) \\
(F_{v,n,2})_{Pjv+1} &= g_j(Q_{jv,n,2} - Q_{jv+1,n,2}) \\
(F_{v,n,3})_{Pjv+1} &= g_j(Q_{jv,n,3} - Q_{jv+1,n,3}) \\
(F_{v,n,4})_{Pjv+1} &= g_j(Q_{jv,n,4} - Q_{jv+1,n,4})
\end{aligned} \tag{2.3}$$

Значення $Q_{vn,0}$ та $P_{v+1,n,0}$ в нульовій точці блока n визначаються як значення в останній точці попереднього блока для n >1 ($t_{n-1,k} = t_{n,0}$).

Якщо розглядається j–край у мережі DNOVP, тоді змінна $P_{jMj+1,n,k}$ повинна бути визначена як значення тиску в інцидентній вузловій точці після обчислень.

$$\frac{dP_{wn1}}{dt} = g_{wn1}(Q_{jMj} - \sum_{inz} Q_{inz}) \tag{2.4}$$

Тут Q_{inz} позначає потоки, що інцидентують вузлові, wn1 – це порядковий номер кінцевого вузла (умови типу n1). Для числового розв'язання рівнянь типу (2.4), формули БОМ з wn1 = 1, 2, ..., n1 виглядають так [1, 8, 9]:

$$U_{n,1} \Longrightarrow P_{jwn1,n,1} = \frac{\tau}{720}\left(251F_{n,0} + 646F_{n,1} - 264F_{n,2} + 106F_{n,3} - 19F_{n,4}\right)_{Pjwn1} + P_{jwn1,n,0};$$

$$U_{n,2} \Longrightarrow P_{jwn1,n,2} = \frac{\tau}{90}\left(29F_{n,0} + 124F_{n,1} + 24F_{n,2} + 4F_{n,3} - F_{n,4}\right)_{Pjwn1} + P_{jwn1,n,0};$$

$$U_{n,3} \Longrightarrow P_{jwn1,n,3} = \frac{3\tau}{80}\left(9F_{n,0} + 34F_{n,1} + 24F_{n,2} + 14F_{n,3} - F_{n,4}\right)_{Pjwn1} + P_{jwn1,n,0};$$

$$U_{n,4} \Longrightarrow P_{jwn1,n,4} = \frac{2\tau}{45}\left(7F_{n,0} + 32F_{n,1} + 12F_{n,2} + 32F_{n,3} + 7F_{n,4}\right)_{Pjwn1} + P_{jwn1,n,0};$$

$$(F_{n,0})_{Pjwn1} = g_{wn1}(Q_{jMj,n,0} - \sum Q_{inz,n,0}) \tag{2.5}$$

$$(F_{n,1})_{Pjwn1} = g_{wn1}(Q_{jMj,\,n,1} - \sum Q_{inz,n,1})$$

$$(F_{n,2})_{Pjwn1} = g_{wn1}(Q_{jMj,\,n,2} - \sum Q_{inz,n,2})$$

$$(F_{n,3})_{Pjwn1} = g_{wn1}(Q_{jMj,\,n,3} - \sum Q_{inz,n,3})$$

$$(F_{n,4})_{Pjwn1} = g_{wn1}(Q_{jMj,\,n,4} - \sum Q_{inz,n,4})$$

2.2.2 Порівняння об'єму операцій обміну даними з об'ємом обчислень, вивід формул для оцінки віртуального прискорення

Відповідно до рисунка 2.1, кожен Q–процес має три вхідні змінні та одну вихідну, тоді як P–процес має дві вхідні та одну вихідну змінну. Значення Q_v, обчислене в елементі v (рисунок 2.1), є змінною зворотного зв'язку, тому нам

не потрібно організовувати передачу параметра Q_v від виходу до входу в Q–процесі. З урахуванням нумерації Qj_v та Pj_v –змінних, показаної на рисунку 2.1 та в системі рівнянь (2.1), v–процеси обчислюють змінні Q_v та P_{v+1}, тобто ми нумеруємо елементи j–ребра DNOVP як v = 1, 2, 3, ..., Mj–1, Mj, Mj+1, i, отже, тиски Pj1 та PjMj+1 в мережі належать відповідним початковим та кінцевим вузлам j–ребра і обчислюються як рішення вузлових рівнянь типу (2.4) в процесах (2.5).

Отже, Q_v –процес повинен виконувати наступні операції обміну даними (DA):

- введення P_v з процесу P_{v+1};
- введення P_{v+1} з процесу P_v;
- надсилання Q_v до процесу P_v;
- надсилання Q_v до процесу P_{v-1}.

Спрямований на обчислення $P_{v+!}$, процес P_v повинен виконувати наступні операції DA:

- надсилання P_{v+1} до процесу Q_v;
- надсилання P_{v+1} до процесу Q_{v+1};
- введення Q_v з процесу Q_v;
- введення Q_{v+1} з процесу Q_{v+1}.

Отже, на кожному кроці обчислення кожен віртуальний Q_v – та P_v – процес повинен виконати чотири операції обміну даними.

При застосуванні БОМ–процесів (2.2), (2.3), (2.5) нам потрібно виходячи зі структури на рис. 2.1 та вищезазначених зауважень докладно перевірити, як виглядають відношення між кількістю обчислювальних операцій (RO) та операцій обміну даними (DA). Тривалість виконання операцій RS в (2.2) становить

$$T_{Qv} = \quad 42t_{mult} + 30t_{sum} + 5t_{mod} + 4t_{div} . \qquad (2.6)$$

У (2.3) обчислення тривають

$$T_{Pv+1} = 32\ t_{mult} + 25\ t_{sum} + 4\ t_{div} \quad (2.7)$$

а в (2.5) –

$$T_{Pw} = 32\ t_{mult} + 25\ t_{sum} + 4\ t_{div} + 5Vart_{sum}\ , \quad (2.8)$$

де t_{mult},t_{sum},t_{mod},$t_{di}v$ – відповідні тривалості операцій множення, сумування, визначення модуля змінної та ділення;

Vartsum позначає перехідну тривалість операції $\sum Q_{inz}$, яка залежить від кількості інцидентних ребер;

індекс $1 \le inz \le inz_{max}$ визначається аналізатором топології.

В Q_v –процесі, який відповідно виконує 81 обчислювальну операцію за (2.2) та (2.5), повинні бути виконані наступні 14 операцій обміну даними:

1. Введення $P_{jv,n,0}$ з процесу P_{v-1}.
2. Введення $P_{jv+1,n,0}$; $P_{jv+1,n,1}$; $P_{jv+1,n,2}$; $P_{jv+1,n,3}$; $P_{jv+1,n,4}$ з процесу P_v.
3. Надсилання $Q_{jvn,1}$; $Q_{jvn,2}$; $Q_{jvn,3}$; $Q_{jvn,4}$ процесу P_v.
4. Надсилання $Q_{jvn,1}$; $Q_{jvn,2}$; $Q_{jvn,3}$; $Q_{jvn,4}$ процесу P_{v-1}.

Ці операції обміну даними становлять 14/81, тобто приблизно 17% щодо кількості операцій обчислення. Щодо загального обсягу операцій можна говорити про "концентрацію" операцій обміну даними:

$$C_{DAO} = N_{DAO}/(N_{RO} + N_{DAO}) \quad (2.9)$$

Для Q_v – процесу маємо C_{DAO} = 14/(81 + 14) = 0.147, тобто 14,7% обчислювальних ресурсів повинні виконувати допоміжні операції.

У P_v – процесі, який відповідно виконує 61 обчислювальну операцію за (2.3) та (2.7), повинні бути виконані наступні 18 операцій обміну даними:

1. Надсилання $P_{jv+1,n,1}$; $P_{jv+1,n,2}$; $P_{jv+1,n,3}$; $P_{jv+1,n,4}$ процесу Q_v.
2. Надсилання $P_{jv+1,n,1}$; $P_{jv+1,n,2}$; $P_{jv+1,n,3}$; $P_{jv+1,n,4}$ процесу $Q_{v+!}$.
3. Введення $Q_{jv,n,0}$; $Q_{jv,n,1}$; $Q_{jv,n,2}$; $Q_{jv,n,3}$; $Q_{jv,n,4}$ з процесу Q_v.

4. Введення $Q_{jv+1,n,0}$; $Q_{jv+1,n,1}$; $Q_{jv+1,n,2}$; $Q_{jv+1,n,3}$; $Q_{jv+1,n,4}$ з процесу $Q_{v+!}$.

Ці операції обміну даними становлять 18/61, тобто приблизно 29,5% щодо кількості операцій обчислення. Для Pv–процесу маємо C_{DAO} = 18/(61 + 18) = 0.227, тобто 22,7% обчислювальних ресурсів повинні виконувати допоміжні операції.

В процесі P_w, який відповідно виконує 61+5Var_{inz} обчислювальних операцій згідно з (2.5) та (2.8), мають бути виконані наступні 9+9Var_{inz} операцій обміну даними:

1. Відсилання $P_{jwn1,n,1}$; $P_{jwn1,n,2}$; $P_{jwn1,n,3}$; $P_{jwn1,n,4}$ процесу Q_{jMj}.

2. Відсилання $P_{jv+1,n,1}$; $P_{jv+1,n,2}$; $P_{jv+1,n,3}$; $P_{jv+1,n,4}$ інцидентним процесам Q_{inz}.

3. Введення $Q_{jMj,n,0}$; $Q_{jMj,n,1}$; $Q_{jMj,n,2}$; $Q_{jMj,n,3}$; $Q_{jMj,n,4}$ з процесу Q_{jMj}.

4. Введення $Q_{inz,n,0}$; $Q_{inz,n,1}$; $Q_{inz,n,2}$; $Q_{inz,n,3}$; $Q_{inz,n,4}$ з процесів Q_{inz}.

Змінна *Varinz* = *inzmax* – 1. Для тестових об'єктів *inzmax* = 4 і *Varinz* = 3, тому для оцінок приймаємо кількість операцій обміну даними в P_w як 9+27=36. Ці операції обміну даними становлять (9+9Varinz)/(61+5Varinz), що приблизно дорівнює 47% відносно кількості операцій обчислення. Для процесу Pw отримали C_{DAO} =(9+9*Varinz*)/(61+5*Varinz*+9+9*Varinz*)=0.321, тобто 32,1% залучених обчислювальних ресурсів мають виконувати операції обміну даними.

Загалом, три розглянуті типи процесів матимуть наступні кількості операцій N_{RO} = 81+61+76 = 218, N_{DAO} = 14+18+36 = 68. Щодо БОМ–ДСМ на рівні паралельності РП–1 отримуємо згідно з формулою (2.9) концентрацію операцій обміну даними відносно загальної кількості віртуальних операцій:

$$C_{DAO} = 68/(218+68) = 0.234 \qquad (2.10)$$

Згідно з формулою (2.9), вищезазначеними прикладами та формулою (2.10) випливає, що на 1–ому рівні паралельності "Одне рівняння – один процес" відсутнє оптимальне співвідношення між обсягами обчислювальних

та допоміжних операцій: у середньому для виконання 23,4% віртуальних ресурсів потрібно застосовувати для допоміжних операцій. Аналіз у [6] показав, що при традиційних методах розв'язку це співвідношення може бути ще гірше і сягати 50%.

Незважаючи на це відношення, ми намагаємося визначити, чи існує та як виглядає віртуальне прискорення при паралельному виконанні розглянутих процесів. Припускаємо, що БОМ–ДСМ–структура j–ребра, яка зображена на рисунку 2.1 і описана рівняннями (2.1), реалізується з використанням процесів Q_v, P_v та P_w, які були описані вище. Припускаємо, що крайні тиски визначаються так: $P_1 = P_{w1}$, $P_{M+1} = P_{w2}$ ($n1 = 2$). Потім враховуємо, що кількість процесів Q_v, P_v та P_w для j–ребра становить відповідно M, M–1 та n1=2. Загальний час виконання обчислювальних операцій в Q_v, P_v та P_w процесах при послідовному виконанні визначається як

$$\begin{aligned} T_{Qj} &= \sum T_{Qv} = M_j T_{Qv}, \\ T_{Pj} &= \sum T_{Pv+1} = (M_j - 1) T_{Pv+1}, \\ T_{Pwn1} &= \sum T_{Pw} = 2 T_{Pw}. \end{aligned} \tag{2.11}$$

Час виконання обчислювальних операцій всіх процесів на віртуальному процесорі буде

$$T(1) = M_j T_{Qv} + (M_j - 1) T_{Pv+1} + 2 T_{Pw} \tag{2.12}$$

Звісно, у T(1) немає допоміжних операцій: структура на рисунку 2.1 реалізується на процесорі зі спільним простором адрес, де вектори $\overline{Q_J} = (Q_{j1}, Q_{j2}, \ldots, Q_{jMj})^T$ und $\overline{P_J} = = (P_{w1}, P_{j2}, \ldots, P_{jMj}, P_{w2})^T$ зберігаються.

При паралельній реалізації процесів на $2M_j+1$ віртуальних процесорах ми повинні враховувати час обчислення та час обміну даними згідно з вищенаведеними оцінками. Припустимо, що TDA – це час виконання віртуальної операції обміну даними в системі "Віртуальна паралельна

архітектура обчислювача – віртуальна мережа зв’язку". Припускаємо, що TDA має однакове значення для процесів Q_v, P_v та P_w. Тоді час виконання процесів, відповідальних за розв’язання системи рівнянь (2.1), обчислюється наступним чином:

$$T_{QvDA} = T_{Qv} + damQ*T_{DA}, \quad (2.13)$$

$$T_{PvDA} = T_{Pv+1} + damP*T_{DA}, \quad (2.14)$$

$$T_{PwDA} = T_{Pw} + damP_w*T_{DA}. \quad (2.15)$$

Кількості операцій обміну даними $damQ$, $damP$, $damP_w$ залежать від застосовуваних числових методів. Як було показано вище, для розглянутого БОМ значення цих параметрів наступні: $damQ = 14$, $damP = 18$, $damP_w = 36$.

При паралельній та одночасній реалізації процесів Q_v, P_v та P_w для j–ребра (рис. 2.1) ми повинні визначити максимальний час тривалості типів процесів за наступними формулами:

$$T_{QjDA} = \mathrm{MAX}_1^{Mj} (T_{Qv} + damQ*T_{DA}), \quad (2.16)$$

$$T_{PjDA} = \mathrm{MAX}_1^{Mj-1} (T_{Pv+1} + damP*T_{DA}), \quad (2.17)$$

$$T_{PwjDA} = \mathrm{MAX}_1^{2} (T_{Pw} + damP_w*T_{DA}), \quad (2.18)$$

Тут T_{QjDA}, T_{PjDA} та T_{PwjDA} є тривалістю віртуальних Q_v, P_v та P_w процесів з урахуванням обчислювальних та операцій обміну даними, які вирішують систему рівнянь (2.1) для всього j–ребра (v = 1,2,...,Mj) за двома граничними тисками згідно із блоко–обчислювальним методом (БОМ). Ці формули також застосовуються до інших використовуваних числових методів. З (2.16)...(2.18) випливає, що час вирішення паралельно T(2Mj+1) залежить від часу виконання найбільш завантаженого процесу, тобто

$$T(2M_j+1) = \mathrm{MAX}(T_{QjDA}, T_{PjDA}, T_{PwjDA}). \quad (2.19)$$

Аналіз процесів за формулами (3.5), (3.6) і (3.8) показує, що

$$T(2M_j+1) = T_{QjDA.} \quad (2.20)$$

Припускаючи, що в (2.20) для всіх v=1,2,...,Mj час виконання процесів однаковий, тобто

$$T_{QjDA} = T_{Qv} + damQ*T_{DA.} \quad (2.21)$$

При вищезазначеному значенні damQ = 14 маємо

$$T_{QjDA} = T_{Qv} + 14*T_{DA}$$

Віртуальне прискорення з урахуванням віртуальних операцій обміну даними визначається наступною формулою:

$$S_{virt} = \frac{T(1)}{T(2M_j+1)} \quad (2.22)$$

З урахуванням (2.12), (2.21) і (2.22) ми отримуємо:

$$S_{virt} = \frac{M_j T_{Qv} + (M_j-1)T_{Pv+1} + 2T_{Pw}}{T_{Qv} + damQ*T_{DA}} = M_j \frac{1+\frac{T_{Pv+1}}{T_{Qv}} - {T_{Pv+1}}/{M_j T_{Qv}} + 2{T_{Pw}}/{M_j T_{Qv}}}{1+damQ*\frac{T_{DA}}{T_{Qv}}} \quad (2.22)$$

З (2.23) випливає, що при рівномірному навантаженні на Q– та P–процеси, тобто, $T_{Qv} = T_{Pv} = T_{Pw}$ може бути досягнута максимальна можлива швидкість прискорення:

$$S_{virtmax} = (2M_j + 1)\frac{1}{1+damQ*\frac{T_{DA}}{T_{Qv}}} \quad (2.24)$$

З (2.24) випливає, наскільки важливо мінімізувати $damQ*T_{DA}/\ T_{Qv.}$

2.2.3 Зв’язки між віртуальними процесами, віртуальна мережа обміну інформацією

На рис. 2.1 було зображено взаємодії між процесами Q та P щодо входів і виходів. На рис. 2.2 запропоновано віртуальні PQ– та QP–комутатори, які складаються з звичайних елементів комутації для паралельних обчислень. Змінна Q_v (вихід процесу Q) реплікується та підключається до входів процесів P_v, P_{v+1} (рис. 2.2, а); Змінна Pv+1 (вихід процесу P) також реплікується та підключається до входів процесів Qv, Qv+1 (рис. 2.2, б).

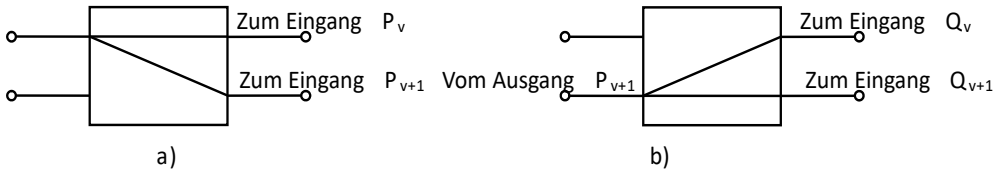


Рисунок 2.2 – Перемикачі віртуального комутатора.

На рис. 2.3 показані PQ– та QP–комутатори, які забезпечують з’єднання між віртуальними паралельними процесами моделі симуляції ребра (рис. 2.1): "Вихід Pv+1 – Входи Qv, Qv+1" та "Вихід Qv – Входи Pv, Pv+1". Віртуальний комутатор для вузлових процесів будується аналогічно. Наприклад, для вузла W1 МДОРП маємо рівняння 2.25 (процес Pw).

$$\frac{dP_{w1}}{dt} = \frac{dP_{1M1}}{dt} = \gamma_{w1}(Q_{1M1} - Q_{21} - Q_{51} - Q_{61}) \qquad (2.25)$$

На рисунку 2.4 показана схема вузлового процесу Pw1 та підключення вихідної змінної Pw1 до входів процесів Q1M1, Q21, Q51, Q61 через комутаційні елементи. Тут змінна Pw1 (PU1) реплікується для передачі в 4 адреси. На входи PU1 змінні надсилаються відповідно до (3.30) через QP–комутатори для ребер Q1, Q2, Q5, Q6.

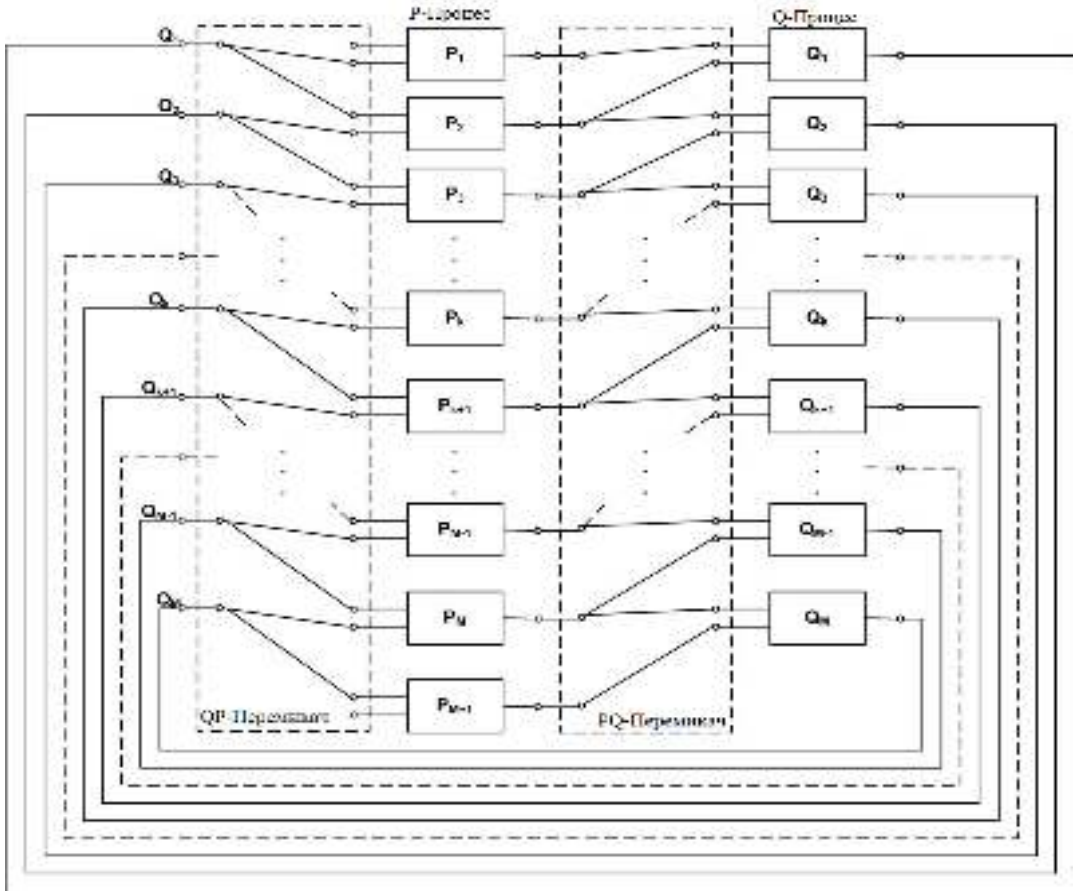


Рисунок 2.3 – Віртуальні QP– та PQ–комутатори

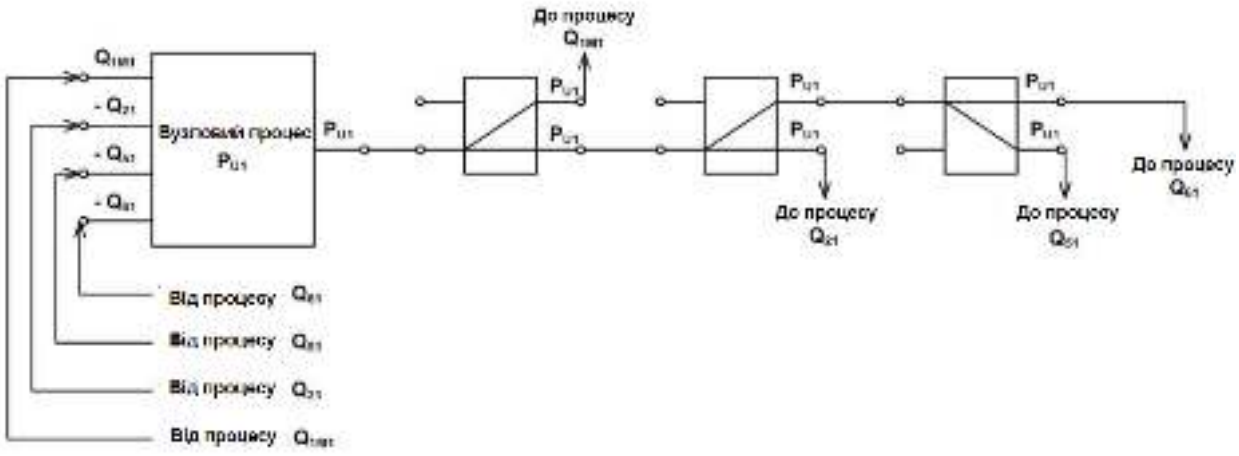


Рисунок 2.4 – Побудова віртуального вузлового комутатора

Для мінімізації часу виконання віртуальних операцій обміну даними $damQT_{DA}$ у (2.24), роботу віртуального комутатора (рис. 2.3) слід організувати наступним чином:

1. З’єднання створюються перед розв’язанням і залишаються незмінними протягом усього періоду симуляції.
2. На кожному кроці ітерації значення компонентів векторів зберігаються та записуються в вихідних регістрах процесів Q та P.
3. Усі процеси синхронізуються в циклі "Обчислення – Збереження" за найдовшим процесом.
4. Для всіх Q– та P–процесів активуються синхронно інструкції Send та Receive; на одному такті поточні значення змінних з’являються на входах процесів Q та P і зберігаються на наступному такті.
5. Запускається i+1–та ітерація розв’язання.

При такій організації функціонування комутатора та паралельного обміну даними Q та P час виконання віртуальних операцій обміну даними визначається наступним чином:

$$T_{DA} = t_{er} + t_{lr} + t_{üb} + t_{em} + t_{rs}, \qquad (2.26)$$

де t_{er} – час запису в регістр у найдовшому Q–процесі;

t_{lr} – час читання з регістру;

$t_{üb}$ – час передачі через мережу зв’язку;

t_{em} – час отримання з лінії зв’язку в регістр;

t_{rs} – час запису з регістру в пам’ять.

2.3 РП–2

2.3.1 БОМ–операції обчислень в $(QP)_{jv}$–БОМ–процесах для j–гілки і МДОРП

На рівні паралелізму РП–2 має відповідність "1 елемент ребра = 1 процесу" означає, що системи рівнянь типу (2.1) для елементу v ($1 \le v \le Mj$) ребра j ($1 \le j \le m$) вирішуються в процесі v–QP. Віртуальна паралельна модель симуляції показана на рис. 2.5.

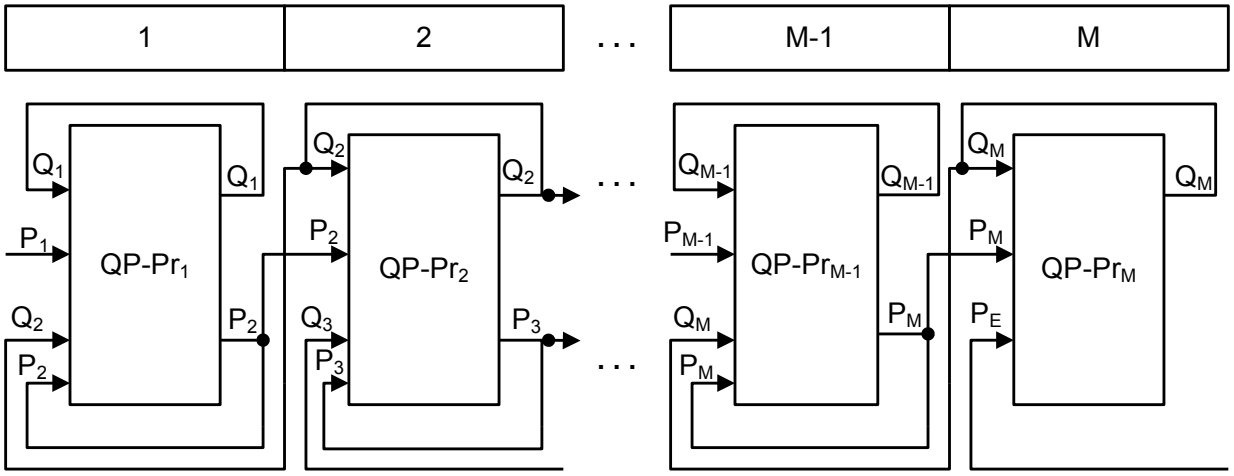


Рисунок 2.5 – Структура віртуальної паралельної Simulation–моделі (ВПSM) на рівні РП–2 «1 елемент гілки – 1 процес»

На основі рис. 2.5 та рівнянь, які потрібно вирішити (2.1), об’єднані QP–процеси для елементів v ребра j (v=1,2,...,Mj–1, Mj) повинні виконати наступні операції:

1. Введення значень тиску Pv від попередніх процесів та значень об’ємного потоку повітря Qv+1 від наступних сусідніх процесів;
2. Розрахунок значень Qv та Pv+1;
3. Вивід значень Qv до попереднього та Pv+1 до наступного сусіднього процесу.

Розглянемо вміст процесів, які повинні реалізувати формули обчислення БОМ.

QP–процес елемента v гілки (рис. 2.5) має вирішити дві рівняння системи (2.1), тобто розв'язати:

$$\begin{cases} \dot{Q}_{jv} = a_j\left(P_{jv} - P_{j,v+1}\right) - \beta_j Q_{jv}\left|Q_{jv}\right| \\ \dot{P}_{j,v+1} = g_j\left(Q_{jv} - Q_{j,v+1}\right) \end{cases}$$

Блоковий m–крок–k–пункт–метод обчислень (БОМ, m=1, k=4) надає наступні обчислювальні формули [1, 8, 9]:

$$U_{n,1} \Longrightarrow Q_{jvn,1} = \frac{\tau}{720}\left(251F_{n,0} + 646F_{n,1} - 264F_{n,2} + 106F_{n,3} - 19F_{n,4}\right)_{Qjv} + Q_{jvn,0};$$

$$U_{n,2} \Longrightarrow Q_{jvn,2} = \frac{\tau}{90}\left(29F_{n,0} + 124F_{n,1} + 24F_{n,2} + 4F_{n,3} - F_{n,4}\right)_{Qjv} + Q_{jvn,0};$$

$$U_{n,3} \Longrightarrow Q_{jvn,3} = \frac{3\tau}{80}\left(9F_{n,0} + 34F_{n,1} + 24F_{n,2} + 14F_{n,3} - F_{n,4}\right)_{Qjv} + Q_{jvn,0};$$

$$U_{n,4} \Longrightarrow Q_{jvn,4} = \frac{2\tau}{45}\left(7F_{n,0} + 32F_{n,1} + 12F_{n,2} + 32F_{n,3} + 7F_{n,4}\right)_{Qjv} + Q_{jvn,0};$$

$$\begin{aligned} (F_{v,n,0})Q_{jv} &= \alpha_j(P_{jv,0} - P_{jv+1,n,0}) - \beta_j Q_{jv,n,0}\left|Q_{jv,n,0}\right| \\ \left(F_{v,n,1}\right)Q_{jv} &= \alpha_j\left(P_{jv,0} - P_{jv+1,n,1}\right) - \beta_j Q_{jv,n,1}\left|Q_{jv,n,1}\right| \\ \left(F_{v,n,2}\right)Q_{jv} &= \alpha_j\left(P_{jv,0} - P_{jv+1,n,2}\right) - \beta_j Q_{jv,n,2}\left|Q_{jv,n,2}\right| \\ \left(F_{v,n,3}\right)Q_{jv} &= \alpha_j\left(P_{jv,0} - P_{jv+1,n,3}\right) - \beta_j Q_{jv,n,3}\left|Q_{jv,n,3}\right| \\ (F_{v,n,4})Q_{jv} &= \alpha_j(P_{jv,0} - P_{jv+1,n,4}) - \beta_j Q_{jv,n,4}\left|Q_{jv,n,4}\right| \end{aligned} \qquad (2.27)$$

$$U_{n,1} \Longrightarrow P_{jvn,1} = \frac{\tau}{720}\left(251F_{n,0} + 646F_{n,1} - 264F_{n,2} + 106F_{n,3} - 19F_{n,4}\right)_{Pjv+1} + P_{jv+1,n,0};$$

$$U_{n,2} \Longrightarrow P_{jvn,2} = \frac{\tau}{90}\left(29F_{n,0} + 124F_{n,1} + 24F_{n,2} + 4F_{n,3} - F_{n,4}\right)_{Pjv+1} + P_{jv+1,n,0};$$

$$U_{n,3} \Longrightarrow P_{jvn,3} = \frac{3\tau}{80}\left(9F_{n,0} + 34F_{n,1} + 24F_{n,2} + 14F_{n,3} - F_{n,4}\right)_{Pjv+1} + P_{jv+1,n,0};$$

$$U_{n,4} \Longrightarrow P_{jvn,4} = \frac{2\tau}{45}\left(7F_{n,0} + 32F_{n,1} + 12F_{n,2} + 32F_{n,3} + 7F_{n,4}\right)_{Pjv+1} + P_{jv+1,n,0};$$

$$\begin{aligned} (F_{v,n,0})_{Pjv+1} &= g_j(Q_{jv,n,0} - Q_{jv+1,n,0}) \\ (F_{v,n,1})_{Pjv+1} &= g_j(Q_{jv,n,1} - Q_{jv+1,n,1}) \\ (F_{v,n,2})_{Pjv+1} &= g_j(Q_{jv,n,2} - Q_{jv+1,n,2}) \end{aligned} \qquad (2.28)$$

$$(F_{v,n,3})_{Pjv+1} = g_j(Q_{jv,n,3} - Q_{jv+1,n,3})$$
$$(F_{v,n,4})_{Pjv+1} = g_j(Q_{jv,n,4} - Q_{jv+1,n,4})$$

Значення $Q_{vn,0}$ та $P_{v+1,n,0}$ у нульовій точці блоку n визначаються як значення в останній точці попереднього блоку для n > 1 (tn–1,k = tn,0). Якщо розглядається j–ребро в мережі МДОРП, то змінна $P_{jMj+1,n,k}$ повинна бути визначена як значення тиску в інцидентному вузлі після вузлового рівняння.

$$\frac{dP_{wn1}}{dt} = g_{wn1}(Q_{jMj} - \sum_{inz} Q_{inz}) \tag{2.28}$$

де Q_{inz} – струми, що входять в вузол;

wn1 – це порядковий номер кінцевого вузла (умови берегового типу n1). Для числового розв'язання рівнянь типу (2.29) формули BAV з wn1 = 1, 2, ..., n1 виглядають так [1, 8, 9]:

$$U_{n,1} \Longrightarrow P_{jwn1,n,1} = \frac{\tau}{720}\left(251F_{n,0} + 646F_{n,1} - 264F_{n,2} + 106F_{n,3} - 19F_{n,4}\right)_{Pjwn1} + P_{jwn1,n,0};$$
$$U_{n,2} \Longrightarrow P_{jwn1,n,2} = \frac{\tau}{90}\left(29F_{n,0} + 124F_{n,1} + 24F_{n,2} + 4F_{n,3} - F_{n,4}\right)_{Pjwn1} + P_{jwn1,n,0};$$
$$U_{n,3} \Longrightarrow P_{jwn1,n,3} = \frac{3\tau}{80}\left(9F_{n,0} + 34F_{n,1} + 24F_{n,2} + 14F_{n,3} - F_{n,4}\right)_{Pjwn1} + P_{jwn1,n,0};$$
$$U_{n,4} \Longrightarrow P_{jwn1,n,4} = \frac{2\tau}{45}\left(7F_{n,0} + 32F_{n,1} + 12F_{n,2} + 32F_{n,3} + 7F_{n,4}\right)_{Pjwn1} + P_{jwn1,n,0};$$

$$(F_{n,0})_{Pjwn1} = g_{wn1}(Q_{jMj,n,0} - \sum Q_{inz,n,0}) \tag{2.30}$$
$$(F_{n,1})_{Pjwn1} = g_{wn1}(Q_{jMj,\,n,1} - \sum Q_{inz,n,1})$$
$$(F_{n,2})_{Pjwn1} = g_{wn1}(Q_{jMj,\,n,2} - \sum Q_{inz,n,2})$$
$$(F_{n,3})_{Pjwn1} = g_{wn1}(Q_{jMj,\,n,3} - \sum Q_{inz,n,3})$$
$$(F_{n,4})_{Pjwn1} = g_{wn1}(Q_{jMj,\,n,4} - \sum Q_{inz,n,4})$$

На рівні паралелізму 2 (РП–2) ми маємо в сумі всього $\sum M_j$ процесів QP (2.27)...(2.30). n1 процесів Pw (2.30), які розглядалися окремо на рівні РП–1, тепер належать до P–частини процесів QP. З урахуванням граничних умов n2

$$\begin{pmatrix} P_{wi}=P_{AEJ}^{1}(QJ) \\ P_{wi}=P_{AEJ}^{n2}(QJ) \end{pmatrix} \quad (2.31)$$

і граничні умови n3

$$\begin{pmatrix} P_{wi}=P_{(n1+n2)+1}=P_{ATM}^{1}=const \\ P_{wi}=P_{n}=P_{ATM}^{n3}=const \end{pmatrix} \quad (2.31)$$

повинні спільно вирішувати алгебраїчну систему рівнянь, що виникла в результаті дискретизації за БОМ.

2.3.2 Порівняння об'ємів операцій обміну даними і обчислювальних операцій, вивід формул для оцінки віртуального прискорення

Тривалість операцій над правими частинами (ПЧ–операції) рівнянь (2.27), (2.28) в $(QP)_{jv}$–БОМ–процесі з врахуванням отриманих вище оцінок (2.6), (2.7)

$$T_{Qv} = 42t_{mult} + 30t_{sum} + 5t_{mod} + 4t_{div}$$
$$T_{Pv+1} = 32\ t_{mult} + 25\ t_{sum} + 4\ t_{div}$$

обчислюється за формулою

$$T_{vQP} = T_{Qv} + T_{Pv+1} = 42t_{mult} + 30t_{sum} + 5t_{mod} + 4t_{div} + 32\ t_{mult} + 25\ t_{sum} + 4\ t_{div},$$

тобто

$$T_{vQP} = 74t_{mult} + 55t_{sum} + 5t_{mod} + 8t_{div}. \quad (2.33)$$

Для останнього елемента j–ребра рівняння (2.27), (2.30) мають бути вирішені в QP–процесі, тобто тривалість із врахуванням (2.6), (2.8)

$$T_{wQP} = T_{Qv}+T_{Pw} = 42t_{mult}+ 30t_{sum}+5t_{mod} + 4t_{div} +32\ t_{mult} + 25\ t_{sum}+ 4\ t_{div}+ 5Vart_{sum},$$

тобто

$$T_{wQP} = 74t_{mult} + 55t_{sum} +5t_{mod} + 8t_{div} + \ 5Vart_{sum}\ , \qquad (2.34)$$

де t_{mult}, t_{sum}, t_{mod}, t_{div} відповідають тривалості операцій множення, сумування, визначення модуля змінної та ділення;

$Vart_{sum}$ позначає змінний час операції $\sum Q_{inz}$, який залежить від кількості інцидентних ребер;

індекс $1 \le inz \le inzmax$ визначається топологічним аналізатором.

Для тестових об'єктів МДОРП, а також у більшості промислових МДОРП маємо inzmax=4.

В РП–2 відношення між обсягами обчислень та обміну даними змінюється в напрямку зниження останніх. З рис. 2.5 та рівнянь (2.1) випливає, що об'єднані процеси (QP)jv–БОМ для елементів v–ребра j–ребра (v=1,2,...,Mj–1, Mj) мають структуру, показану на рис. 2.6, і повинні виконувати наступні операції з обміну даними та обчислень:

- введення значень тиску Pv та потоку повітря Qv+1;
- обчислення значень Qv– та Pv+1;
- вивведення значень Qv та Pv+1.

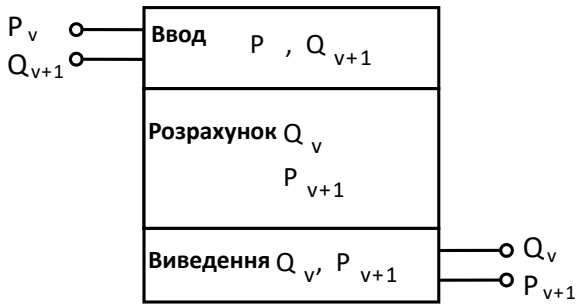


Рисунок 2.6 – Структура (QP)jv –процесу.

При нумерації змінних Qjv та Pjv, яка показана на рис. 2.5, 2.6 та у системі рівнянь (2.1), об'єднані процеси (QP)jv обчислюють змінні Qv та Pv+1. Іншими словами, елементи ребра j для МДОРП мають номерацію v = 1, 2, 3, ..., Mj–1, Mj, Mj+1. Таким чином, тиск Pj1 та PjMj+1 у мережі буде належати відповідним початковим та кінцевим вузлам ребра j і обчислюватиметься як розв'язок вузлових рівнянь типу (2.29) у процесах (QP)jv–БАВ для v=1 та v=Mj–елементів ребра j.

При використанні процесів БОМ (2.27), (2.28), (2.30) виходячи зі структури на рис. 2.5, 2.6 та вищевказаних зауважень, слід ретельно перевірити, які відношення існують між обсягами операцій обчислень (RO) та операцій обміну даними (DA).

У (QP)jv БОМ процесі, який, згідно з (2.27), (2.28), виконує 81+61=142 операції обчислень, порівняно з РП–1 має виконати менше операцій обміну даними: 9 змінних $P_{jv+1,n,0}$; $P_{jv+1,n,1}$; $P_{jv+1,n,2}$; $P_{jv+1,n,3}$; $P_{jv+1,n,4}$ та $Q_{jvn,1}$; $Q_{jvn,2}$; $Q_{jvn,3}$; $Q_{jvn,4}$ обчислюються в (QP)jv БОМ процесі, в розділі Q, і залишаються в процесі для виконання розділу P. Так само P–розділ (QP)jv БОМ процесу не потрібно відправляти 4 обчислені змінні $P_{jv+1,n,1}$; $P_{jv+1,n,2}$; $P_{jv+1,n,3}$; $P_{jv+1,n,4}$ і не вводити 5 змінних Qjv $Q_{jv,n,0}$; $Q_{jv,n,1}$; $Q_{jv,n,2}$; $Q_{jv,n,3}$; $Q_{jv,n,4}$. Отже, в РП–2 (QP)jv БОМ процес має виконати наступні 14 операцій обміну даними:

1. Введення $P_{jv,n,0}$ з процесу (QP)jv–1.
2. Відправлення $Q_{jvn,1}$; $Q_{jvn,2}$; $Q_{jvn,3}$; $Q_{jvn,4}$ до процесу (QP)jv–1.
3. Відправлення $P_{jv+1,n,1}$; $P_{jv+1,n,2}$; $P_{jv+1,n,3}$; $P_{jv+1,n,4}$ до процесу (QP)jv+1.

4. Введення $Q_{jv+1,n,0}$; $Q_{jv+1,n,1}$; $Q_{jv+1,n,2}$; $Q_{jv+1,n,3}$; $Q_{jv+1,n,4}$ з процесу (QP)jv+1.

Ці операції обміну даними становлять 14/142, що дорівнює приблизно 9,8% відносно обсягу операцій обчислень. Відносно загального обсягу операцій можна, так само як і в РП–1, говорити про "концентрацію" операцій обміну даними.

$$C_{DAO} = N_{DAO}/(N_{RO} + N_{DAO}). \quad (2.35)$$

Для (QP)jv БОМ процесу на РП–2 маємо C_{DAO} = 14/(142 + 14) = 0,09; тобто 9,0% залучених обчислювальних ресурсів призначено для виконання допоміжних операцій.

У (QP)jw БОМ процесі виконуються наступні рівняння:

$$\begin{cases} \dot{Q}_{jv} = a_j\left(P_{jv} - P_{j,v+1}\right) - \beta_j Q_{jv}\left|Q_{jv}\right| \\ \dot{P}_{j,v+1} = g_j\left(Q_{jv} - Q_{j,v+1}\right) \end{cases} \quad (2.36)$$

Відповідно до (2.27), (2.30) вирішуються для останнього елемента v=Mj ребра j–типу. При цьому виконується 81+61+5Varinz обчислень, і повинно бути здійснено наступні 9+5Varinz операцій передачі даних:

1. Відправлення $P_{jwn1,n,1}$; $P_{jwn1,n,2}$; $P_{jwn1,n,3}$; $P_{jwn1,n,4}$ інцидентним $(QP)_{inz}$ БОМ процесам.

2. Введення $P_{jMj,n,0}$ з $(QP)_{jMj-1}$ БОМ процесу.

3. Введення $Q_{inz,n,0}$; $Q_{inz,n,1}$; $Q_{inz,n,2}$; $Q_{inz,n,3}$; $Q_{inz,n,4}$ з $(QP)_{inz}$ БОМ процесів.

4. Відправлення $Q_{jMjn,1}$; $Q_{jMjn,2}$; $Q_{jMjn,3}$; $Q_{jMjn,4}$ до $(QP)_{jMj-1}$ БОМ процесу.

Змінна Varinz = inzmax – 1. Для тестових об’єктів inzmax = 4 і Varinz = 3, тому для оцінок ми використовуємо кількість операцій обробки даних (DA) в (QP)jw БОМ процесі в розмірі 9+15=24. Ці операції DA становлять (9+5Varinz)/(142+5Varinz), що приблизно дорівнює 15,3% відносно кількості

операцій обробки даних. Для (QP)jw БОМ процесу маємо C_{DAO} =(9+5Varinz)/(142+5Varinz+9+5Varinz)=24/181=0.13, тобто 13% участь обчислювальних ресурсів повинні виконувати операції обробки даних.

З рівняння (2.36) та вищевказаних прикладів випливає, що на 2–му рівні паралельності "Один елемент – один (QP)–БОМ–процес" відсоток операцій обробки даних в порівнянні з операціями обробки даних стає кращим: від 9,8% до 15,3% від віртуальних ресурсів повинні бути використані для операцій обробки даних.

Отримаємо формули для оцінки віртуального прискорення при паралельному виконанні розглянутих процесів. Припустимо, що БОМ–ДСМ, зображений на рис. 2.5 і описаний рівняннями (2.1), реалізує структуру ВПСМ для j–краю із зазначеними вище (QP)jv– та (QP)jw БОМ процесами. Припустимо, що тиск на границі в початковому вузлі v=1, Pj1 = Pjw1 відомий з розв'язку рівнянь МДОРП. Значення Qjv, Pjv+1 для v=1,2,...,Mj–1 обчислюються в процесах (QP)jv–БОМ (2.27), (2.28). У випадку останнього елемента j–краю v=Mj, витрати повітря QjMj та тиск PjMj+1 = Pw в кінцевих вузлах обчислюються в процесі (QP)jw–БОМ (2.27), (2.30).

Загальний час обчислень у процесах (QP)jv– та (QP)jw–БОМ для j–краю при послідовному виконанні складається

$$T_{QPj}= \sum T_{vQP} = (M_j–1)(T_{Qv} + T_{Pv+1}),\ v=1,2,...M_j–1 \qquad (2.37)$$

$$T_{QPwj}= T_{QMj} + T_{PMj+1}=T_{wQP},v=M_j,\ T_{PMj+1}=T_{Pw}. \qquad (2.38)$$

де T_{QPwj} – це час обчислення тиску вузла, яке виконується для останнього елементу v=M_j ребра j за допомогою (2.27), (2.30) і визначається за формулою (2.34).

Час виконання обчислень усіх процесів на віртуальному процесорі буде

$$T(1) =T_{QPj} + T_{QPwj} = (M_j–1)(T_{Qv} + T_{Pv+1}) + T_{QMj} + T_{Pw} \qquad (2.39)$$

При паралельній реалізації віртуальних (QP)jv– та (QP)jw–процесів БОМ, ми повинні врахувати час обчислень відповідно до (2.33), (2.34) та час DA згідно з вищезазначеними оцінками. Як і в РП–1, час TDA є тривалістю віртуальної операції DA в системі ‘Віртуальна паралельна архітектура обчислювальних машин – віртуальна мережа зв’язку’. Припустимо, що TDA для (QP)jv– та (QP)jw–процесів БОМ має однакове значення. Тоді тривалість процесів, відповідальних за вирішення системи рівнянь (2.1), обчислюється наступним чином:

$$T_{vQPDA} = T_{vQP} + damQP*T_{DA}\,, \tag{2.40}$$

$$T_{wQPDA} = T_{wQP} + damQP_w*T_{DA}. \tag{2.41}$$

Кількість операцій обміну даними damQP та damQPw залежать від використовуваних числових методів. Як було показано вище, для розглянутого тут БАВ ці параметри мають значення: $damQP = 14,\ damQP_w = 24.$

При паралельній реалізації (QP)jv– та (QP)jw–БОМ–процесів для j–ребра з v=1,2,...,Mj−1 (рис. 2.5), згідно з (2.40), (2.41) маємо визначити максимальну тривалість видів процесів. Процеси (QP)jv–БОМ однакові для всіх v=1,2,...,Mj−1, тому ми можемо використовувати наступну оцінку:

$$T_{QPjDA} = MAX_1^{Mj-1}\,(T_{vQP} + damQP*T_{DA}) = T_{vQP} + damQP*T_{DA}\,. \tag{2.42}$$

У постановці завдання з обчисленням єдиного тиску на краю PMj+1 = Pw в кінцевому вузлі j–ребра ми маємо:

$$T_{QpwjDA} = T_{wQP} + damQP_w*T_{DA}\,. \tag{2.43}$$

Тут T_{QPjDA} та T_{QPwjDA} – це тривалість віртуальних (QP)jv– та (QP)jw–БОМ–процесів з урахуванням обчислень та операцій обміну даними, які вирішують систему рівнянь (2.1) для всього j–ребра (v = 1,2,...,Mj) з

врахуванням значення тиску на краю, здійснюючи блочні методи (БОМ). Ці формули також залишаються актуальними для інших використовуваних числових методів. З (2.42), (2.43) випливає, що тривалість T(Mj) паралельного вирішення залежить від часу виконання найбільш завантаженого процесу, тобто:

$$T(M_j) = \text{MAX}(T_{QPjDA}, T_{QPwjDA}). \quad (2.44)$$

Аналіз (QP)jv–процесів за формулами (2.27), (2.28) та (QP)jw–процесів за формулами (2.27), (2.30) показує, що тривалість обчислень для Qv і QMj за формулою (2.27) є однаковою. Обчислення Pjv за формулою (2.28) та Pjw за формулою (2.30) відрізняються за обсягом операцій у виразі ∑Qinz у рівнянні Pjw системи (2.36). Також різняться обсяги операцій обміну даними, наприклад, для розглянутих вище прикладів маємо damQP = 14, damQPw= 24. З цих оцінок випливає, що

$$T_{wQP} + damQP_w * T_{DA} > T_{vQP} + damQP * T_{DA}$$

та результат операції (2.44) має виглядати таким чином:

$$T(M_j) = T_{QPwjDA} = T_{wQP} + damQP_w * T_{DA} \quad (2.45)$$

Враховуючи вище зазначене damQPw = 24, отримуємо:

$$T_{QPwjDA} = T_{wQP} + 24 * T_{DA} \quad (2.46)$$

Віртуальне прискорення з урахуванням віртуальних операцій обробки даних визначається формулою

$$S_{virt} = \frac{T(1)}{T(M_j)}. \quad (2.47)$$

З урахуванням (2.39), (2.43) ми отримуємо

$$S_{virt} = \frac{(M_j - 1)(T_{Qv} + T_{Pv+!}) + T_{QMj} + T_{Pw}}{T_{wQP} + damQP_w * T_{DA}} \quad (2.48)$$

Припустимо, що $T_{Qv} + T_{Pv+1} = T_{vQP}$, $T_{QMj} + T_{Pw} = T_{wQP}$ і $T_{vQP} = T_{wQP}$. Після підстановки цих змінних у (2.48) і кількох перетворень отримаємо наступну формулу:

$$S_{virt} = \frac{(M_j - 1)T_{wQP} + T_{wQP}}{T_{wQP} + damQP_w * T_{DA}} \quad (2.49)$$

З (2.49) випливає, що при однаковому навантаженні (QP)jv– та (QP)jw–БОМ процесів, тобто TvQP= TwQP, може бути досягнута максимальна можлива швидкодія:

$$S_{virtmax} = M_j \frac{1}{1 + damQP_w * \frac{T_{DA}}{T_{wQP}}}. \quad (2.50)$$

Вище ми визначили, що

$$T_{wQP} = 74t_{mult} + 55t_{sum} + 5t_{mod} + 8t_{div} + 5Vart_{sum}$$

і damQPw = 24, при цьому 5Var=15. З цими даними ми маємо

$$T_{wQP} = 74t_{mult} + 70t_{sum} + 5t_{mod} + 8t_{div}$$

і співвідношення $24T_{DA} / T_{wQP} = T_{DA} / (3.08t_{mult} + 2.9t_{sum} + 0.2t_{mod} + 0.33t_{div})$. Отримуємо для цих відносин значення приблизно $t_{mult} = 4t_{sum}$; $t_{div} = 5t_{sum}$; $t_{mod} = t_{sum}$

$$S_{virtmax} = M_j \frac{1}{1+\frac{T_{DA}}{16.7 t_{sum}}}. \quad (2.51)$$

З (2.50), (2.51) випливає, наскільки важливо зменшити значення виразу damQPw*TDA/ TwQP.

2.3.3 Схеми зв'язку між віртуальними процесами, віртуальна мережа зв'язку

Схема зв'язку між віртуальними процесами випливає зі структури VPSM (рисунок 2.5). Процес (QP)jv для елементу v на ребрі j має схему зв'язку з сусідніми процесами (QP)jv–1 і (QP)jv+1, яка зображена на рисунку 2.7 (v=k).

Рисунок 2.7 – Схема зв'язків (QP)jv–процесу з сусідніми процесами (v=k)

Підключення до сусідніх процесів реалізується рівномірно за допомогою комутаційних елементів 2×2 з перехресним підключенням (див. рисунок 2.7, входи зображені справа). На цьому рівні паралелізму процеси вузлів Pw об'єднуються з будь–яким процесом Q інцидентного ребра. На рисунку 2.8 показана схема підключення для вузла w1 тестового об'єкта.

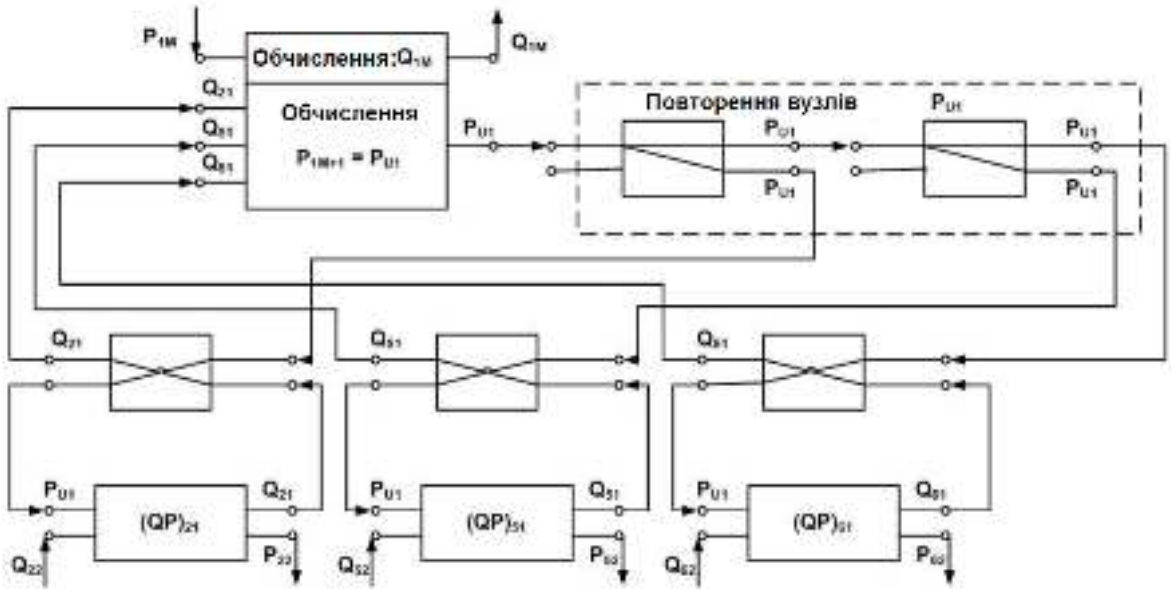


Рисунок 2.8 – Зв’язки вузлового процесу P_{U1} (P_{w1}) з процесами інцидентеих гілок

Схеми підключення, зображені на рисунках 2.7 і 2.8, можна легко розширити для ребер та всього МДОРП. Створення віртуальної мережі зв’язку передбачено як функція інтерфейсу користувача паралельного середовища моделювання. Ця функція буде використовувати інформацію від аналізатора топології та генератора рівнянь.

2.4 РП–3

2.4.1 Структура віртуальних QP–процесів з підпорядкуванням „Система рівнянь j–гілки – процес“.

На рис. 2.9–a показано j–гілку мережі з початковим вузлом „атмосфера“ (гранична умова Pj1=PA) і кінцевим вузлом w1(u1, гранична умова PjMj+1 = Pu1=Pw1) [6]. На рис. 2.9–b показано структуру (QP)j– і (QP)jw –БОМ–процесів, які утворюють загальний процес j–гілки. (QP)j– БОМ–процес реалізує ланцюг БОМ– формул для заданого j (на рис.2.9 j=1) і поточних номерів елементів v=1, 2,..., Mj . Останній Mj–елемент j–гілки використовує

граничне значення PjMj, що обчислюється в (Mj – 1)–елементі і потребує величину PjMj+1=Pu1=Pw1, яка обчислюється (QP)jw– БОМ–процесом.

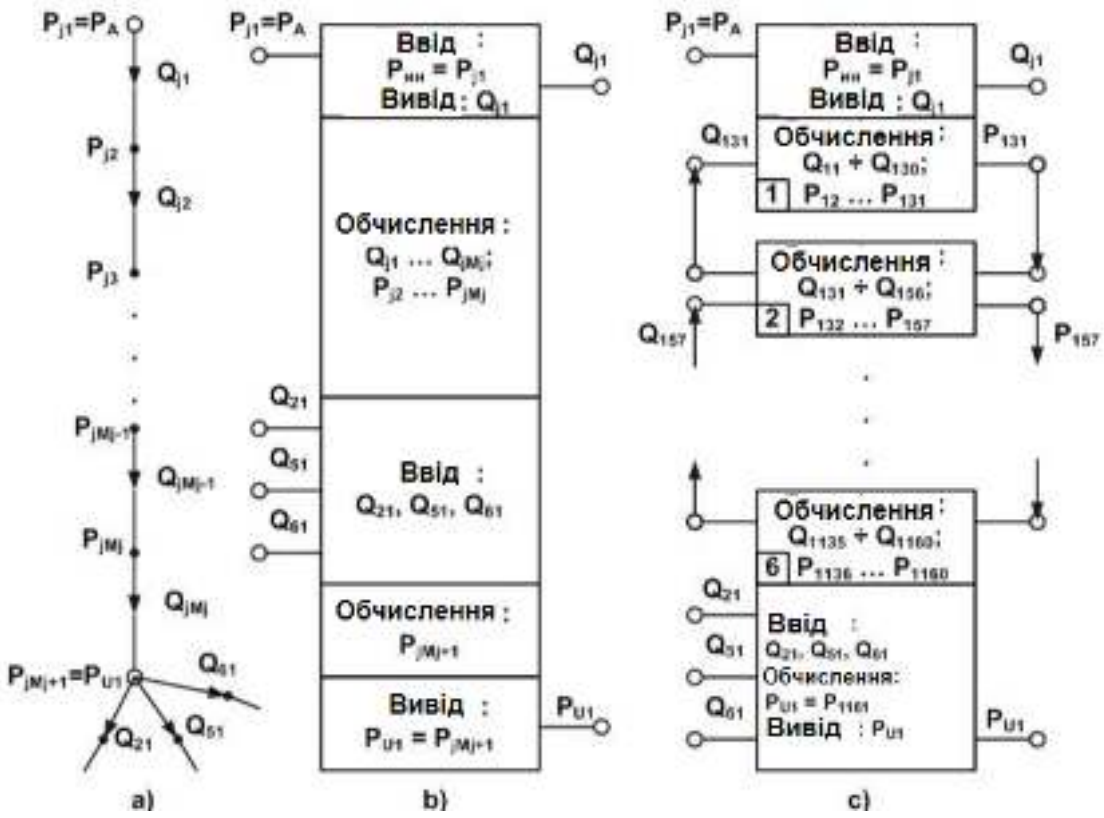


Рисунок 2.9 – Структури віртуальних QP–процесів на рівні РП–3

На рисунку 2.9 a – j–гілка МДОРП (вузли P11=PA і U1=W1 при j=1 тестової мережі); b – структура належних j–гілці (QP)j– та(QP)jw –БОМ–процесів; c – структура належних j–гілці (QP)j– та (QP)jw–БОМ–процесів при підпорядкуванні „lmin–частка гілки – процес“ (Q1–гілка розділена на 6 lmin–часток).

2.4.2 Порівняння об’ємів операцій обміну даними і обчислень, оцінка віртуального прискорення

В [9] показано, що на РП–3 варіант "1 вузол – 1 процес", де гілка j апроксимується Mj елементами, поліпшується співвідношення між обсягами обчислень та DA–операцій для всіх країв (незалежно від застосованого

числового методу). З проведеного вище аналізу для РП–2 видно, що при використанні БОМ це співвідношення буде ще кращим. Процес, підпорядкований гілці, вирішує 2Mj рівнянь і виконує такі допоміжні операції:

– введення Pj1 з вузлового процесу інцидентного початкового вузла (наприклад, Q1: P11 = PA – атмосферний тиск, рис. 2.9а);

– введення повітряних потоків з початкових елементів ребер, які інцидентні кінцевому вузлу j–гілки (для Q1 – вузол U1 та повітряні потоки Q21, Q51, Q61);

– відправлення повітряного потоку Qj1 процесу початкового вузла (для Q1 це не виконується);

– відправлення обчисленого в кінцевому вузлі ребра Qj тиску PjMj+1 (для j = 1: P1M1+1 = PU1) процесам, які обчислюють повітряні потоки Qj1 в інцидентних ребрах (для j = 1 це елементи Q21, Q51 та Q61).

З цього випливає, що кількість вспоміжних операцій між процесами залишається рівною 4. Це є принаймні в Mmin разів краще, ніж на РП–2 [6]. У порівнянні з варіантом 1 тут частка операцій обміну даними вища, оскільки для кожного ребра ми призначили більшу кількість процесів (Mj/Mmin). При цьому досягнута краща рівномірність навантаження на процеси.

Аналіз показує, що час циклу розв’язання (LZ) рівнянь j–гілок на рівні "Ребро – Процес" буде

$$T_{jLZ} = M_j T_{QPj}, \tag{2.52}$$

де T_{QPj} – це тривалість процесу QP, яка залежить від типу рівнянь j–гілки та формул обчислення БОМ. Реалізація m процесів в системі з одним віртуальним процесором для всього МДОРП вимагатиме наступний час обчислень:

$$T_{LZ}(1) = \sum_{j=1}^{m} M_j T_{QPj}. \tag{2.53}$$

При паралельній реалізації у віртуальній м–процесорній системі тривалість LZ для МДОРП залежатиме від тривалості граничного процесу lmax і часу DA (РП–3, 1–й варіант).

$$T_{LZ}(m) = \max_{j=q}\left(M_j T_{QPj}\right) + T_{3DA}^{(1)}. \quad (2.54)$$

Віртуальне прискорення для РП–3 Варіант 1 становить

$$S_{v3}^{(1)} = \frac{T_{LZ}(1)}{T_{LZ}(m)} = \frac{\sum_{j=1}^{m} M_j T_{QPj}}{\max\limits_{j=q}\left(M_j T_{QPj}\right) + T_{3DA}^{(1)}}. \quad (2.55)$$

Операція MAX виконується відносно даних, що містяться в таблиці TABLAST і повертає:

$$\max_{j=q}\left(M_j T_{QPj}\right) = M_{jmax} T_{QPjmax}$$

З цим результатом можна перетворити формулу (2.55) так:

$$S_{v3}^{(1)} = \frac{T_{LZ}(1)}{T_{LZ}(m)} = \frac{1 + \left(\sum_{j=1}^{m} M_j T_{QPj}\right)/M_{jmax} T_{QPjmax}}{1 + \frac{T_{3DA}^{(1)}}{M_{jmax} T_{QPjmax}}}. \quad (2.56)$$

де jmax – це номер критичного краю lmax (v ≠ jmax).

З формули (2.56) видно, що організація обчислень для обробки даних є вирішальним фактором для ефективності підходів паралелізації. Максимальна кількість операцій передачі даних у РП–3 залежить від кількості вузлів в МДОРП. У найгіршому випадку з послідовною організацією передачі даних ми отримаємо:

$$T_{3DA}^{(1)} = n * T_{UDA} \quad (2.57)$$

При паралельному виконанні цих операцій на всіх вузлах отримаємо

$$T_{3DA}^{(1)} = T_{UDA},$$

де T_{UDA} – час виконання операцій взаємодії з даними (DA) у процесі краю j, який обчислює тиск вузла P_{Ui} *(i = 1, 2, …, n)*.

2.4.3 Схема з'єднань між віртуальними процесами та віртуальна мережа зв'язку.

Зі структури процесів на малюнку 2.9 (b, c) видно, що схеми з'єднань між процесами j (Варіант 1 РП–3) та процесами частинних краєвих ребер, інцидентних вузлу (Варіант 2 РП–3), схожі і повторюють схему на малюнку 2.7 для (QP)jv–процесу (v=k). Процеси структури на малюнку 2.9c включають у себе Mmin QP–процесів РП–2 (процес QPjv слід розглядати як підпроцес відносно процесу РП–3), які обчислюють Q–, P–змінні ребра j від v=1 до Mmin (перший процес), v=Mmin+1 до v=2Mmin (другий процес) і так далі. Відповідно до схеми з'єднань, останній підпроцес QPjMmin відправляє PjMmin+1 першому підпроцесу QPjMmin+1 сусіднього процесу та отримує від нього QjMmin+1. Точно так само останній підпроцес QPj2Mmin сусіднього процесу відправляє Pj2Mmin+1 своєму сусідові та отримує від нього Qj2Mmin+1. Якщо кількість елементів в процесах менша за Mmin, то схема з'єднань залишається без змін: передача значень тиску та об'ємного потоку здійснюється за допомогою елементів перемикачів 2×2. Їхня кількість на одному ребрі дорівнює NPj – 1 (або NPj, якщо передбачено NPj+1 процесів). На рисунку 2.10 показано схему з'єднань між процесами РП–3 для тестового об'єкта [6]: процес 2 (рисунок 2.9c) ребра j=1 з'єднаний з сусідами, процесом 1 та процесом 3, за допомогою перехресно ввімкнених елементів перемикачів.

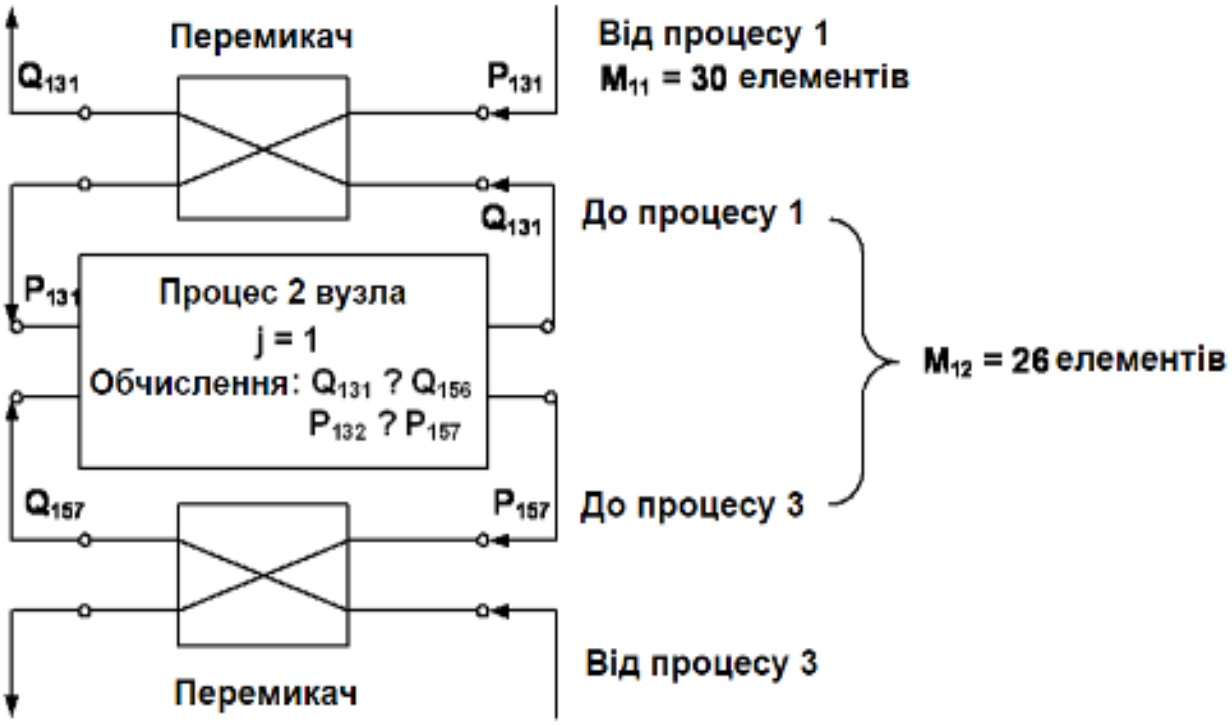


Рисунок 2.10 – Схема зв’язків між РП–3–процесами

Головні процеси мають таку кількість підпроцесів РП–2:

– процес 1 обробляє M11 = 30 елементів (Mmin), тобто має 30 підпроцесів, які обчислюють Q11…Q130, P12…P131;

– процес 2 обробляє M12 = 26 елементів (<Mmin), тобто має 26 підпроцесів, які обчислюють Q131…Q156, P132…P157;

– процес 3 обробляє M13 = 26 елементів (<Mmin), тобто має 26 підпроцесів, які обчислюють Q157…Q182, P158…P183.

Отже, віртуальний комутатор (віртуальна мережа зв’язку) віртуальної паралельної моделі симуляції РП–3 буде аналогічно побудований до комутатора РП–2 з використанням (2x2) елементів перемикання. Для мінімізації загального часу обміну даними на РП–3 (Варіант 2) рекомендується мінімізувати кількість процесів у процесорах, максимізуючи при цьому Mmin, щоб мати менше елементів у схемах зв’язку. Це призведе до ефективнішого використання віртуальних ресурсів.

$$T_{3DA}^{(2)} = (N_P - n) \cdot T_{zpDA} + nT_{UDA}$$

Для віртуального комутатора необхідний блок управління, який може керувати зберіганням даних, які передаються, у регістрах буфера, а також координувати одночасну паралельну передачу даних між сусідніми процесами.

2.5 РП–4

2.5.1 Порівняння об’ємів операцій обміну даними і обчислень, оцінка віртуального прискорення

РП–4 розбиває граф МДОРП на набір частинних графів Gs. При цьому намагається мінімізувати кількість зв’язків між частинними графами. Рівняння систем, що належать кожному частковому графу, вирішуються в MIMD–процесах. Кількість процесів залежить від кількості часткових графів. Структура процесів БОМ (Багатооператорний модуль, MIMD) відображає топологію часткових графів. На прикладі розглянемо розбиття тестового об’єкта на три фрагмента (рис. 2.11):

– F1 – з ребрами Q1, Q2, Q7, Q8;
– F2 – з ребрами Q3, Q4;
– F3 – з ребрами Q5, Q6.

Ребра Q2 та Q7 входять до фрагмента F1 та з’єднують фрагменти F1 – F2, F2 – F3. Зв’язок між фрагментами F1 та F3 реалізується за допомогою вузлів U1, U4.

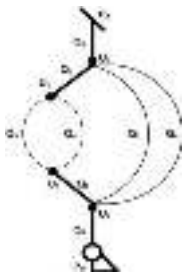


Рисунок 2.11 – Розділення графу тестового МДОРП на три фрагменти

Обчислювальні операції БОМ в РП–4 виконуються щодо підсистем рівнянь (F1, F2, F3), визначених як в РП–2, РП–3, та призначених відповідним ланцюгам процесів (QP)jv та (QP)jw. На рис. 2.12 представлені блок–схема та з'єднання процесів, які відповідають вказаним фрагментам F1, F2, F3 [9].

З'єднання F1 з F2:

1) Для обчислень Q3 та Q4 надсилається PU2;
2) Q71 надсилається до Q3–підпроцесу для обчислення PU3;
3) PU2 отримує PU3 для обчислення Q71;
4) Q31 та Q41 отримуються для обчислення PU2.

З'єднання F1 з F3:

1) PU1 та PU4 надсилаються для обчислень Q5 та Q6;
2) Отримано Q51, Q61 та Q5M5, Q6M6 для обчислення PU1 та PU4.

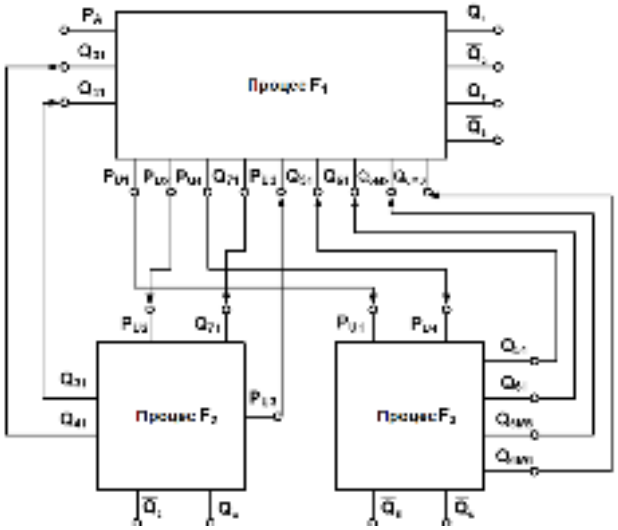


Рисунок 2.12 – Блок–схема процесів, підпорядкованих фрагментам графу

Фрагменти F2 та F3 не мають прямого з'єднання, операції обміну даними відбуваються через фрагмент F1.

Оцінку взаємозв'язків між об'ємами обчислювальних операцій і операцій обміну даними слід проводити з урахуванням отриманих фрагментних процесів. З рису. 2.12 випливає, що процес F1 є набором пар рівнянь, яким належав фрагмент $M_{F1} = \frac{l_1+l_2+l_7+l_8}{\Delta\varepsilon}$.

Елементи вузлу послідовно розв’язуються. При цьому кількість вхідних операцій дорівнює 7 (Q31, Q41, PU3, Q51, Q61, Q5M5, Q6M6), а кількість вихідних операцій – 4 (PU1, PU2, PU4, Q71). З цього можна визначити, що для кожних 8 обчислювальних операцій MF1 для обчислення правих частин рівнянь типу

$$\begin{cases} \dot{Q}_{jv} = a_j\left(P_{jv} - P_{j,v+1}\right) - \beta_j Q_{jv}\left|Q_{jv}\right| \\ \dot{P}_{j,v+1} = g_j\left(Q_{jv} - Q_{j,v+1}\right) \end{cases}$$

і для кожних $16M_{F1}$ таких операцій відповідно до рівнянь типу

$$\begin{cases} \dfrac{dg_{1qs}^{(v)}}{dt} = \alpha_{1qs}\left(gP_{1qs}^{(v)} - gP_{1qs}^{(v+1)}\right) - \beta_{1qs}^{L} g_{1qs}^{(v)} - \beta_{1qs}^{T}\left|g_{1qs}^{(v)}\right| g_{1qs}^{(v)} \\ \dfrac{dP_{1qs}^{(v+1)}}{dt} = \gamma_{1qs}(g_{1qs}^{(v)} - g_{1qs}^{(v+1)}) \end{cases}$$

$$\begin{cases} \dfrac{dg_{2qs}^{(v)}}{dt} = \alpha_{2qs}\left(gP_{2qs}^{(v)} - gP_{2qs}^{(v+1)}\right) - \beta_{2qs}^{L} g_{2qs}^{(v)} - \beta_{2qs}^{T}\left|g_{2qs}^{(v)}\right| g_{2qs}^{(v)} \\ \dfrac{dP_{2qs}^{(v+1)}}{dt} = \gamma_{2qs}(g_{2qs}^{(v)} - g_{2qs}^{(v+1)}) \end{cases}$$

тобто коефіцієнт набуде наступного діапазону значень

$$\frac{8M_{F1}}{11} \le N_{RO}^{F_1}/N_{DA}^{F_1} \le \frac{16M_{F1}}{11}.$$

Для процесів F2 та F3 ми отримуємо відповідно:

$$\frac{8M_{F2}}{5} \le N_{RO}^{F_2}/N_{DA}^{F_2} \le \frac{16M_{F2}}{5}$$

$$\frac{8M_{F2}}{6} \le N_{RO}^{F_3}/N_{DA}^{F_3} \le \frac{16M_{F3}}{6}$$

Ці відносини показують, що на РП–4 частина операцій обміну даними в процесах, що відповідають фрагментам мережі, значно менша порівняно з

рівнями РП–1, РП–2, РП–3. При застосуванні процесів БОМ вищезазначені відносини будуть більш вигідними.

Час циклу розв'язання паралельної реалізації процесів для всього МДОРП визначається найтривалішим процесом Fmax = F, тому отримуємо віртуальне прискорення.

$$S_{v4} = \frac{T(1)}{T_F}. \qquad (2.58)$$

де $T(1) = \sum M_j T_{QPj}$ для j=1,2,...,m.

Час виконання операцій розв'язання QP (T_{QPj}) залежить від типу j–ребра або пари рівнянь. Час виконання процесу F складається з часу обчислення та часу передачі даних (DA). $T_F = \sum M_i T_{QPi} + T_{FDA}$ для i=1, 2, ..., m_F. При цьому mF – це кількість ребер у максимальному фрагменті мережі F. Формулу (2.58) можна подати у більш інформативній формі.

$$S_{v4} = \left[\sum M_j T_{QPj} / \sum M_i T_{QPi}\right] * 1/\left[1 + T_{FDA} / \sum M_i T_{QPi}\right]. \qquad (2.59)$$

З'єднання між процесами F згідно схеми на рис. 2.12 виконуються за допомогою тих самих комутаційних елементів 2*2, що і на рівнях РП–2, РП–3.

2.6 Визначення підсистеми та її основні функції

Підсистема РПМС – це частина апаратних ресурсів, системного програмного забезпечення, програмних засобів моделювання, яка в дружньому до користувача стилі реалізує певну групу споріднених за змістом функцій середовища на всіх етапах паралельного моделювання складних динамічних систем з зосередженими й розподіленими параметрами [9].

Підсистема обміну інформацією ПОРПМС має виконувати наступні основні функції:

– повний функціональний список учасників СРПМ та наявних ресурсів СРПМ, які мають проводити обмін даними;

– ієрархічне представлення потоків даних для вибраного ZRS;

– тестова система для визначення реальних параметрів обміну операціями в мережі зв’язку ZRS;

– оптимізація паралельних програм щодо операцій обміну даними;

– зіставлення результатів апріорного аналізу та отриманих в ZRS ефектів операцій передачі даних з ефективністю підходів паралелізації;

– інтеграція з іншими підсистемами;

– методичні матеріали з паралельного програмування за допомогою MPI та OpenMP.

2.7. Висновки

1. Підсистема обміну інформацією має забезпечити інформаційну підтримку користувачів РПМС на всіх етапах розробки паралельних симуляторів і проведення модельних експериментів.

2. З огляду на проблемну орієнтацію РПМС і застосування перспективного блокового обчислювального методу (БОМ) визначено можливі рівні розпаралелювання (РП), дискретні форми Simulation–моделей та схеми обміну інформацією на кожному рівні.

3. Сформульовано визначення та основні функції підсистеми

3 РОЗРОБКА КІЛЬЦЕВОГО ТЕСТЕРУ ДИНАМІКИ ОБМІНУ МІЖ ДВОМА ПРОЦЕСОРАМИ

3.1 Постановка задачі

У паралельному моделюванні та симуляції динаміки складних технічних систем у реальному часі важливу роль відіграють можливості реального часу розроблених симуляторів, які реалізуються в рамках розподіленого паралельного середовища моделювання на цільових обчислювальних системах. Такі системи мають динамічні характеристики, які залежать від параметрів процесорів, наявних мереж зв’язку та програмних компонентів, що відповідають за обмін даними. Пропонується надати розробнику симулятора тест, за допомогою якого він може оцінити швидкість обміну даними між компонентами.

3.2 Simulation–модель і структура тестера, схема обміну даними між процесорами

Тест працює як генератор наступної синусоїдальної функції [1]

$$y = A \cdot sin(\omega t + \varphi), \tag{3.1}$$

де A – амплітуда сигналу;

ω – кругова частота;

φ – фаза сигналу;

t – незалежна змінна часу.

Генератор є рішенням диференціального рівняння

$$\frac{d^2x}{dt^2} = -\omega^2 x, \tag{3.2}$$

з початковими умовами

$$x(0) = Asin(\varphi), x = A\omega xos(\varphi), \tag{3.3}$$

Розробимо x–генератор як рішення (3.2) на двохпроцесорній системі вказаного паралельного ZRS. Для балансування навантаження процесорів ми позначимо

$$\frac{dx}{dt} = \omega y,$$

Тоді отримаємо $\frac{d^2x}{dt^2} = \omega\frac{dy}{dt}$ і рівняння (3.1) може бути представлене у вигляді системи рівнянь

$$\begin{cases} \frac{dx}{dt} = \omega y \\ \frac{dy}{dt} = -\omega x \end{cases}, \tag{3.4}$$

з початковими умовами

$$x(0) = Asin(\varphi), y = Axos(\varphi), \tag{3.5}$$

3.3 Алгоритми вирішувачів рівнянь тестера, обчислювальні методи

Для тестування ми чисельно розв’яжемо систему рівнянь (3.4) за допомогою двопроцесорної системи. При цьому кожен процесор буде інтегрувати рівняння з ідентичними правими частинами. Таким чином, досягнута однакова навантаженість процесорів. Рішення системи рівнянь (3.4)

складається з двох процесорів та компонентів віртуальної мережі, що утворюють закритий віртуальний кільцевий канал. Кожен процесор реалізує процес, який виконує наступні операції: введення правих частин рівнянь (3.4) з віртуальної мережі, інтегрування введених змінних з початковими умовами (3.5), множення результатів інтегрування, виведення обчислених змінних у віртуальну мережу.

Дискретна модель симуляції (DSM) для (3.4), (3.5) за методом Ейлера (E) для i = 0,1,2 ... складається з формул:

$$\begin{cases} x(i+1) = x(i) + h\omega y(i) \\ y(i+1) = y(i) + h[-\omega x(i)] \end{cases}, \tag{3.6}$$

де $x(0) = Asin(\varphi), y = Axos(\varphi)$.

За методом Адамса–Бешфорта (AB) DSM виглядає наступним чином:

$$\begin{cases} x(0) = Asin(\varphi) \\ x(1) = x(0) + h\omega y(0), i = 1 \\ x(i) = x(i-1) + \frac{h}{2}[3\omega y(i-1) - \omega y(i-2)], i > 1 \end{cases}, \tag{3.7}$$

$$\begin{cases} y(0) = Acos(\varphi) \\ y(1) = y(0) + h(-\omega x(0)), i = 1 \\ y(i) = y(i-1) + \frac{h}{2}[3(-\omega x(i-1)) - (-\omega x(i-2))], i > 1 \end{cases}, \tag{3.8}$$

У DSM за методом Рунге–Кутта (RK) передбачаються наступні обчислення:

$$x(i+1) = x(i) + \frac{1}{6}K_1^x + \frac{1}{3}\left(1 - \sqrt{\frac{1}{2}}\right)K_2^x + \frac{1}{3}\left(1 + \sqrt{\frac{1}{2}}\right)K_3^x + \frac{1}{6}K_4^x, \tag{3.9}$$

$$y(i+1) = y(i) + \frac{1}{6}K_1^y + \frac{1}{3}\left(1 - \sqrt{\frac{1}{2}}\right)K_2^y + \frac{1}{3}\left(1 + \sqrt{\frac{1}{2}}\right)K_3^y + \frac{1}{6}K_4^y, \tag{3.10}$$

Коефіцієнти K_j обчислюються згідно з формулами (3.11), (3.12) для j=1,2,…5:

$$\begin{cases} K_1^x = h[\omega y(i)] \\ K_2^x = h[\omega y(i) + K_1^x/2] \\ K_3^x = h\left[\omega y(i) - \left(1/2 - \sqrt{1/2}\right) K_1^x/2 + \left(1 - \sqrt{1/2}\right) K_2^x\right] \\ K_4^x = h\left[\omega y(i) - \sqrt{1/2}\, K_2^x + \left(1 + \sqrt{1/2}\right) K_3^x\right], \end{cases} \tag{3.11}$$

$$\begin{cases} K_1^y = h[\omega x(i)] \\ K_2^y = h[\omega x(i) + K_1^y/2] \\ K_3^y = h\left[\omega x(i) - \left(1/2 - \sqrt{1/2}\right) K_1^y/2 + \left(1 - \sqrt{1/2}\right) K_2^y\right] \\ K_4^y = h\left[\omega x(i) - \sqrt{1/2}\, K_2^y + \left(1 + \sqrt{1/2}\right) K_3^y\right]. \end{cases} \tag{3.12}$$

Функцію $x = A \cdot sin(\omega t + \varphi)$ дискретизують з кроком часу T, причому T залежить від тривалості операцій в обох процесорах.

$$T = 2(T_{INT} + T_{DA} + T_{\omega})$$

Тут T_{INT}, T_{DA}, T_{ω}– тривалість інтегрування, операцій обміну даними та множення на h. Це стосується ступеневої апроксимації (рис. 3.1)

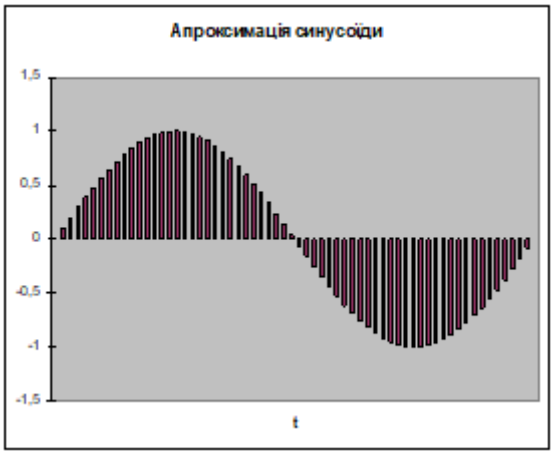


Рисунок 3.1 – Апроксимація синусоїди

Похибка апроксимації становить $\Delta x(t) = x(t) - x(nT)$, де дискретний час представлений як t=nT, n=0, 1, 2, …. Аналіз показує, що максимальна похибка виникає на першому кроці часу n=1, тобто

$$\Delta x_{max} = x(T) - x(0) = Asin(\omega T) \quad (3.13)$$

З припущенням, що для малих аргументів sin(Arg) = Arg, використовуємо для подальших оцінок формулу:

$$\Delta x_{max} = A\omega T \quad (3.14)$$

Відносна похибка в відсотках при x = A складає:

$$\delta x = \frac{\Delta x}{x} * 100\% = \omega T * 100\% \quad (3.15)$$

Таким чином, для генерації часової функції синуса з відносною похибкою δx% при заданій частоті f, час дискретизації від двопроцесорного рішення системи рівнянь (3.1) повинен бути

$$T = \frac{0{,}01\delta x}{\omega} \quad (3.16)$$

3.4 Програмна реалізація

В рамках роботи було реалізовано 2 програми тестера для перевірки можливостей системи. Перший тестер використовує технологію MPI для обміну даними між процесорами.

Message Passing Interface (MPI) – це стандартна специфікація інтерфейсу передачі повідомлень для паралельних обчислень в системах з розподіленою пам’яттю. MPI не є мовою програмування. Це бібліотека функцій, які програмісти можуть викликати з коду на мовах C, C++ або Fortran для написання паралельних програм. За допомогою MPI можна динамічно створювати MPI комунікатор і мати одночасно працюючі на окремих вузлах кластера множини процесів. Кожен процес має унікальний ранг MPI для

ідентифікації його, власний простір пам’яті та виконується незалежно від інших процесів. Процеси взаємодіють між собою, передаючи повідомлення для обміну даними. Паралелізм виникає, коли завдання програми розбивається на невеликі частини та розподіляється серед процесів, кожен з яких обробляє свою частину [10].

Використовуючи метод з п. 3.3 було створено програмний додаток який генерує функцію синусу з заданими параметрами. Алгоритмічно така програма виконує дії які видно на рис. 3.2.

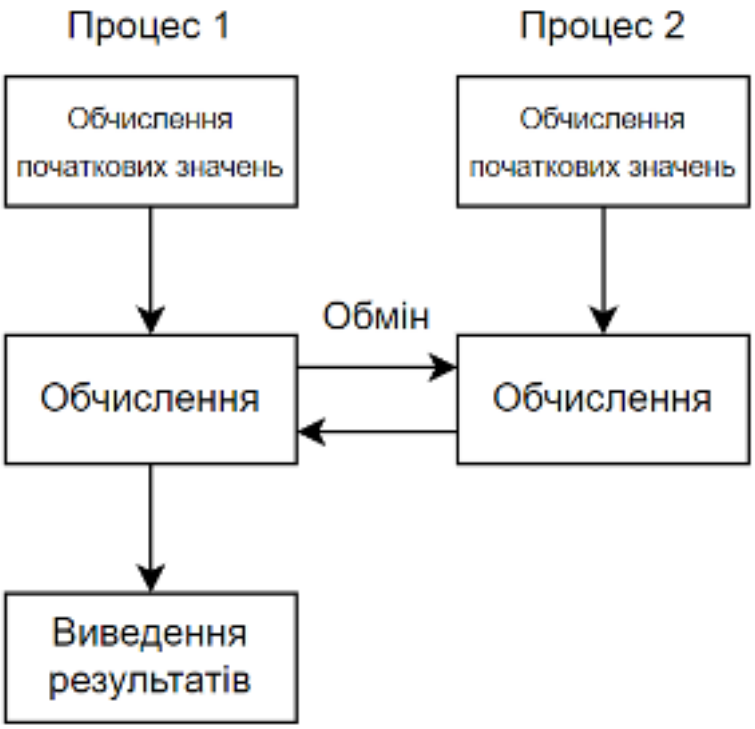


Рисунок 3.2 – Алгоритм тестера MPI

Програмну реалізацію алгоритму нам мові програмування С можна побачити на рис. 3.3.

Рисунок 3.3 – Реалізація розрахунку для MPI тестера

На рисунку 3.3 видно розбиття розв’язання на 2 етапи: перший – отримання початкових даних перед обчисленням та другий – обчислення, використовуючи обмін між процесами за допомогою MPI. В даній частині використовуються функції MPI_Recv та MPI_Send для обміну повідомленнями між процесами у паралельному середовищі. MPI_Recv отримує повідомлення, а MPI_Send відправляє його до іншого процесу [11, 12]. Перший процес починає обмін з відправки повідомлення, а відповідний йому інший процес починає з отримання повідомлення, тому аргументи в першому процесі відрізняються від другого процесу.

Аргументи MPI_Recv:

– &xPrev – адреса змінної, в яку буде записане отримане повідомлення.

– 1 – кількість елементів у буфері;

– MPI_DOUBLE – тип елементів у буфері (в даному випадку, тип double);

– 0 – ранг процесу, від якого отримується повідомлення (у випадку першого процесу – змінна to_rank);

– 1 – ідентифікатор повідомлення;

– MPI_COMM_WORLD – спільний комунікатор для всіх процесів;

– &status – інформація про стан виконання команди MPI_Recv.

Аргументи MPI_Send:

– &yPrev – адреса змінної, яку буде відправлено як повідомлення;

– 1 – кількість елементів у буфері;

– MPI_DOUBLE – тип елементів у буфері (в даному випадку, тип double);

– 0 – ранг процесу, якому відправляється повідомлення (у випадку першого процесу – змінна to_rank);

– 4 – ідентифікатор повідомлення;

– MPI_COMM_WORLD – спільний комунікатор для всіх процесів.

Також даний тестер дозволяє перевіряти зв’язки між різними вузлами системи. Для цього потрібно виконати запуск програми використовуючи більше ніж 2 процеси. При цьому буде виконано послідовні тести між першим та кожним наступним процесом. Це дозволяє перевірити можливості системи обміну в обраній системі. Код програми який дозволяє виконувати таке тестування можна побачити на рис. 3.4.

```
if(my_rank==0){
    for(int i=1; i<size; i++){
        MPI_Barrier(MPI_COMM_WORLD);
        startTime = MPI_Wtime();
        exchange(my_rank, i);
        endTime = MPI_Wtime();
        deltaTime = endTime - startTime;
        printf("0:%d-%f;\n", i, deltaTime);
    }
}
else{
    for(int i=1; i<size; i++){
        MPI_Barrier(MPI_COMM_WORLD);
        if(my_rank==i){
            exchange(my_rank, i);
        }
    }
}
```

Рисунок 3.4 – Код для тестування декількох процесорів

Дана частина коду має в собі декілька функцій MPI, опишемо їх докладніше.

MPI_Barrier використовується для синхронізації всіх процесів у комунікаторі, тобто кожен процес чекає, поки всі інші досягнуть цієї точки виконання. В тестері використовується для того щоб кожен процес дочекався своєї черги для обміну з першим процесом. Має спільний аргумент – MPI_COMM_WORLD [13].

MPI_Wtime використовується для отримання поточного часу в секундах. Ця функція може бути використана для вимірювання часу виконання певної частини програми. Не має аргументів [14].

Повна програмна реалізація тестера який використовує технологію MPI. доступна в додатку Б.

Також було створено тестер для систем зі спільною пам’яттю використовуючи можливості OpenMP. OpenMP, що означає Open Multi–Processing – це API (інтерфейс програмування додатків), яке підтримує багатоплатформенне програмування обміну даними в спільній пам’яті для багатоядерного виконання в мовах програмування C, C++ та Fortran. Він призначений для сприяння паралельному програмуванню, надаючи набір директив компілятора, бібліотечних процедур та змінних середовища [15].

Через обмеження OpenMP з приводу ручного керування процесами, не кожний математичний метод можливо виконувати паралельно [16]. З даного приводу було реалізовано розділення обчислень на 2 частини. 1–й процес обчислює першу половину ряду даних, а 2–й процес – другу половину. Для такі реалізації потрібно проводити підготовку даних перед обчисленнями як спільних між потоками так і даних в кожному потоці. Також необхідно збирати данні разом після виконання обчислень. Алгоритм роботи такої програми наведено на рис. 3.5.

Рисунок 3.5 – Алгоритм тестеру використовуючи OpenMP

Програмна реалізація тестеру на мові програмування С можна побачити на рис. 3.6, 3.7.

```
double start = omp_get_wtime();
for(;;){
    if(tSim>tMax){
        break;
    }
    count++;
    tSim+=h;
}
tSim-h;
for(int i=0;i<count;i++){
    tSim+-h;
    if(i>-count/2){
        tSim2-tSim;
        break;
    }
}
tSim=h;
```

Рисунок 3.6 – Обчислення спільних початкових значень

Рисунок 3.7 – Обчислення початкових значень та значень функції

3.5 Експериментальні дослідження

Проведемо експериментальні дослідження, спрямовані на вивчення продуктивності розробленого проблемно орієнтованого паралельного моделюючого середовища.

Визначимо параметри експерименту. Тести програм відбувались на двопроцесорній системі. Змінювався параметр tMax для ускладнення задачі обчислення. Виконувались заміри часу необхідного на виконання обчислень (Tda) в секундах. Заміри відбувались 5 разів на кожний рівень складності і записувався середній результат. Виконувались тестування для програм з MPI, OpenMP, та послідовної версії програми. Результати експерименту занесені в таблиці 3.1–3.3.

Таблиця 3.1 – результати експерименту програми з MPI.

tMax	10	100	1000	5000	10000	15000	25000	40000
Tda	0,031	0,31	3,1	16,1	32,2	47,8	79	128,5

Таблиця 3.2 – результати експерименту програми з OpenMP.

tMax	10	100	1000	5000	10000	15000	25000	40000
Tda	0,0016	0,017	0,22	1,08	2,2	3,37	5,67	9,16

Таблиця 3.3 – результати експерименту послідовної програми.

tMax	10	100	1000	5000	10000	15000	25000	40000
Tda	0,0029	0,032	0,32	1,6	3,22	5,35	8,24	13,21

3.6 Висновки

З результатів експерименту можна побачити, що з ростом складності задачі моделювання збільшується й час витрачений на її розв’язання. Це є закономірним для усіх трьох варіантів програм (табл. 3.1–3.3).

У таблиці 3.1 можна побачити що зі збільшення часу виконання (Tda) при збільшенні tMax свідчить про те, що програма з MPI має високу витрату часу при великих обчислювальних завданнях. Час виконання зростає лінійно, що може бути пов’язано зі збільшенням обсягу обчислень.

Таблиця 3.2 OpenMP показує значно кращі результати порівняно з MPI при подібних обчислювальних завданнях. Час виконання від складності обчислень також зростає лінійно, при чому набагато менше ніж у програмі з MPI.

Таблиця 3.3 показує результати виконання послідовних обчислень. Вони є швидшими ніж програма з MPI але повільніше ніж програма OpenMP. Зростання також відбувається лінійно.

З цього можна зробити висновки що на системі, на якій проводилось тестування витрачає багато часу на передачу даних між процесами, адже програма з MPI виконувалась довше ніж програма з OpenMP, яка не має в собі передачу даних між процесами.

Отже реалізовано тестер, який дозволяє користувачам визначити динамічні властивості системної складової РПМС «процесор–канал зв’язку–процесор».

4 АЛГОРИТМИ ФУНКЦІОНУВАННЯ ПІДСИСТЕМИ ОБМІНУ ІНФОРМАЦІЄЮ

4.1 Презентація підсистеми

4.1.1 Апаратні ресурси MIMD–кластера ДонНТУ

Обчислювальні ресурси активно використовуються під час виконання обчислень і організації роботи системи (у тому числі адміністрування та обслуговування). Серед таких ресурсів можна виділити серверне обладнання, таке як центральні процесори, оперативна пам’ять та сховище даних.

В наяності надпродуктивного MIMD–кластера ДонНТУ є наступне обладнання:

– 40 обчислювальних вузлів від Supermicro. Кожен із цих серверів складається з двох обчислювальних вузлів, оснащених процесорами Intel® Xeon® X5560 (4 ядра, 8 потоків на кожен процесор). Кожен процесор має підключену до нього регістровану оперативну пам’ять обсягом до 24 ГБ.

– 10 адміністративних вузлів Dell PowerEdge R710. Усі сервери Dell R710 обладнані по два процесори Intel® Xeon® X5560 (4 ядра, 8 потоків на кожен процесор) на чипсеті Intel® 5520. Кожен із цих серверів оснащений регістрованою оперативною пам’яттю об’ємом до 48 ГБ. З них 6 серверів мають два диски по 140 ГБ в апаратному RAID1 для забезпечення надійності системи та 4 сервери із збільшеною кількістю дискових ресурсів (два диски по 140 ГБ в апаратному RAID1 та додатково 4 диски по 1 ТБ).

Обмін інформацією між компонентами системи відбувається за допомогою мережевих ресурсів. До цих ресурсів відноситься мережеве обладнання, таке як комутатори, маршрутизатори та з’єднувальні елементи:

– Комутатор Dell DGS–1210–28/ME оснащений 24 портами Ethernet і 4 портами SFP, з загальною пропускною здатністю 1 Гб/с.

– Два комутатори Dell PowerConnect 5448 у системі мають по 48 портів Ethernet і 4 порти SFP, які працюють за стандартом 1 Гб/с.

– Маршрутизатор Mikrotik RB2011UiAS–2HnD–IN має 5 портів Ethernet зі швидкістю 100 Мб/с, 5 портів Ethernet зі швидкістю 1 Гб/с та 1 SFP порт зі швидкістю 1 Гб/с.

– Комутатор InfiniBand ISR 2004 має 96 портів, які використовують підключення CX4 і мають пропускну здатність 20 Гб/с кожен.

Усе обладнання кластеру об’єднано у високопродуктивну багаторівневу мережу, яка дозволяє вузлам системи ефективно обмінюватись інформацією між собою [17].

4.1.2 Програмні ресурси MPI та OpenMP

Окрім згаданих в розділі 3.4 функцій MPI також варто приділити увагу типам даних, які використовуються при обміні інформацією. У таблиці 4.1 наведено всі доступні типи даних стандарту MPI та їх позначення при використанні в мові програмування C.

Таблиця 4.1 – Типи даних стандарту MPI

Типи даних MPI	Тип даних C
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

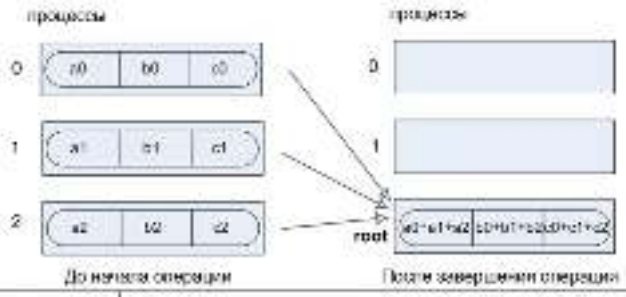
Для організації обміну даними важливу роль відіграє операція редукції, зміст та синтаксис якої мають вигляд (рис.4.1):

```
int MPI_Reduce(void *sendbuf, void *recvbuf, int
count, MPI_Datatype type, MPI_Op op, int root,
MPI_Comm comm)
```

sendbuf	буфер пам'яті з повідомленням, що відправляється
recvbuf	буфер пам'яті для результуючого повідомлення (тільки для процесу з рангом root)
count	Кількість елементів в повідомленнях
type	тип елементів повідомлень
op	операція, яка має бути виконана над даними
root	ранг процесу, на якому має бути отриманий результат
comm	комунікатор, в рамках якого виконується операція

Рисунок 4.1 – MPI_Reduce

Типи операцій, які можуть бути виконані в MPI_Reduce наведені на рис. 4.2. Вони дозволяють ефективно використовувати обчислювальні або аналітичні ресурси під час обміну, а не після нього. Через це збільшується швидкість програми та зменшується кількість необхідного коду.



Операція	Опис
MPI_MAX	Визначення максимальної величини
MPI_MIN	Визначення мінімальної величини
MPI_SUM	Визначення суми величин
MPI_PROD	Визначення результату множення величин
MPI_LAND	Виконання логічної операції «І» над величинами
MPI_BAND	Виконання бітової операції «І» над величинами повідомлень
MPI_LOR	Виконання логічної операції «АБО» над величинами
MPI_BOR	Виконання бітової операції «АБО» над величинами
MPI_LXOR	Виконання логічної операції виключаючого «АБО» над величинами повідомлень
MPI_MAXLOC	Визначення максимальних величин та їх індексів

Рисунок 4.2 – Операції при редукції

Наведемо короткі відомості про OpenMP для кращого розуміння принципу роботи зі стандартом [18].

1. Визначення

1.1. OpenMP (Open Multi–Processing) – це розширення мови для паралельного програмування в C/C++ і фортрані. Ядром є спеціальні команди компілятора, що звуться також директивами або Pragma, які вводяться в текст програми. Тим самим маркуються частини програми, які повинні виконуватись паралельно (на системах з загальною пам’яттю.)

1.2. OpenMP – це специфікація директив компілятора, бібліотечних функцій і змінних середовища, яку розроблено групою провідних SW/HW–постачальників з метою стандартизації програмування на MIMD–системах з загальним адресним простором (Shared Memory). OpenMP розширює послідовні мови програмування C/C++ і фортран за рахунок конструктивів для SPMD–програмування, розподілу роботи, синхронізації та декларування спільних (shared) і приватних (private) змінних.

2. Зміст специфікації

2.1. Потоки

OpenMP дозволяє використовувати паралельні потоки для розпаралелювання виконання коду. Керування кількістю потоків може бути визначено динамічно або статично.

2.2. Механізми синхронізації

OpenMP надає різноманітні механізми синхронізації, такі як директиви *omp critical*, *omp barrier*, *omp atomic* та інші. Ці механізми дозволяють управляти доступом до критичних розділів коду та забезпечувати коректну взаємодію паралельних ниток.

2.3. Паралельність задач

OpenMP підтримує концепцію задач, що дозволяє програмісту виражати паралельні обчислення як невеликі задачі, які можуть бути виконані асинхронно. Директиви, такі як *omp task* та *omp taskwait*, використовуються для створення та синхронізації задач.

2.4. Паралельність даних

OpenMP надає можливості для паралельної обробки даних, такі як цикли з директивою *omp parallel for*. Це дозволяє однаковим частинам даних оброблятися паралельно різними потоками.

2.5. Модель пам’яті

OpenMP використовує спільну модель пам’яті, що дозволяє потокам ділити дані. Проте, програміст повинен бути обережним щодо можливості конфліктів при зверненні до спільних ресурсів та використання синхронізації.

4.2 Організація обміну даними в паралельному симуляторі мережевого динамічного об’єкту з зосередженими параметрами (МДОЗП)

Мова йде про розробку MIMD–симулятора мережевого аеродинамічного об’єкту з зосередженими параметрами (МДОЗП), приклад графу якого наведено на рис. 4.3 [1,8,9].

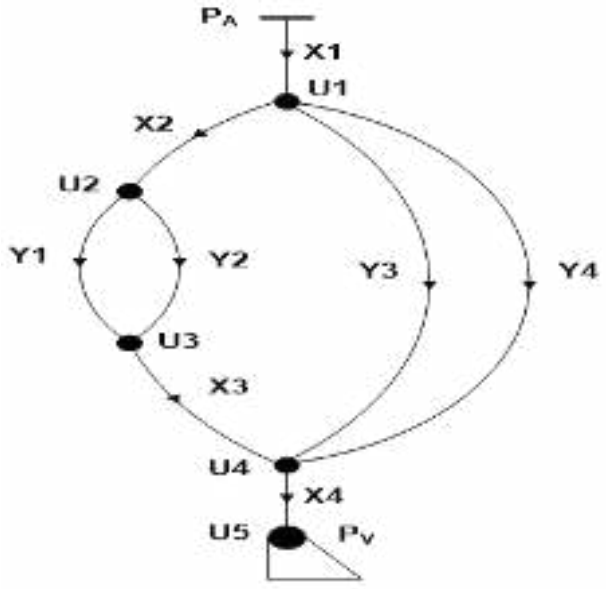


Рисунок 4.3 – Граф МДОЗП (m= 8 гілок Q1…Q8, n=5 вузлів U1…U5), дерево X1…X4 і антидерево Y1…Y4.

Структуруємо вектор потоків $Q = (Q_1, Q_2,...,Q_m)^T$ відповідно до гілок дерева й антидерева:

$Q = (X,Y)^T$,

$X = (X_1, X_2,...,X_{n-1})^T$ – вектор потоків в гілках дерева,

$Y = (Y_1,Y_2,...,Y\gamma)^T$ – вектор потоків в гілках антидерева.

Кожна гілка об'єкту описується рівнянням перехідного процесу[1]

$$K*dQ/dt + R*Q*|Q| = H, \quad (4.1)$$

де Q – потік повітря $м^3/с$;

R – аеродинамічний опір;

H – різниця тиску в початковому і кінцевому вузлах гілки $H_{ньютон}/ м^2$;

K – коефіцієнт інерційності потоку повітря.

Модель МДОЗП складається з n–1 рівнянь для вузлів та γ= m–(n–1) рівнянь для контурів (m – кількість гілок, n – кількість вузлів).

Введемо поряд з векторами Q, X, Y діагональні матриці параметрів K, R та вектори депресій $H = (H_1,H_2,...,H_m)^T$ і нелінійних елементів $Z = (Z_1,Z_2,...,Z_m)^T$, $Z_i = Q_i*|Q_i|$ та структуруємо їх відносно векторів X, Y:

$$H = \left(H_{x1}H_{x2}H_{x3}H_{x4}H_{y1}H_{y2}H_{y3}H_{y4}\right)^T$$

$$Z = (X_1|X_1|\ X_2|X_2|\ X_3|X_3|\ X_4|X_4|\ Y_1|Y_1|\ Y_2|Y_2|\ Y_3|Y_3|\ Y_4|Y_4|)^T$$

Модель МДОЗП описується системою рівнянь (4.2) [1]

$$\begin{cases} A * Q = 0 \\ S * K\frac{dQ}{dt} + SRZ = SH' \end{cases} \quad (4.2)$$

Тут ***A(Ax Ay)*** – матриця інциденцій, ***S(Sx Sy)*** – матриця контурів. Модель (4.2) трансформується в Simulation–модель (4.3), (4.4) [1] :

$$X = -W * Y, \quad (4.3)$$

$$\frac{dY}{dt} = Hu - RuZ, \quad (4.4)$$

де $W = A_x^{-1} A_y$;

Hu = TP*H;

Ru = TP*R;

TP = V*S;

$V = \left(S_x K_x - S_y K_y W\right)^{-1}$.

Початкові умови: $Y(0)=Y_0$, $X(0) = -W * Y_0$.

На рис. 4.4 показано блок–схему С–програми чисельного розв'язання системи рівнянь (4.3), (4.4). Ця програма реалізує послідовний SISD–симулятор.

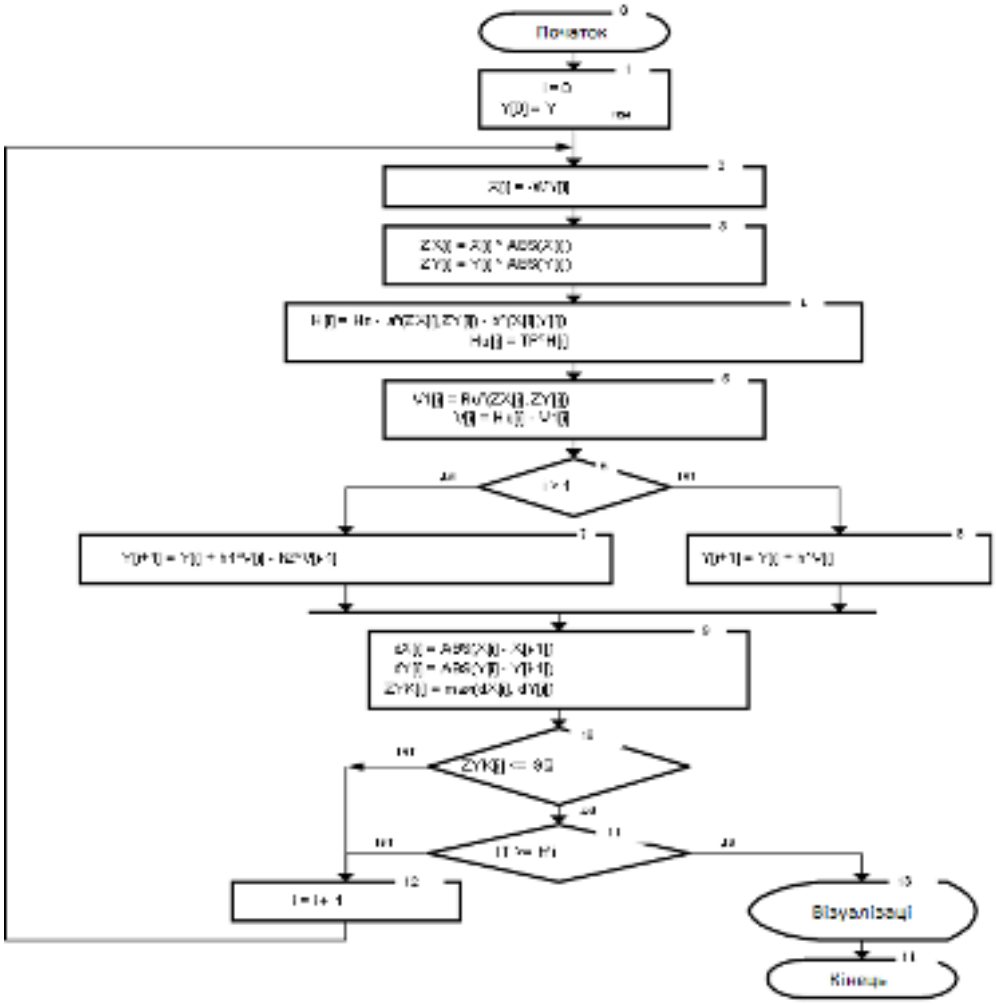


Рисунок 4.4 – Блок–схема С–програми (послідовний С–SISD–симулятор)

На рис. 4.5 показано блок–схему С–програми чисельного паралельного розв'язання системи рівнянь (4.3), (4.4) методом Адамса–Башфорта. Ця програма реалізує паралельний MIMD–симулятор. Передбачається γ процесів (процесорів), які паралельно обчислюють компоненти X1....Xn–1, Y1....Yγ векторів X, Y. Кожний процесор виконує програму рис. 4.2. Взаємодію процесорів виконує MPI–програма, текст якої наведено в додатку Б.

Визначено наступні векторно–матричні операції обчислень і обміну даними:

– Ініціалізація матриць:

```
void InitMatrix(char*filename, Matrix pMatrix)
```

– Ініціалізація векторів:

```
void InitVector(char*filename, Vector pVector)
```

– Елемент вектору як результат множення **i**–рядка матриці на вектор:

```
float VectorElement(Matrix pMatrix, Vector pVector, int rank)
```

Ці функції ініціалізації зчитують відповідні дані матриць і векторів з підготовлених файлів (або з топологічного аналізатора) та присвоюють ці дані елементам матриць і векторів.

Функція VectorElement обчислює i–елемент вектора (i – це номер процесу, в якому виконуються обчислення).

В головній частині програми декларуються змінні, що стосуються програми, а також виконується ініціалізація бібліотеки MPI. Для цього необхідні наступні функції:

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);
```

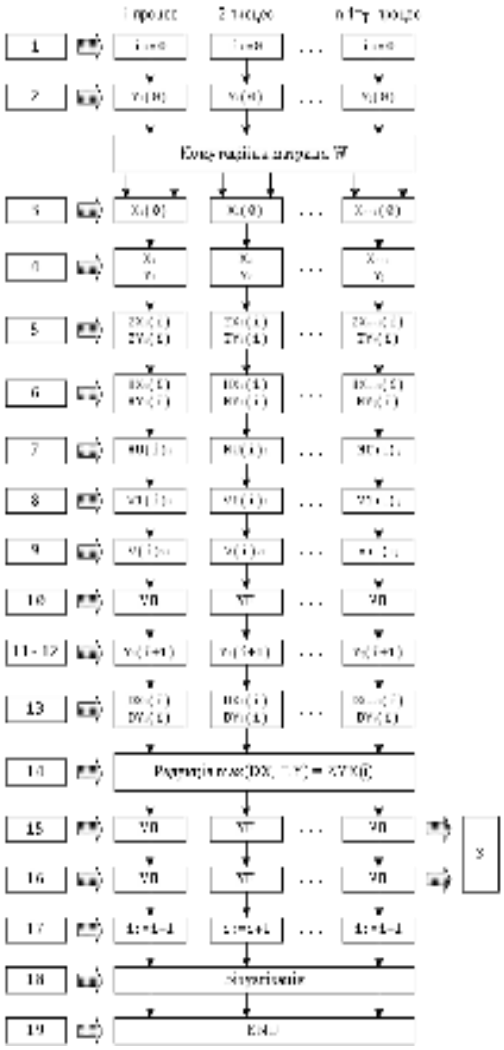


Рисунок 4.5 – Блок–схема паралельної MPI–програми (паралельний C–MIMD–симулятор)

За допомогою цих функцій надається кількість процесорів, що замовлена користувачем, а процесори отримують логічні імена (номери) від **0** до **γ–1**.

Процесор **0** ініціалізує дані початкових умов.

З огляду на структурування топологічних матриць **A(Ax Ay), S(Sx Sy),** матриць параметрів **Rdia(Rx Ry), Kdia(Kx Ky)** і векторів **H(Hx Hy),**

Z(Zx Zy) відносно векторів струмів повітря в гілках дерева **X** і антидерева **Y** названі матриці мають розмірності (n–1)*(n–1), (n–1)*(γ), γ*γ , а вектори – (n–1)*1, γ*1.

Під час ініціалізації процесором **0** всі інші процесори мають чекати. MPI–команда

```
MPI_Barrier(MPI_COMM_WORLD)
```

гарантує, що всі процесори зможуть стартувати тільки після завершення ініціалізації. Паралельний алгоритм передбачає, що матриці й вектори, які задіяні в обчисленнях поза **0**–процесором, відомі також всім іншим процесорам. Для пересилання даних із **0**–процесора (наприклад , матриці ***W***) всім іншим процесорам використовується MPI–команда

```
MPI_Bcast(W, M*N, MPI_FLOAT, 0, MPI_COMM_WORLD);
```

Для гарантування завершень всіх таких колективних операцій передачі даних до початку наступних сегментів програми застосовано в відповідних місцях програми команду синхронізації MPI_Barrier().

Щоб запам'ятати результати, кожен процесор відкриває масиви даних з іменами Xproc–„Prozessnummer", Yproc–„Prozessnummer" і записує туди свої дані.

Після завершення підготовчих операцій розпочинається інтегрування рівнянь. В кожному процесорі обчислюються компоненти вектора ***X*** (в програмі вони мають ім'я ***X***local) і векторів ***Z***x, ***Z***y (***Z***xlocal, ***Z***ylocal) . Потім в **0**–процесорі формуються вектори ***X***, ***Z***x, ***Z***y із обчислених компонентів. Ця операція виконується MPI–командою типу

```
MPI_Gather(&Xlocal, 1, MPI_FLOAT, X, 1, MPI_FLOAT,0, MPI_COMM_WORLD)
```

0–процесор надає в розпорядження всім процесорам–учасникам обчислень необхідні вектори ***X***, ***Z***x та ***Z***y за допомогою команди MPI_Bcast().

Далі кожен процесор обчислює праву частину диференціальних рівнянь (змінні ***V***local та ***VP***local) за методом Адамса–Башфорта. Інтегрування диференціальних рівнянь проводиться з кроком ***h*** до досягнення заданої тривалості моделювання ***T***model або стаціонарного стану МДОЗП. Для контролю стаціонарних станів слугує MPI–функція

```
MPI_Reduce(&DeltaY, &MaxDeltaY, 1, MPI_FLOAT, MPI_MAX, 0, MPI_COMM_WORLD),
```

яка визначає максимальну різницю Delta***Y*** (Delta***X***) обчислених ***Y, X*** на сусідніх кроках і запам'ятовує в **0**–процесорі для умовного переходу за виразом *max (DeltaX, DeltaY) <Delta*.

Вектор ***Y*** формується із обчислених в кожному процесорі компонент аналогічно формуванню векторів ***X***, ***Z***x, ***Z***y.

Після інтегрувальних обчислень початкові масиви даних закриваються і виконується заключна функція MPI_Finalize().

4.3 Організація обміну даними в паралельних симуляторах мережевого динамічного об'єкту з розподіленими параметрами (МДОРП)

4.3.1 Орієнтація на MIMD–системи з розподіленим адресним простором і MPI–засобами паралельного програмування

Паралельний симулятор МДОРП базується на чисельному розв'язанні множини системи рівнянь Simulation–моделі j–гілки

$$\begin{cases} \dot{Q}_{jv} = a_j\left(P_{jv} - P_{j,v+1}\right) - \beta_j Q_{jv}\left|Q_{jv}\right| \\ \dot{P}_{j,v+1} = g_j\left(Q_{jv} - Q_{j,v+1}\right) \end{cases}, \tag{4.3}$$

де індекс ***v*** змінюється в межах ***v*** = 1, 2, …, Mj. Розглянемо організацію обміну при реалізації паралельного симулятора на цільовій обчислювальній системі, структуру якої показано на рис. 4.6.

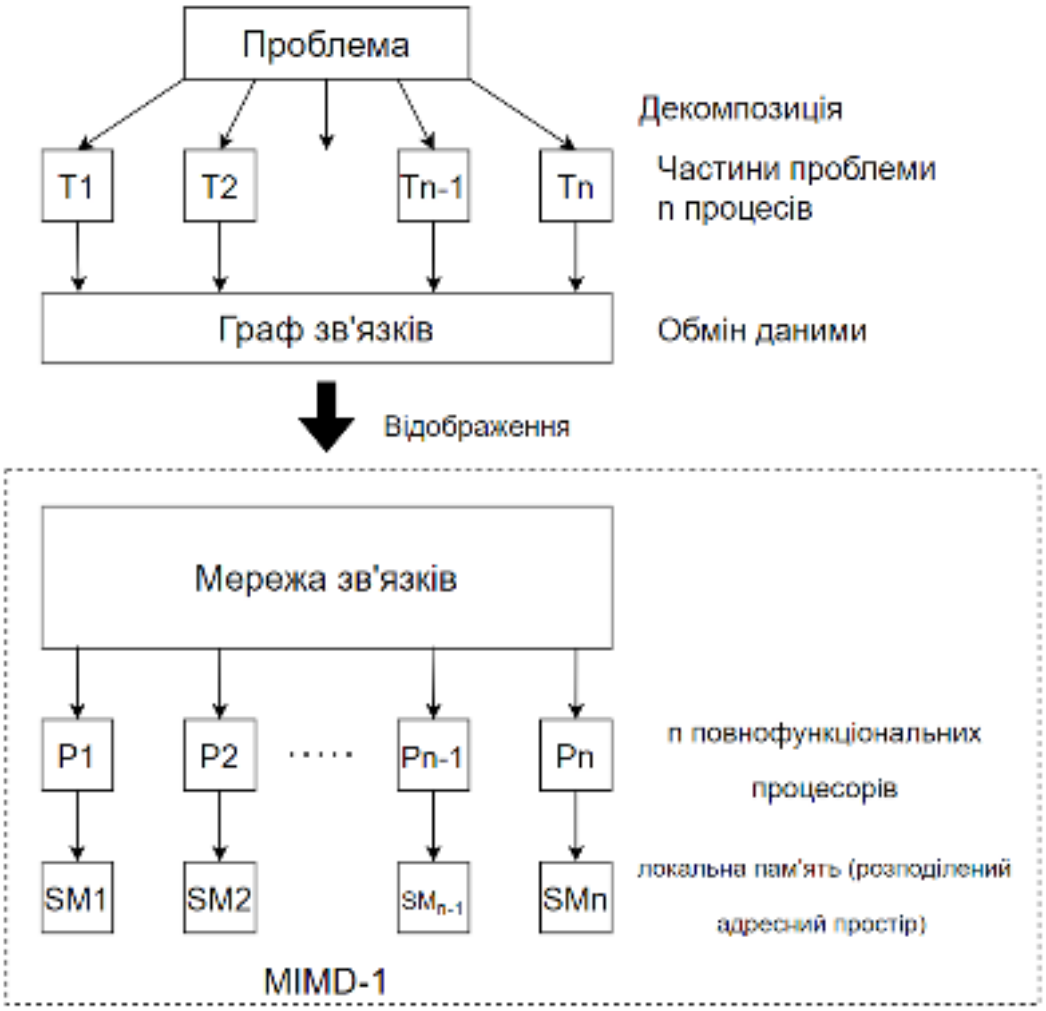


Рисунок 4.6 – Структура MIMD–системи з розподіленим адресним простором і MPI–засобами обміну інформацією

На всіх можливих рівнях розпаралелювання обчислення невідомих змінних *Qj(v, t)* і *Pj(v,t)* має виконуватись за наступною схемою. На нульовому етапі відбувається ініціалізація процесів, а потім виконується цикл тривалістю N, де N – кількість тактів (ітерацій) вирішення задачі і містить три етапи. Час вирішення рівняння для Qjv дорівнює сумі часу виконання 2 додавань, 3 множення і 1 операції знаходження абсолютного значення, а для Pj,k+1 – 1 додавання і 1 множення. На наступному етапі відбувається обчислення Qjv та Pj,v+1 заданим числовим методом. Час виконання цього етапу для методу Адамса–Башфорта другого порядку обчислюється за наступною формулою: t2 = 2tмн. + 2tдод. Після цього відбувається обмін даними між процесами і перехід на новий такт вирішення задачі (рис. 4.7).

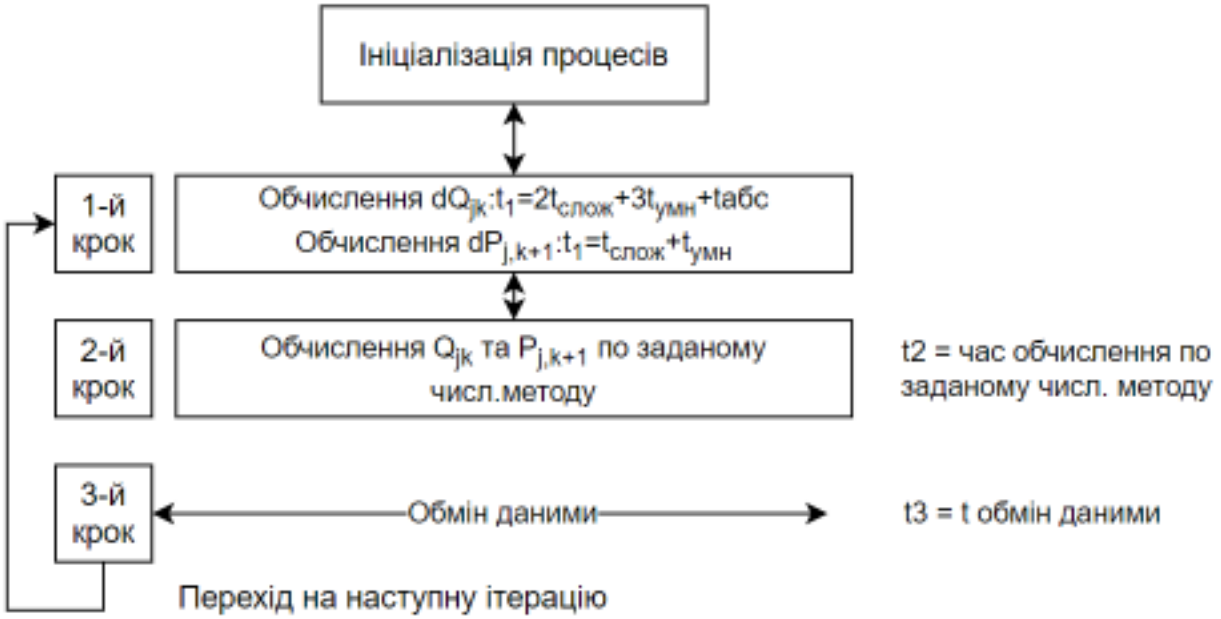


Рисунок 4.7 – Алгоритм обчислень Qj(v, t) і Pj(v,t)

На рис. 4.8 представлено блок–схему MIMD–симулятора для третього рівня розпаралелювання: кількість процесів (процесорів) дорівнює кількості гілок *m*, при цьому кожна гілка має *Mj* елементів. Процеси нумеруються від *0* до *m–1*, *i* – номер ітерації. На початковому етапі відбувається ініціалізація паралельної частини програми за допомогою функції MPI_Init, а також визначення загальної кількості паралельних процесів (MPI_Comm_size) і визначення номера процесу (MPI_Comm_rank). Потім відбувається паралельне обчислення правих частин систем рівнянь і формул обраного числового методу для гілки як об’єкта з розподіленими параметрами. Отримані значення збираються в нульовому процесі за допомогою функції колективної взаємодії процесів MPI_Gather. У отриманих матрицях тисків 0–й процес виконує заміщення внутрішніх граничних умов, а потім розподіляє ці дані всім іншим процесам (функція MPI_Bcast). Після цього програма переходить до виконання наступної ітерації (i := i + 1). Текст MPI–програми наведено в додатку Б.

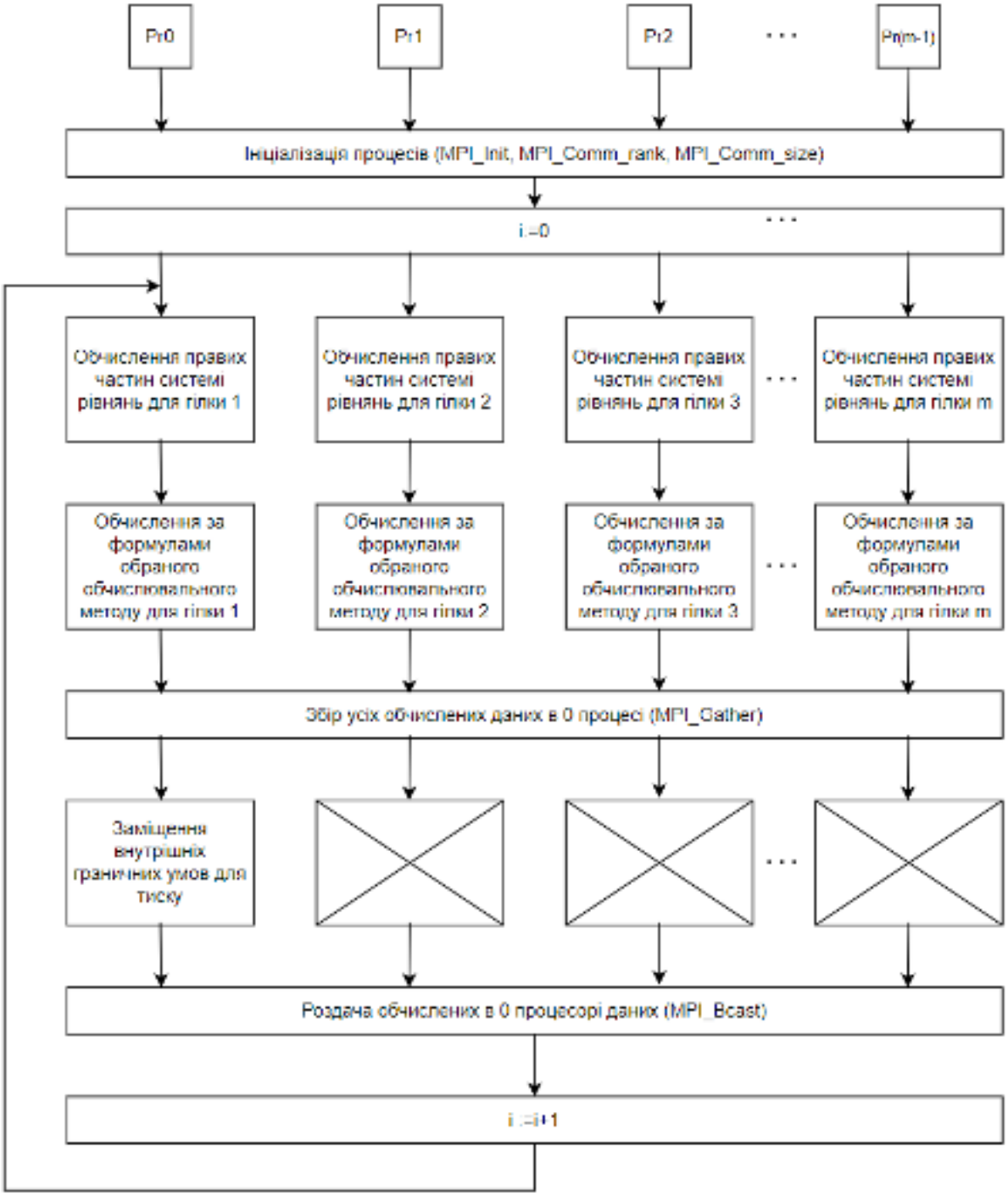


Рисунок 4.8 – Блок–схема MPI–програми функціонування паралельного MIMD–симулятора МДОРП.

4.3.2 Орієнтація на MIMD–системи з загальною пам’яттю і OpenMP–засобами паралельного програмування

Розглянемо організацію обміну даними при реалізації паралельного симулятора МДОРП на цільовій MIMD–системі, структура якої показана на рис. 4.9. Тут всі процесори працюють з загальною пам’яттю (Shared Memory). Обмін даними між процесорами і синхронізація їх паралельної роботи програмується засобами стандарту OpenMP [18].

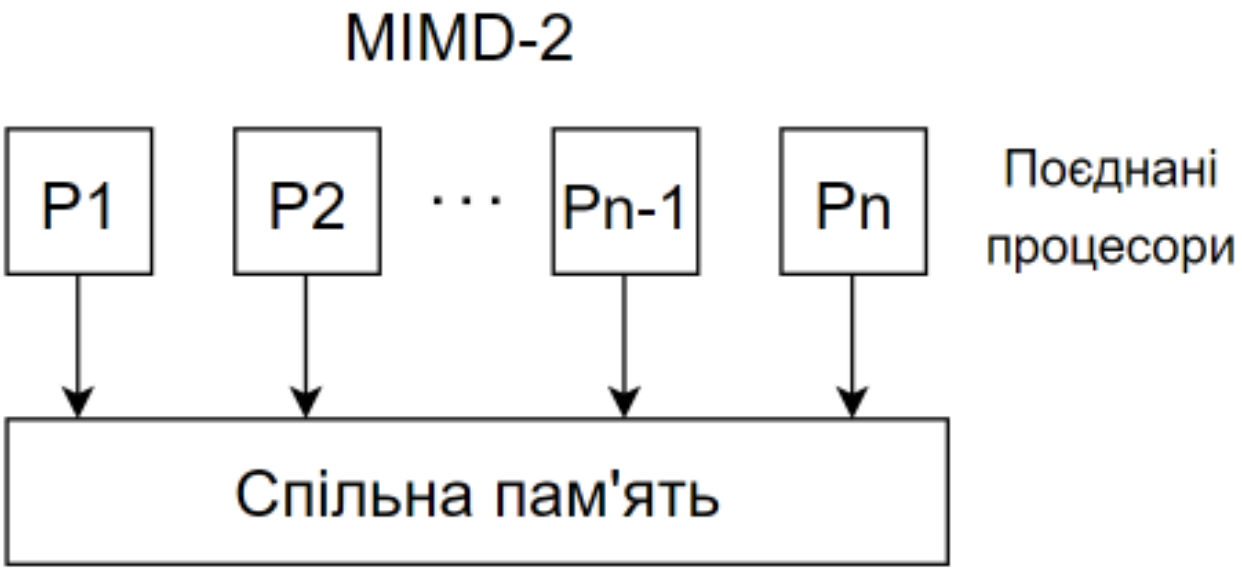


Рисунок 4.9 – Структура MIMD–системи з загальною пам’яттю

МДОРП як об’єкт паралельного моделювання з можливими чотирма рівнями розпаралелювання (розділ 2) потребує певної підготовки до реалізації його Simulation–моделі на цій архітектурі [1, 8, 9].

Пропонуються наступні системно–програмні операції з побудови OpenMP–симулятора. Для визначення кількості процесорів, що необхідні для обчислення кожної із гілок, необхідно розрахувати крок дискретизації за координатою простору Δx. Для ілюстрації цього на рис. 4.10 приведена тестова мережа із визначеною довжиною гілок у метрах.

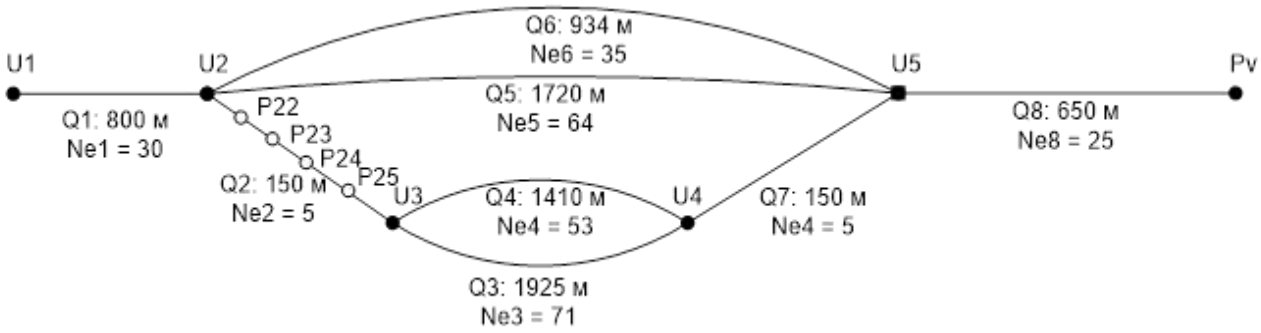


Рисунок 4.10 – Мережа із визначеними параметрами

На рис. 4.10: *U1* – атмосфера; *U2–U5* – внутрішні вузли; P_V – точка підключення вентилятора. Проведемо розрахунок кроку Δx, виходячи із відомої кількості процесорів системи і орієнтуючись на перший рівень паралельності «одне рівняння – один процес». Хай цільова MIMD–система рис. 4.9 має 576 процесорів, які відповідають 576 рівнянням. Оскільки кожний елемент мережі довжиною Δx описується двома рівняннями системи (4.5), мережу необхідно поділити на 288 елементів. Знайдемо загальну довжину мережі як суму довжин гілок:

$l_{мережі} = l_{Q1} + l_{Q2} + l_{Q3} + l_{Q4} + l_{Q5} + l_{Q6} + l_{Q7} + l_{Q8}$ = 800 + 150 + 1925 + 1410 + 1720 + 934 + 150 + 650 = 7739 м

Таким чином, крок дискретизації за координатою простору: $\Delta x = \frac{l_{\text{мережі}}}{N_e} = \frac{7739}{288} = 26{,}87$м.

Маючи значення кроку, можливо розрахувати кількість елементів для кожної окремої гілки (рис. 4.10 та табл. 4.2): гілка 1: : $N_{e1} = \frac{l_{Q1}}{\Delta x} = \frac{800}{26{,}87} = 30$ елементів; гілки 2 і 7: $N_{e2} = \frac{l_{Q2}}{\Delta x} = \frac{150}{26{,}87} = 5$ елементів; гілки 3–6, 8: аналогічно. Перевірочний підрахунок загальної кількості елементів мережі:

$N_e = \sum N_{ei} = N_{e1} + N_{e2} + N_{e3} + N_{e4} + N_{e5} + N_{e6} + N_{e7} + N_{e8} = 30 + 5 + 71 + 53 + 64 + 35 + 5 + 25 = 288$ елементів

На рис. 4.11 зображена мережа із позначенням кількості рівнянь Simulation–моделей гілок при обраному кроці дискретизації по простору Δx.

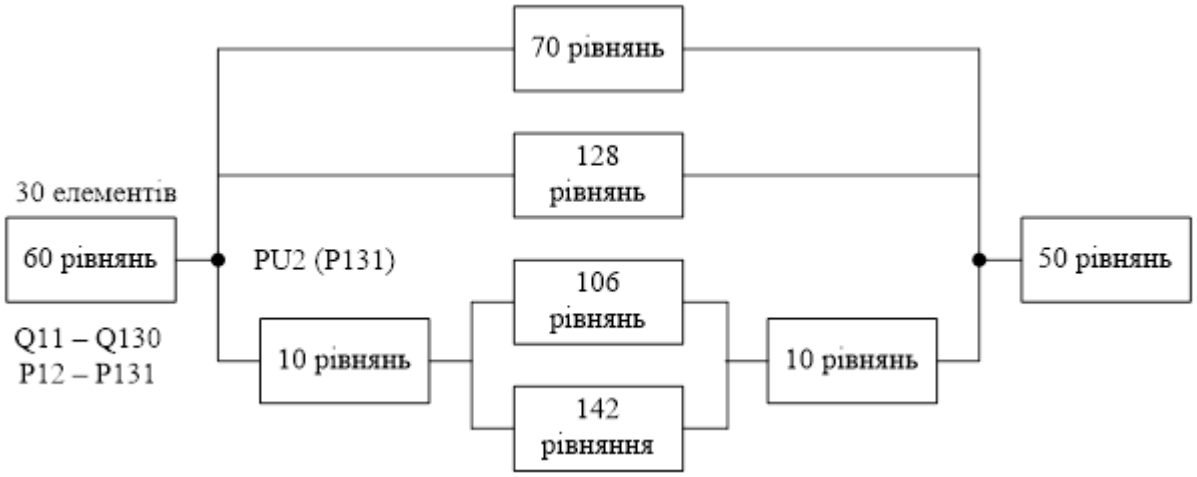


Рисунок 4.11 – Структура об’єкта і кількість рівнянь Simulation–моделей гілок

Перевірочний підрахунок загальної кількості рівнянь:

$N_p = \sum N_{pi} = N_{p1} + N_{p2} + N_{p3} + N_{p4} + N_{p5} + N_{p6} + N_{p7} + N_{p8} = 60 + 10 + 142 + 106 + 128 + 70 + 10 + 50 = 676$.

Дані й отримані параметри гілок мережного об’єкту зведені в табл. 4.2.

Таблиця 4.2 – Параметри гілок мережного об’єкту

Номер гілки	Довжина, м	Кількість елементів	Кількість CPU
Q1	800	30	60
Q2	150	5	10
Q3	1925	71	142
Q4	1410	53	106
Q5	1720	64	128
Q6	934	35	70
Q7	150	5	10
Q8	650	25	50

Для демонстрації принципів того, як імплементується дискретизація гілок по простору, розглянемо в якості прикладу реалізацію гілки Q1 з точки зору розподілення задач між процесорами та зв’язку по загальним даним, яку

мають між собою процесори. Ланцюг процесорів, що моделюють гілку Q1, показаний на рис. 4.12.

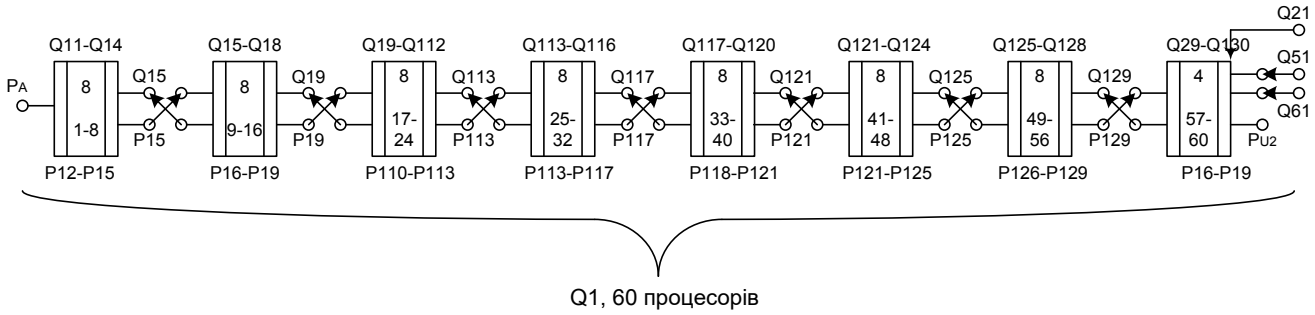


Рисунок 4.12 – Ланцюг процесорів, що моделюють гілку Q1

Рівняння у вузлах мережі відрізняються від позавузлових рівнянь, тому що повинні забезпечувати виконання першого закону Кірхгофа для вузлів. Їх програмна реалізація грає роль віртуального комутатора між вузлами. Наприклад, диференціальне рівняння для тиску у вузлі U2 (рис. 4.10) має наступний вигляд:

$$\frac{dP_{U2}}{dt} = \gamma(Q_{130} - Q_{21} - Q_{51} - Q_{61})$$

Зв’язок між собою рівнянь, що складають пару (один елемент), приведений на рис. 4.13.

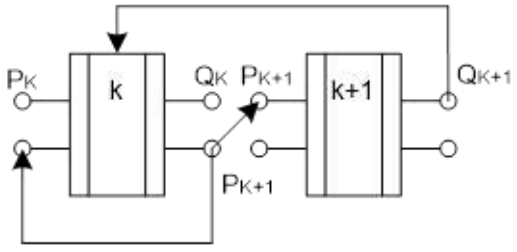


Рисунок 4.13 – Зв’язок усередині пари рівнянь

Система диференціальних рівнянь, яка описує сусідні елементи, має наступний вигляд:

$$\begin{cases} \dfrac{dQ_k}{dt} = \alpha(P_k - P_{k+1}) - \beta Q_k|Q_k| \\ \dfrac{dP_{k+1}}{dt} = \gamma(Q_k - Q_{k+1}) \\ \dfrac{dQ_{k+!}}{dt} = \alpha(P_{k+1} - P_{k+2}) - \beta Q_{k+1}|Q_{k+1}| \\ \dfrac{dP_{k+2}}{dt} = \gamma(Q_{k+1} - Q_{k+2}) \\ \cdots \end{cases}$$

На рис. 4.14 приведена схема взаємодії між процесами для першого рівня розпаралелювання.

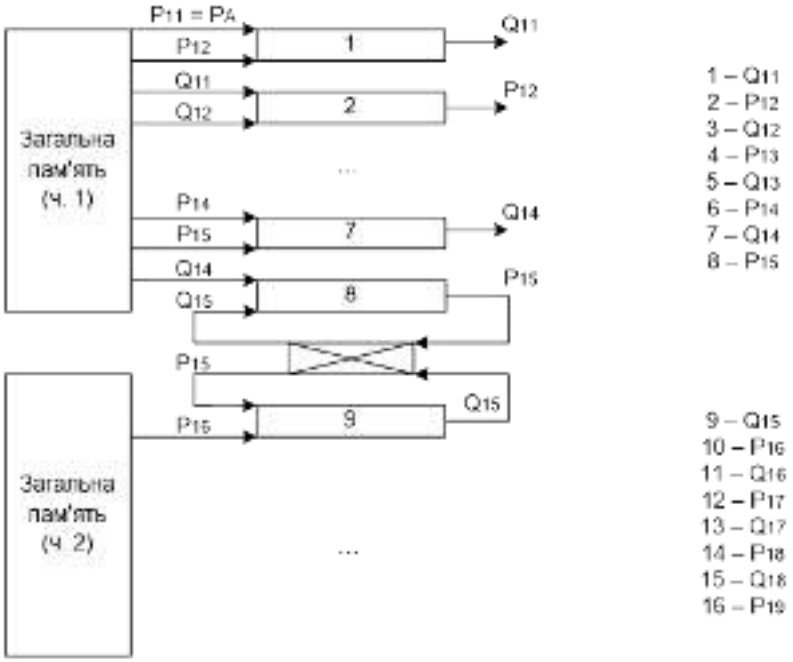


Рисунок 4.14– Схема взаємодії «один процес – одне рівняння»

Міжгілкові з’єднання, які забезпечують комутацію процесів у вузлі U2, приведені на рис. 4.15.

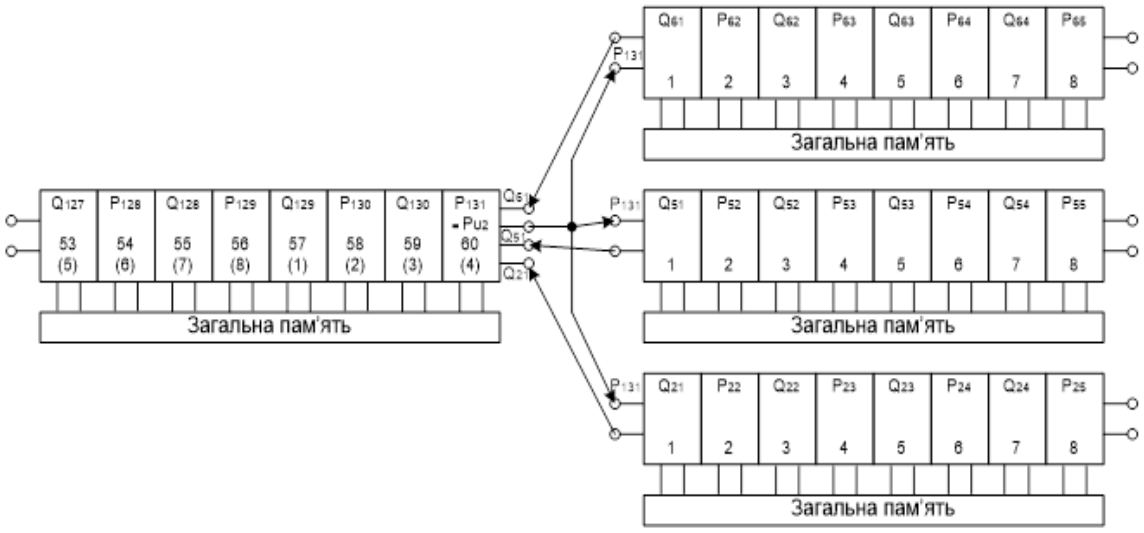


Рисунок 4.15 – Реалізація вузла U2

Для обчислення тиску P_{U2} потрібні значення повітряних потоків Q_{21}, Q_{51}, Q_{61}, а також Q_{130}, що обчислюється в сусідньому процесі. Вузлові рівняння для вузла U2 включають елементи чотирьох гілок (Q1, Q6, Q5, Q2) і мають наступний вигляд:

$$\begin{cases} \frac{dQ_{130}}{dt}=\alpha(P_{130}-P_{131})-\beta Q_{130}|Q_{130}| \\ \frac{dP_{U2}}{dt}=\gamma(Q_{130}-Q_{21}-Q_{51}-Q_{61}) \\ \frac{dQ_{61}}{dt}=\alpha(P_{U2}-P_{62})-\beta Q_{61}|Q_{61}| \\ \frac{dP_{62}}{dt}=\gamma(Q_{61}-Q_{62}) \\ \frac{dQ_{51}}{dt}=\alpha(P_{U2}-P_{52})-\beta Q_{51}|Q_{51}| \\ \frac{dP_{52}}{dt}=\gamma(Q_{52}-Q_{52}) \\ \frac{dQ_{21}}{dt}=\alpha(P_{U2}-P_{22})-\beta Q_{21}|Q_{21}| \\ \frac{dP_{22}}{dt}=\gamma(Q_{21}-Q_{22}) \\ \dots \end{cases}$$

Детальні міжгілкові з’єднання у вузлі U5 (рис. 4.8) приведені на рис. 4.16.

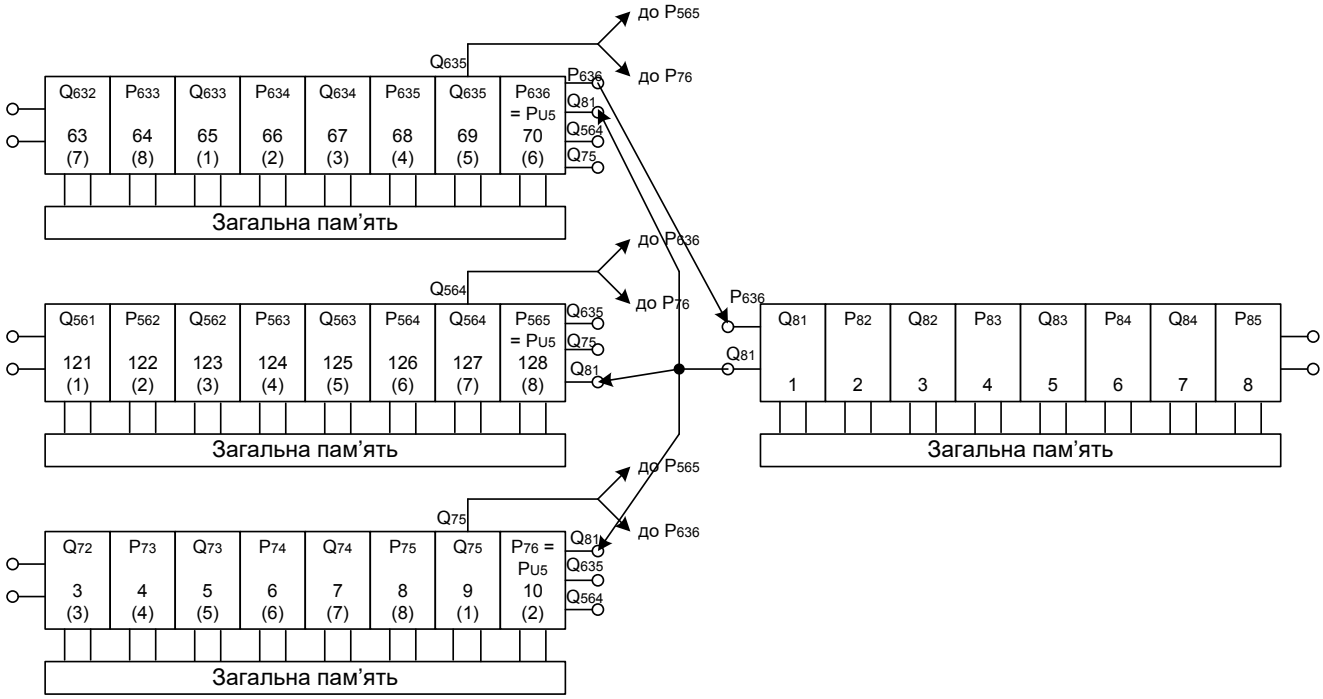


Рисунок 4.16 – Реалізація вузла U5

Вузлові рівняння для вузла U5 включають елементи чотирьох гілок (Q6, Q5, Q7, Q8) і мають наступний вигляд:

$$\begin{cases} \dfrac{dQ_{564}}{dt} = \alpha(P_{564} - P_{U5}) - \beta Q_{564}|Q_{564}| \\ \dfrac{dP_{U5}}{dt} = \gamma(Q_{564} + Q_{75} + Q_{635} - Q_{81}) \\ \dfrac{dQ_{635}}{dt} = \alpha(P_{635} - P_{U5}) - \beta Q_{635}|Q_{635}| \\ \dfrac{dQ_{75}}{dt} = \alpha(P_{75} - P_{U5}) - \beta Q_{75}|Q_{75}| \\ \dfrac{dQ_{81}}{dt} = \alpha(P_{81} - P_{U5}) - \beta Q_{81}|Q_{81}| \\ \dots \end{cases}$$

Детальні міжгілкові з'єднання у вузлі U3 приведені на рис. 4.17.

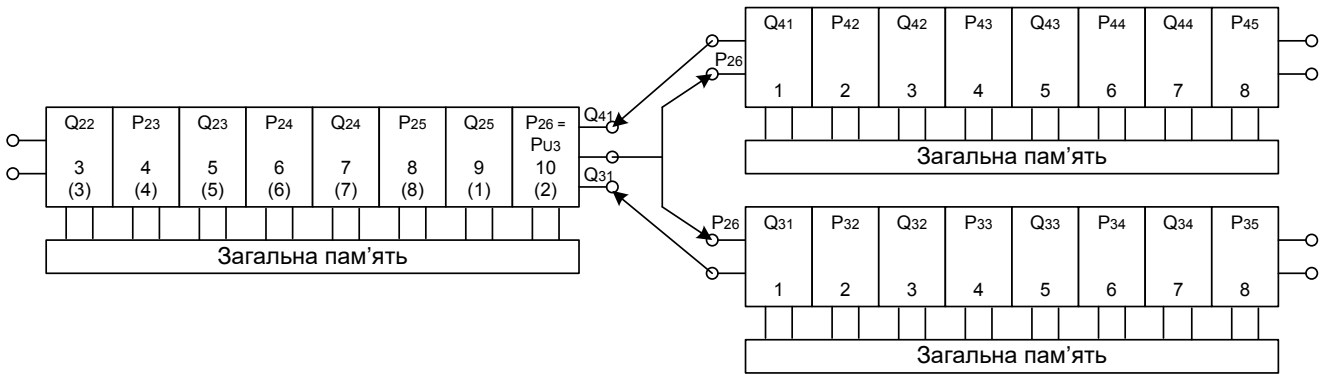


Рисунок 4.17 – Реалізація вузла U3

Потік повітря Q_{25} є сумою потоків Q_{31} і Q_{41} (початкових елементів пов'язаних з ним гілок). Вузлові рівняння для вузла U3 включають елементи трьох гілок (Q2, Q4, Q3) і мають наступний вигляд:

$$\begin{cases} \frac{dQ_{25}}{dt} = \alpha(P_{25} - P_{U5}) - \beta Q_{25}|Q_{25}| \\ \frac{dP_{U3}}{dt} = \gamma(Q_{25} - Q_{31} - Q_{41}) \\ \frac{dQ_{41}}{dt} = \alpha(P_{U3} - P_{42}) - \beta Q_{41}|Q_{41}| \\ \frac{dP_{42}}{dt} = \gamma(Q_{41} - Q_{42}) \\ \frac{dQ_{31}}{dt} = \alpha(P_{U3} - P_{32}) - \beta Q_{31}|Q_{31}| \\ \frac{dP_{32}}{dt} = \gamma(Q_{31} - Q_{32}) \\ \dots \end{cases}$$

Детальні міжгілкові з'єднання у вузлі U4 приведені на рис. 4.18.

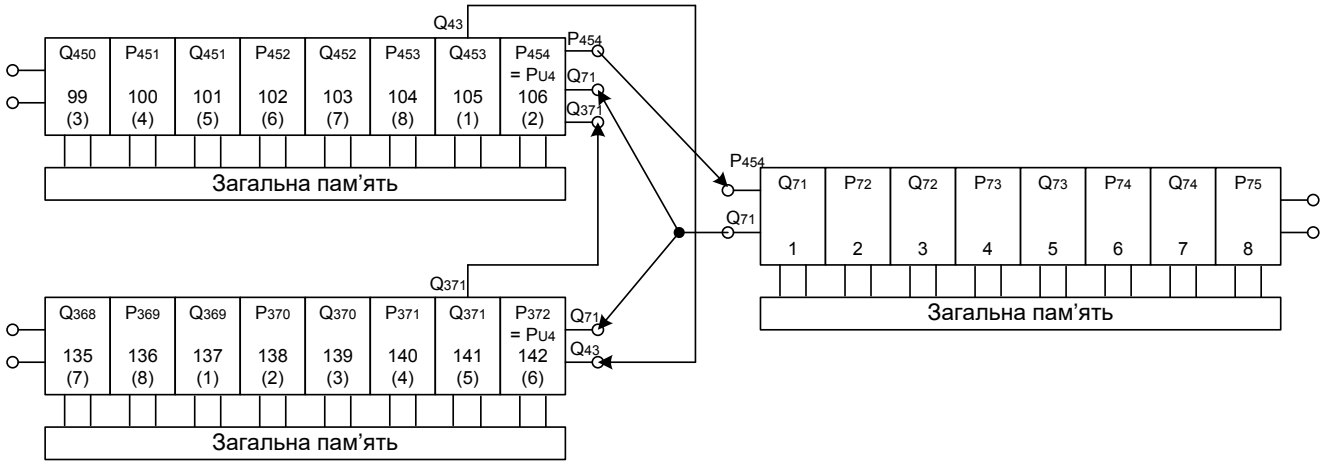


Рисунок 4.18 – Реалізація вузла U4

Вузлові рівняння для вузла U4 включають елементи трьох гілок (Q4, Q3, Q7) і мають наступний вигляд:

$$\begin{cases} \dfrac{dQ_{453}}{dt} = \alpha(P_{453} - P_{U4}) - \beta Q_{453}|Q_{453}| \\ \dfrac{dP_{U4}}{dt} = \gamma(Q_{453} - Q_{371} - Q_{71}) \\ \dfrac{dQ_{371}}{dt} = \alpha(P_{371} - P_{U4}) - \beta Q_{371}|Q_{371}| \\ \dfrac{dQ_{71}}{dt} = \alpha(P_{U4} - P_{72}) - \beta Q_{71}|Q_{71}| \\ \ldots \end{cases}$$

Блок–схему алгоритму функціонування OpenMP–симулятора показано на рис. 4.19 [19, 20].

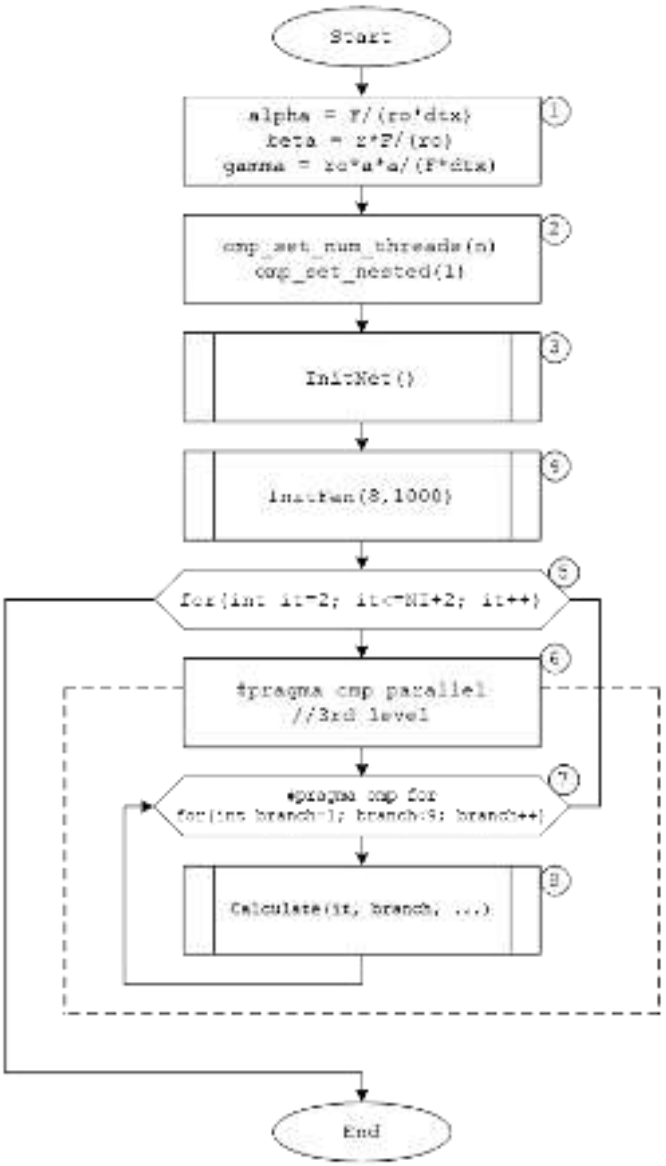


Рисунок 4.19 – Блок–схема функціонування OpenMP–симулятора

Пояснення до блоків схеми:

1 – обчислення констант, що використовуються в диференціальних рівняннях, на основі вхідних даних;

2 – встановлення максимальної кількості процесів для кожного із рівнів розпаралелювання, а також дозвіл на вкладений паралелізм;

3 – процедура ініціалізації мережі нульовими значеннями повітряного потоку і тиску, а також визначення довжини кожної із гілок (кількість елементів дискретизації);

4 – процедура ініціалізації тиску останнього елементу восьмої гілки (вентилятор);

5 – цикл часових ітерацій (одна ітерація дорівнює 1 мск); починається з 2, щоб можна було використовувати значення із індексами *it–1*, *it–2* як нульові;

6 – директива OpenMP, що відкриває розпаралелювання по гілкам мережі (третій рівень розпаралелювання): один процес обчислює одну гілку;

7 – цикл проходження по гілкам мережі із допоміжною директивою OpenMP;

8 – процедура обчислення гілки, вхідними параметрами для якої є номер ітерації і номер гілки.

Використані директиви OpenMP [15]:

1) #pragma omp parallel

Ця директива створює паралельний регіон для наступного за нею структурованого блоку. Вона повідомляє компілятору, що структурований блок коду повинен бути виконаний паралельно, в декількох потоках. Слід відзначити, що в кінці паралельного регіону виконується бар'єрна синхронізація (barrier synchronization). Інакше говорячи, досягнувши кінця регіону, всі потоки блокуються до тих пір, поки останній потік не завершить свою роботу.

За замовчуванням всі змінні в паралельному регіоні є загальними, але із цього правила є три виключення: приватними є індекси паралельних циклів for, локальні змінні блоків паралельних регіонів, а також будь–які змінні, вказані в розділах private, firstprivate, lastprivate і reduction. Таким чином, в розробленій програмі масиви тиску і потоку повітря H і Q є загальними як вказані поза межами паралельного регіону.

2) #pragma omp for

Ця директива відноситься до директив розподілення роботи (work–sharing directive). Такі директиви застосовуються не для паралельного виконання коду, а для логічного розподілення групи потоків, щоб реалізувати вказані конструкції управляючої логіки. Директива #pragma omp for повідомляє, що при виконанні циклу for в паралельному регіоні ітерації циклу повинні бути розподілені між потоками групи.

За замовчуванням в OpenMP для планування паралельного виконання циклів for застосовується алгоритм, що називається статичним плануванням (static scheduling). Це означає, що всі потоки групи виконують однакову кількість ітерацій циклу. Якщо n – кількість ітерацій циклу, а T – кількість потоків в групі, кожний потік виконає n/T ітерацій (припустимо, що n не ділиться на T без залишку) [19].

Використані функції OpenMP:

1) omp_get_thread_num().

Повертає номер потоку в групі потоків. Слід зазначити, що вона повертає не Windows–ідентификатор потоку, а число в діапазоні від 0 до omp_get_num_threads – 1.

2) omp_set_num_threads(nthreads)

Задає число потоків для виконання наступного паралельного регіону, який зустрінеться потоку, що виконується.

3) omp_set_nested(nested)

Якщо вхідний аргумент дорівнює «істині», вкладений паралелізм стає дозволеним (можливі декілька рівнів вкладеності) [19].

4.3.3 Орієнтація на MIMD–системи з комбінованою організацією пам’яті і MPI/OpenMP–засобами паралельного програмування

Розглянемо організацію обміну даними при реалізації паралельного симулятора МДОРП на цільовій MIMD–системі, структура якої показана на рис. 4.20. Тут в кожен обчислювальний вузол системи рис.4.6 підключено m процесорів (CPU) з загальною пам’яттю (Shared Memory). Обмін даними між CPU і синхронізація їх паралельної роботи програмується засобами стандарту OpenMP [10], а між вузлами – засобами MPI. Саме таку архітектуру має кластер ДонНТУ.

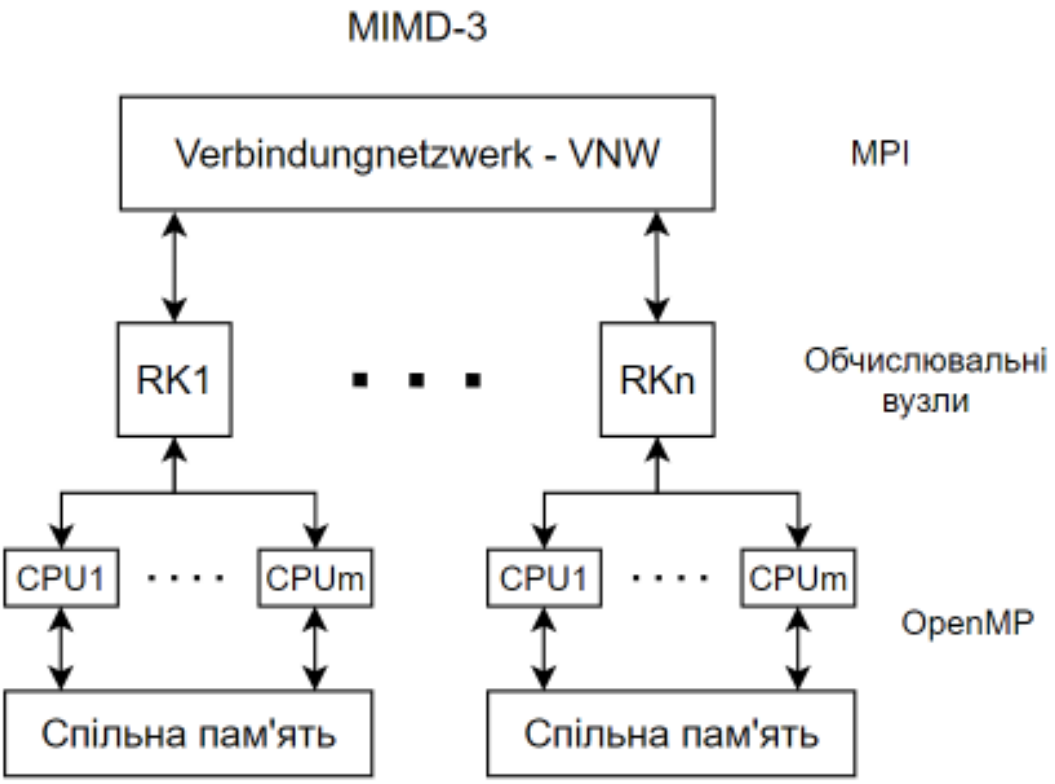


Рисунок 4.20 – MIMD–система з комбінованою організацію адресного простору

Для МДОРП як об’єкту паралельного моделювання з можливими чотирма рівнями розпаралелювання (розділ 2) пропонується наступна підготовка до реалізації його Simulation–моделі на цій архітектурі як поєднання підходів, розглянутих в п.4.3.1 та п.4.3.2:

1. Вузлові рівняння реалізуються процесорами обчислювальних вузлів на всіх можливих рівнях розпаралелювання.

2. З огляду на результати апріорного аналізу (розділ 2) пропонується зосередитись на третьому рівні РП–3 (рис.2.9).

3. Для варіанту «гілка – процес» (рис. 2.9b) треба розглянути дві можливих ситуації з відображенням на цільову MIMD–систему (ЦС):

3.1. Кількість гілок МДОРП **m** не перевищує кількість RKn (рис. 4.20) обчислювальних вузлів ЦС. Тут призначаємо Simulation–моделям кожної гілки процесор–вузол ЦС, а Mj (QP)j–процесів програмуємо на CPUm (рис. 4.20) процесорів з загальною пам’яттю засобами OpenMP (п. 4.3.2).

3.2. Кількість гілок МДОРП **m** > **RKn** (рис. 4.18). В цьому випадку виконуємо розподіл гілок на вузлові процесори за формулою **m / RKn** і отримуємо навантаження на CPUm (рис. 4.20) процесорів, зумовлене кількістю (QP)j–процесів **Mj*(m / RKn**). Розподіл процесів між CPU і паралельне програмування виконуємо з врахуванням результатів, отриманих в п. 4.3.1, 4.3.2.

4.3.4 Перспективна архітектура і розвиток підсистеми

Паралельна обчислювальна система HPE Hawk, що експлуатується в надпотужному обчислювальному центрі (HRLS) Штутгартського університету, має ієрархічну архітектуру обчислювальних вузлів з наступними рівнями ієрархії (рисунки 4.21 – 4.24):

– Ядро (Core);

– Складне ядро (Core CompleX) CCX – 4 ядра з взаємодією через L3 (доступ до загальної пам’яті цього рівня);

– Кластер з двох CCX – Cluster Complex Die (CCD), 8 ядер;

– Socket – 8 CCD, 64 ядра;

Вузол компонується з двох Socket’ів – 16 CCD, 128 ядер [21, 23].

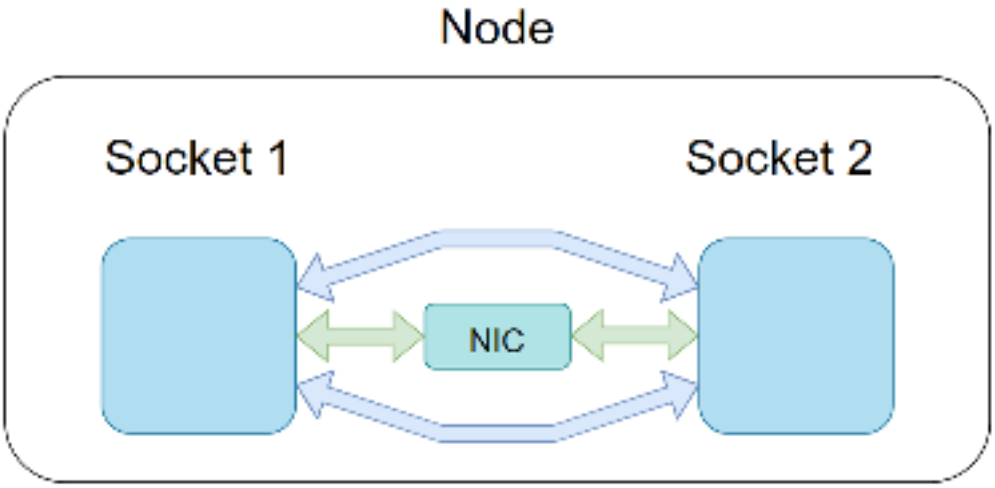


Рисунок 4.21 – Вузол системи HAWK

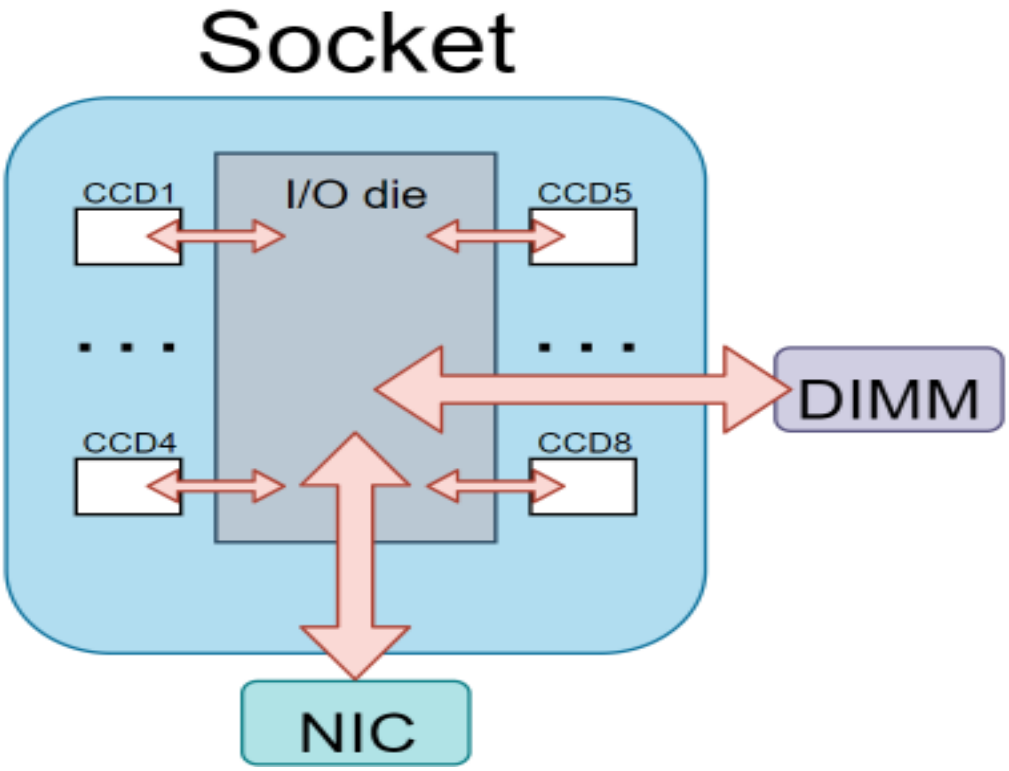


Рисунок 4.22 – Сокет системи HAWK

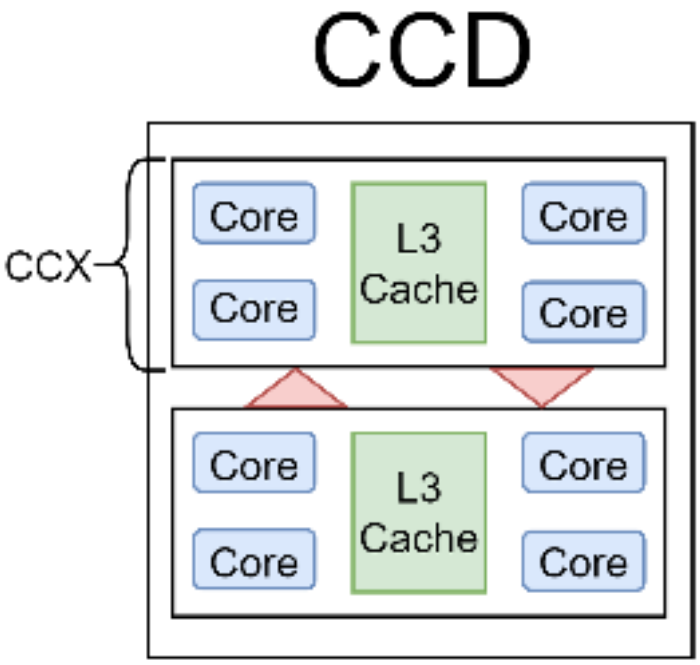


Рисунок 4.23 – Кластер ядер (складне ядро) системи HAWK

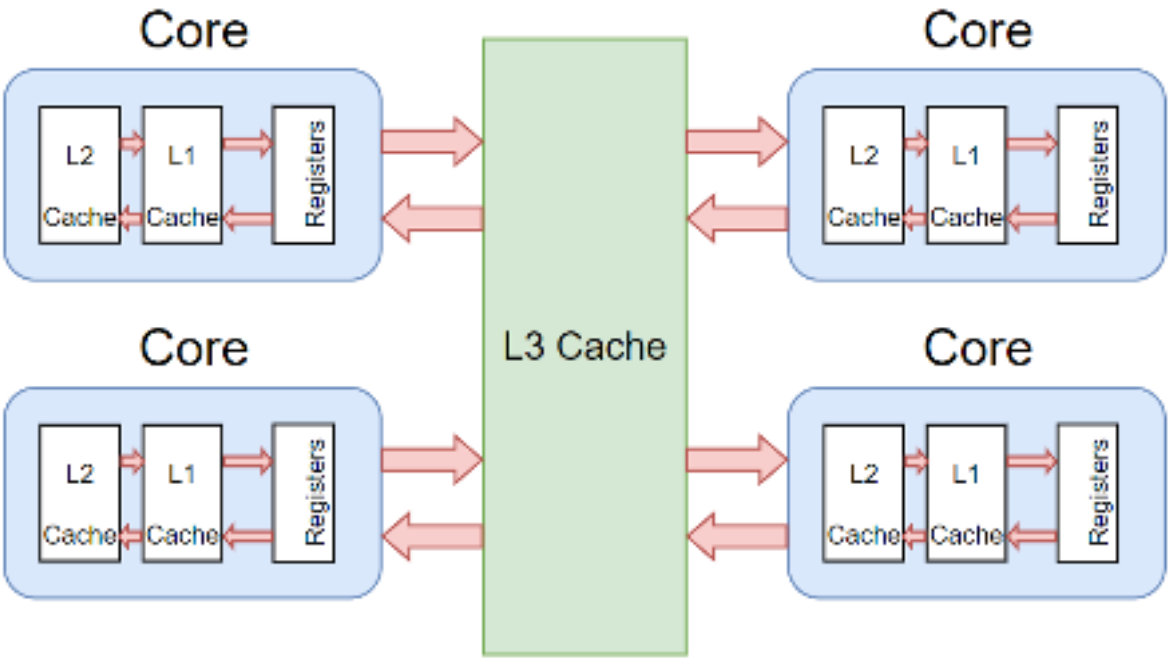


Рисунок 4.24 – Частина CCD системи HAWK

Також на рис. 4.24 можна побачити, що в складі ядер знаходиться кеш пам’ять 1–го, 2–го рівнів та регістрова пам’ять.

Обмін даними в кластері між серверами відбувається через поєднану мережу 40–портових Infiniband комутаторів. Швидкість з’єднань в обидва боки становить 200 Гбіт/с, а затримка при передачі між найближчими сусідами – 1.3 мікросекунди.

Кожні 16 серверів пов’язані одним Infiniband–комутатором. Ще 23 порти використовуються для зв’язку між комутаторами. Використовуючи ці зв’язки, комутатори між собою пов’язані у неповний 9–вимірний гіперкуб. В повному 9–вимірному гіперкубі при 16 серверах на один комутатор як одна грань куба використовувалося б 8192 сервери. Але так як усього HAWK в собі налічує 5632 сервери, то багатовимірний куб є урізаним [22].

На рис. 4.25 показано бажаний розподіл обчислювальних операцій при моделюванні гілки МДОРП. Ми виходимо з того, що в гілці як об’єкті з розподіленими параметрами Simulation–модель має ієрархію з наступними рівнями:

– одне рівняння;

– один елемент апроксимації з кроком Δx та два рівняння;

– вся гілка з Mj = lj/Δx елементами та 2Mj рівняннями.

Доцільно проаналізувати можливість реалізації рівнянь на рівнях «ядро – складне ядро CCX – конструктив CCD – конструктив Socket – вузол». Такий «апаратний» підхід диктуватиме можливий крок Δx: вузол реалізує 128 рівнянь, тобто допускає 64 кроки Δx і для всього МДОРП треба визначати цей крок, орієнтуючись на гілку максимальної довжини – $l_{max}/64$.

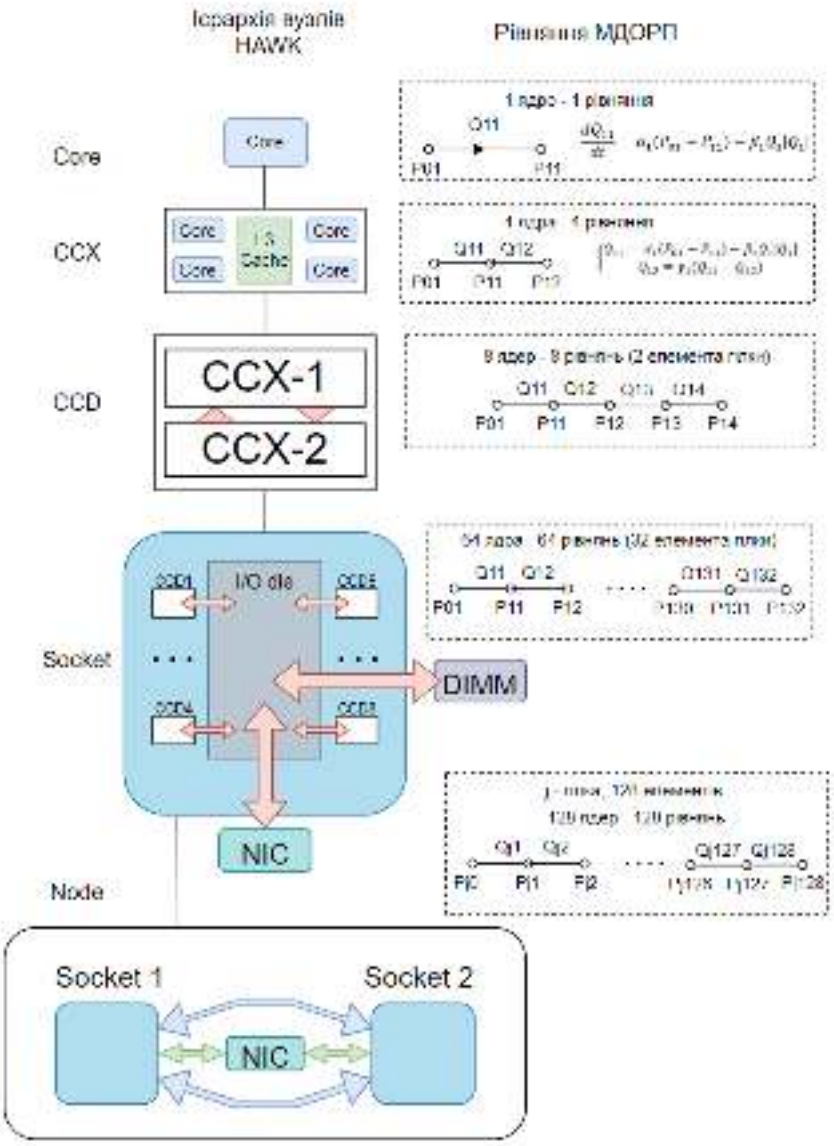


Рисунок 4.25 – Ієрархія обчислювальних вузлів системи Hawk і бажана кількість вирішуваних рівнянь гілок МДОРП.

4.4 Висновки

1. Підсистема обміну інформацією будується на базі апаратних засобів MIMD–кластера ДонНТУ з перспективою приєднання ресурсів HLRS Штутгартського університету. Програмування функцій підсистеми тісно пов’язано з архітектурно релевантним підходом до можливих рівнів розпаралелювання Simulation–моделей МДОРП і виконується мовою C/C++ з застосуванням засобів стандартів MPI/OpenMP.

2. При паралельному моделюванні МДОЗП доцільно визначати XY–процеси з програмуванням обміну між процесорами MPI–командами індивідуальної і колективної передачі/прийому повідомлень.

3. Розглянуто організацію обмінів інформацією при паралельному моделюванні МДОРП на трьох наявних архітектурах MIMD–систем.

4. Перспективним напрямком розвитку підсистеми є врахування ієрархії обчислювальних вузлів MIMD–системи Hawk, що приєднується до ресурсів ParSimTech–центру ДонНТУ.

ВИСНОВКИ

1. Підсистема обміну інформацією будується на базі апаратних засобів MIMD–кластера ДонНТУ з перспективою приєднання ресурсів HLRS Штутгартського університету. Програмування функцій підсистеми тісно пов'язано з архітектурно релевантним підходом до можливих рівнів розпаралелювання Simulation–моделей МДОРП і виконується мовою C/C++ з застосуванням засобів стандартів MPI/OpenMP. Сформульовані основні функції підсистеми забезпечать інформаційну підтримку користувачів РПМС на всіх етапах розробки паралельних симуляторів МДОРП і проведення модельних експериментів.

2. З огляду на проблемну орієнтацію РПМС і застосування перспективного блокового обчислювального методу (БОМ) визначено можливі рівні розпаралелювання (РП), дискретні форми Simulation–моделей та схеми обміну інформацією на кожному рівні.

3. Встановлено, що з ростом складності задачі моделювання збільшується й час витрачений на її розв'язання. Запропоновано й реалізовано тестери, який дозволяє користувачам визначити динамічні властивості системної складової РПМС «процесор–канал зв'язку–процесор».

4. При паралельному моделюванні МДОЗП доцільно визначати XY–процеси з програмуванням обміну між процесорами MPI–командами індивідуальної і колективної передачі/прийому повідомлень.

5. Розглянуто організацію обмінів інформацією при паралельному моделюванні МДОРП на трьох наявних архітектурах MIMD–систем. Системи з розподіленим адресним простором дозволяють побудову паралельних симуляторів з використанням MPI–засобів на всіх можливих рівнях розпаралелювання. Апаратно–програмні засоби систем з загальною пам'яттю привабливі можливістю апроксимації МДОРП по просторовій координаті в залежності від кількості процесорів. Поєднання засобів стандартів MPI і

OpenMP дозволяє врахувати всі властивості МДОРП при проблемній орієнтації РПМС.

6. Перспективним напрямком розвитку підсистеми є врахування ієрархії обчислювальних вузлів MIMD–системи Hawk, що приєднується до ресурсів ParSimTech–центру ДонНТУ.

СПИСОК ЛІТЕРАТУРИ

1. Svjatnyj V.A. Modellierung und Simulation der luftgasdynamischen Prozesse und Entwicklung der Automatisierungssysteme für Grubenbewetterung. Habilitationsarbeit / Святный В. А. Моделирование аэрогазодинамических процессов и разработка систем управления проветриванием угольных шахт; Дисс. докт. техн. наук – ДПИ, Донецк, 1985 – 407 с.

2. Комп'ютерні мережі: [навчальний посібник] / А. Г. Микитишин, М. М. Митник, П. Д. Стухляк, В. В. Пасічник. — Львів: «Магнолія 2006», 2013. — 256 с.

3. Network Routing: Algorithms, Protocols, and Architectures : [Електронний ресурс] : Medhi, Deepankar; Ramasamy, Karthik – Режим доступу : https://books.google.com.ua/books?id=IM–Y2W0RIF0C&dq=%22router+is+a+specialized+computer%22&pg=PA19&redir_esc=y#v=onepage&q=%22router%20is%20a%20specialized%20computer%22&f=false. – Загл. з екрану.

4. Brian Hill Cisco: The Complete Reference // Полный справочник по Cisco = Cisco: The Complete Reference. — «Вильямс». — С. 1088.

5. 10 Advantages of InfiniBand : [Електронний ресурс] : NADDOD. – Режим доступу: https://www.naddod.com/blog/top–10–advantages–of–infiniband. – Загл. з екрану.

6. "Verbs programming tutorial" : [Електронний ресурс] : Dotan Barak OpenSHEM, 2014. Mellanox. https://www.csm.ornl.gov/workshops/openshmem2014/documents/presentations_and_tutorials/Tutorials/Verbs%20programming%20tutorial–final.pdf. – Загл. з екрану.

7. Паралельні однокрокові методи чисельного розв'язання задачі Коші**:** монографія / Л.П. Фельдман, І.А. Назарова. – Донецьк: «ДВНЗ» ДонНТУ, 2011. – 185 с.

8. Svyatnyy V. A. Zur Entwicklung der problemorientierten verteilten parallelen Simulationsumgebung (POVPSU). Ein Bericht über den Forschungsaufenhalt. Vortrag am HLRS – Universität Stuttgart, 2022, – 74 c.

9. Svjatnyj V. A. Apriorische Analyse von aufgrund der blockartigen numerischen Verfahren diskretisierten Simulationsmodellen der dynamischen Netzobjekte mit verteilten Parametern (DNOVP). Forschungsbericht – MPI Magdeburg, 2023, – 75 c.

10. Message Passing Interface : [Електронний ресурс] : New Mexico State University – Режим доступу : https://hpc.nmsu.edu/discovery/mpi/introduction/ – Загл. з екрану.

11. Blocking receive for a message : [Електронний ресурс] : High – Performance Portable MPI – Режим доступу : https://www.mpich.org/static/docs/v3.2/www3/MPI_Recv.html – Загл. з екрану.

12. MPI_Send: [Електронний ресурс] : High – Performance Portable MPI – Режим доступу : https://www.mpich.org/static/docs/v3.3/www3/MPI_Send.html – Загл. з екрану.

13. MPI_Barrier:[Електронний ресурс]:High – Performance Portable MPI –Режим доступу: https://www.mpich.org/static/docs/v3.3/www3/MPI_Barrier.html – Загл. з екрану.

14. Timing Programs : [Електронний ресурс] : Lori Pollock – Режим доступу: https://www.eecis.udel.edu/~pollock/367/manual/node23.html – Загл. з екрану.

15. OpenMP Compilers & Tools : [Електронний ресурс] : OpenMP – Режим доступу: https://www.openmp.org/resources/openmp–compilers–tools/ – Загл. з екрану.

16. Countable loops in openMP : [Електронний ресурс] : Intel – Режим доступу: https://community.intel.com/t5/Intel–Moderncode–for–Parallel/Countable–loops–in–openMP/td–p/1160784 – Загл. з екрану.

17. Проектування комп'ютерної мережі високопродуктивного високо– продуктивного обчислювального кластеру / Корнілов О.О., Мірошкін О.М., Святний В.А. Покровськ: «ДВНЗ» ДонНТУ, 2022. – 13 с.

18. OpenMP Application Program Interface, Version 2.5 : [Електронний ресурс] : OpenMP Architecture Review Board – Режим доступу: www.openmp.org/mp–documents/spec25.pdf – Загл. з екрану.

19. Svjatnyj V. Virtuelle parallele Simulationsmodelle und Devirtualisie–rungsvorgang der Entwicklung von parallelen Simulatoren für dynamische Netzobjekte mit verteilten Parametern // Звіт про наукове співробітництво з HLRS – ДонНТУ, Донецьк, 2006 – 118 с.

20. Святный В.А., Гибридные вычислительные системы [Текст]/ В. А. Святный// Учеб. пособие для вузов по спец. "Электрон. вычисл. машины". – Киев: Вища школа, 1980. – С. 103–117.

21. The AMD EPYC Rome processor : [Електронний ресурс] : Björn Dick (HLRS), Thomas Bönisch (HLRS) – Режим доступу: https://kb.hlrs.de/platforms/upload/Processor.pdf – Загл. з екрану.

22. Hawk Interconnect Network : [Електронний ресурс] : Björn Dick (HLRS), Thomas Bönisch (HLRS), Bernd Krischok (HLRS) – Режим доступу: https://kb.hlrs.de/platforms/upload/Interconnect.pdf – Загл. з екрану.

23. The 2nd Generation AMD EPYC Processor Redefines Data Center Economicshttps://www.amd.com/system/files/documents/TIRIAS–White–Paper–AMD–Infinity–Architecture.pdf

ДОДАТОК А

Заява на затвердження теми і керівника

Завідувачу кафедри
Електронної техніки
Ковальову Сергію Олександровичу
студента групи КІм–22,
спеціальності 123 Комп'ютерна інженерія
Корнілова Олександра Олексійовича

ЗАЯВА

Прошу призначити керівником моєї роботи магістра д.т.н., проф. Святного Володимира Андрійовича

(вчена ступінь, вчене звання, посада, П.І.Б викладача)

і закріпити за мною наступну тему: «Підсистема обміну інформацією проблемно орієнтованого паралельного моделюючого середовища»

(найменування теми)

«__» _____ 2023__р. _____

(підпис студента)

ПОГОДЖЕНО:

Науковий керівник _____ Святний В. А.

(підпис) (прізвище та ініціали)

ДОДАТОК Б

Лістинги

Лістинг *MPI–програми MIMD–симулятора МДОЗП (m=8, n=5, γ=m–(n–1)=4)*

```
/* — INCLUDES ————————————————————————————————————————————————————————————
———————— */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "mpi.h"

/* — FUNCTIONS ———————————————————————————————————————————————————————————
———————— */

#define M                       4
#define N                       4
#define Tmax            150
#define MAX(A, B)       ((A) > (B) ? (A) : (B))

typedef float Matrix[M][N];
typedef float Vector[M];

void InitMatrix(char *filename, Matrix pMatrix)
 {

int i, j;
FILE *f_handler;
f_handler = fopen(filename, "rt");

if (f_handler == NULL)
printf("Error open file %s", filename);

for (i=0; i < M; i++)
for (j=0; j < N; j++)
fscanf(f_handler, "%f", &pMatrix[i][j]);
}/*InitMatrix*/

void InitVector(char *filename, Vector pVector)
 {
int i;
FILE *f_handler;
   f_handler = fopen(filename, "rt");
   if (f_handler == NULL)
printf("Error open file %s", filename);

for (i=0; i < M; i++){
                        fscanf(f_handler, "%f", &pVector[i]);
   }
}/*InitVector*/

void ShowMatrix(Matrix pMatrix)
```

```
{
    int i, j;

    for (i=0; i < M; i++)
    {
          for (j=0; j < N; j++)
        printf("%f ", pMatrix[i][j]);
          printf("\n");
     }
     printf("\n\n");
}/*ShowMatrix*/

void ShowVector(Vector pVector, char *string)
 {
          int i;
          printf("\n %s ", string);
          for (i=0; i < M; i++)
              printf("%f ", pVector[i]);
     printf("\n\n");
}/* ShowVector */

float VectorElement(Matrix pMatrix, Vector pVector, int rank)
{
   int i;
float element = 0.0;
for (i=0; i < M; i++)
element += pMatrix[rank][i]*pVector[i];
    return (element);
}

/* ____________________________________________________________________________
_____________ */

 main (int argc, char **argv)
{
   int my_rank;                          /* My process rank */
int p;                                          /* The number of process */
int t, i;

float h = 0.1;
float Xlocal, Ylocal;
float Zxlocal, Zylocal, Vlocal, VPlocal;
float Tmodel = 250.0;
float DeltaX, DeltaY, MaxDeltaX, MaxDeltaY;
float Delta = 0.01;

FILE *fpX;
FILE *fpY;

char XFileName[10], YFileName[10];

Matrix W, Tpx, Tpy, Rpx, Rpy;
Vector X, Y, Hx, Hy, Zx, Zy, V, Vp;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);
```

```
if (my_rank == 0)
 {
InitMatrix("w.dat", W);
InitMatrix("bpx.dat", Tpx);
InitMatrix("bpy.dat", Tpy);
InitMatrix("rpx.dat", Rpx);
InitMatrix("rpy.dat", Rpy);

InitVector("y.dat", Y);
InitVector("hx.dat", Hx);
InitVector("hy.dat", Hy);
}

MPI_Barrier(MPI_COMM_WORLD);

MPI_Bcast(W, M*N, MPI_FLOAT, 0, MPI_COMM_WORLD);
MPI_Bcast(Tpx, M*N, MPI_FLOAT, 0, MPI_COMM_WORLD);
MPI_Bcast(Tpy, M*N, MPI_FLOAT, 0, MPI_COMM_WORLD);
MPI_Bcast(Rpx, M*N, MPI_FLOAT, 0, MPI_COMM_WORLD);
MPI_Bcast(Rpy, M*N, MPI_FLOAT, 0, MPI_COMM_WORLD);

MPI_Bcast(Y, M, MPI_FLOAT, 0, MPI_COMM_WORLD);
MPI_Bcast(Hx, M, MPI_FLOAT, 0, MPI_COMM_WORLD);
MPI_Bcast(Hy, M, MPI_FLOAT, 0, MPI_COMM_WORLD);

MPI_Barrier(MPI_COMM_WORLD);

sprintf(XFileName, "Xproc%d", my_rank);
fpX = fopen(XFileName, "w");
sprintf(YFileName, "Yproc%d", my_rank);
fpY = fopen(YFileName, "w");

X[my_rank]=0.0;
for (t=0; t*h < Tmodel; t++) {

/* Calculation of the Vector X, Zx and Zy components */

Xlocal = VectorElement(W, Y, my_rank);
DeltaX = fabs(Xlocal – X[my_rank]);
Zxlocal = Xlocal*fabs(Xlocal);
Zylocal = Y[my_rank]*fabs(Y[my_rank]);

MPI_Barrier(MPI_COMM_WORLD);

/* Building of the Vectors X, Zx, Zy in root process */

MPI_Gather(&Xlocal, 1, MPI_FLOAT, X, 1, MPI_FLOAT, 0,
MPI_COMM_WORLD);
MPI_Gather(&Zxlocal, 1, MPI_FLOAT, Zx, 1, MPI_FLOAT, 0,
MPI_COMM_WORLD);
MPI_Gather(&Zylocal, 1, MPI_FLOAT, Zy, 1, MPI_FLOAT, 0,
MPI_COMM_WORLD);

MPI_Barrier(MPI_COMM_WORLD);

/* Send Vectors X, Zx, Zy to another processes */
```

```
MPI_Bcast(X, M, MPI_FLOAT, 0, MPI_COMM_WORLD);
MPI_Bcast(Zx, M, MPI_FLOAT, 0, MPI_COMM_WORLD);
MPI_Bcast(Zy, M, MPI_FLOAT, 0, MPI_COMM_WORLD);

MPI_Barrier(MPI_COMM_WORLD);

/* Output */

fprintf(fpX, "%f \n", X[my_rank]);
fprintf(fpY, "%f \n", Y[my_rank]);

/* Integration */
Vlocal = VectorElement(Tpx, Hx, my_rank) + VectorElement(Tpy, Hy, my_rank) –
         VectorElement(Rpx, Zx, my_rank) – VectorElement(Rpy, Zy, my_rank);
MPI_Barrier(MPI_COMM_WORLD);

if (t == 0)
Ylocal = Y[my_rank] + h*Vlocal;
else
Ylocal = Y[my_rank] + 1.5*h*Vlocal – 0.5*h*VPlocal;

MPI_Barrier(MPI_COMM_WORLD);

VPlocal = Vlocal;

MPI_Barrier(MPI_COMM_WORLD);

DeltaY = fabs(Ylocal – Y[my_rank]);

MPI_Reduce(&DeltaX, &MaxDeltaX, 1, MPI_FLOAT, MPI_MAX, 0,
MPI_COMM_WORLD);
MPI_Reduce(&DeltaY, &MaxDeltaY, 1, MPI_FLOAT, MPI_MAX, 0,
MPI_COMM_WORLD);

if (my_rank == 0)
if (MAX(MaxDeltaX, MaxDeltaY) < Delta)
t = Tmodel/h + 1;

MPI_Bcast(&t, 1, MPI_INT, 0, MPI_COMM_WORLD);

/* Building of the vector Y */
MPI_Gather(&Ylocal, 1, MPI_FLOAT, Y, 1, MPI_FLOAT, 0, MPI_COMM_WORLD);

/* Send Vector Y to another processes */
MPI_Bcast(Y, M, MPI_FLOAT, 0, MPI_COMM_WORLD);

MPI_Barrier(MPI_COMM_WORLD);
}
fclose(fpX);
fclose(fpY);
MPI_Finalize();
} /*main*/
```

Додаток: матриці та параметри тестового МДОЗП (структуровані по X,Y)

```
A
 1 –1  0  0  0  0 –1 –1
```

```
 0  1  0  0 –1 –1  0  0
 0  0 –1  0  1  1  0  0
 0  0  1 –1  0  0  1  1
```

S

```
 1  1  1  1  1  0  0  0
 1  1  1  1  0  1  0  0
 1  0  0  1  0  0  1  0
 1  0  0  1  0  0  0  1
```

Діагональні матриці R, K з діагоналями:

R

```
   0.387 0.072 0.072 0.314 3.0 2.677 1.933 1.275
```

K

```
   153.0 29.0  29.0  125.0 465.0 273.0 450.0 221.0
```

H–вектор

```
   ( 0     0     0     4100   0     0     0     0)^T
```

Вектор потоків в стаціонарному режимі після завершення перехідного процесу

$Q_{стац}$ (71.65 28.92 28.92 71.65 14.04 14.88 19.20 23.65)^T

```
   ( X1     X2   X3     X4    Y1    Y2    Y3     Y4)^T
```

Лістинг *MPI–програми MIMD–симулятора МДОРП (m=8, n=5)*

```
/* — INCLUDES ———————————————————————————— */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "mpi.h"

/* — ОПРЕДЕЛЕНИЕ КОНСТАНТ ———————————————————— */

#define M               10              // к–во подветвей
#define IT              10000           // к–во итераций
#define h               0.01            //
#define GQ              0.05            // точность для потоков воздуха
#define GP              0.05            // точность для давлений
#define m               8               // количество ветвей
#define n               5               // количество узлов
#define Ro              1.293           // плотность воздуха
#define a               331.5           // скорость звука

#define PROG            1               //постепенное возникновение возмущений

typedef double MatrixQ[m][M];
typedef double MatrixP[m][M+1];
typedef double Vector[m];
typedef double VectorX[M];

/* — MAIN FUNCTION ——————————————————————————— */
```

```
MatrixQ Q, Ptmp;
MatrixP P;
VectorX Xi_1, XPi_1, Xi_2, XPi_2, Qi, Pi;

int main(int argc, char **argv)
{
	int i, j, k, l, nNode, nSrc, nDest, nCounter, nLastNode;

	int my_rank; // Ранг процесса
	int p;	// Количество процессов

	double starttime, endtime; // Время выполнения программы

	char strFileName[20];  // Имя файла результатов

	//FILE* stream[m];
	FILE* stream;

	Vector L = {800.0, 150.0, 1925.0, 1410.0, 1720.0, 934.0, 150.0, 600.0};
			// Длины ветвей
	Vector S = {6.5, 6.5, 4.67, 5.98, 4.5, 4.97, 6.5, 6.5};
					// Площади ветвей
	Vector R = {0.484e–03, 0.484e–03, 1.056e–03, 1.9e–03, 1.12e–03, 1.37e–03, 0.484e–03, 0.484e–
03}; // Удельные сопротивления в ветвях
	double dE, alpha, beta, gamma;

	int A[n–1][m] = {1, –1, 0, 0, –1, –1, 0, 0,
					0,  1,–1,–1,  0,  0, 0, 0,
					0,  0, 1, 1,  0,  0,–1, 0,
					0,  0, 0, 0,  1,  1, 1,–1};

	MPI_Status status;
	MPI_Init(&argc, &argv);
	MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
	MPI_Comm_size(MPI_COMM_WORLD, &p);

	// Вычисление аэродинамических параметров, каждый процесс считает свое значение
	dE = L[my_rank] / M;
	alpha = (S[my_rank] / (Ro * dE));
	beta = ((R[my_rank] * S[my_rank]) / Ro);
	gamma = ((Ro * a * a) / (S[my_rank] * dE));

	if(my_rank == 0) // Задание граничных условий
	{
		starttime = MPI_Wtime();

		for(k = 0; k < m; k++)
		{
			for(j = 0; j <= M; j++)
			{
				if(j < M)
				{
					Q[k][j] = 0.0;
					Ptmp[k][j] = 0.0;
				}

				if((j == 0) && (k == 0) && (PROG == 1))
```

```
                        {
                                P[k][j] = 4100.0;
                        }else{
                                P[k][j] = 0.0;
                        }
                } // for(j)
        } // for(k)

        for(j = 0; j < M; j++)
        {
                Xi_1[j] = Xi_2[j] = 0.0;
                XPi_1[j] = XPi_2[j] = 0.0;
                Qi[j] = 0.0;
                Pi[j] = 0.0;
        }
}else{
        for(k = 0; k < m; k++)
        {
                for(j = 0; j <= M; j++)
                {
                        if(j < M)
                        {
                                Q[k][j] = 0.0;
                        }
                        P[k][j] = 0.0;
                } // for(j)
        } // for(k)

        for(j = 0; j < M; j++)
        {
                Xi_1[j] = Xi_2[j] = 0.0;
                XPi_1[j] = XPi_2[j] = 0.0;
                Qi[j] = 0.0;
                Pi[j] = 0.0;
        }
}
MPI_Barrier(MPI_COMM_WORLD);

sprintf(strFileName, "out%d.txt", my_rank + 1);
//stream[my_rank] = fopen(strFileName, "w");
stream = fopen(strFileName, "w");

nLastNode = –1;
i = 0;

while(i < IT)
{
        if((my_rank == 0)&&(PROG == 1))
        {
                if(i < 500)
                {
                        P[0][0] = i * 8.2;
                }else{
                        P[0][0] = 4100.0;
                }
        }
        for(j = 0; j < M; j++)
```

```
		{
			if((my_rank < (m–1)) && (j == (M–1)))
			{
				MPI_Send(&XPi_1[j], 1, MPI_DOUBLE, my_rank + 1, 0,
MPI_COMM_WORLD);
			}

			Xi_1[j] = alpha * (P[my_rank][j] – P[my_rank][j+1]) –
					beta * (Q[my_rank][j] * fabs(Q[my_rank][j]));

			if(j < M–1)
			{
				XPi_1[j] = gamma * (Q[my_rank][j] – Q[my_rank][j+1]);
			}else{
				nNode = 0;
				XPi_1[j] = 0.0;
				for(l = 0; l < n–1; l++)
				{
					if(A[l][my_rank] == 1)
					{
						nNode = l;
						if(my_rank > 0)
						{
							if(A[l][my_rank – 1] == 1)
							{
								nLastNode = 0;
							}
						}
						break;
					}
				} // for(l)

				if(nLastNode == 0)
				{
					if(my_rank > 0)
					{
						MPI_Recv(&XPi_1[j], 1, MPI_DOUBLE, my_rank – 1,
0, MPI_COMM_WORLD, &status);
					}
					nLastNode = –1;
				}else{
					for(l = 0; l < m; l++)
					{
						switch(A[nNode][l])
						{
							case 1:
								XPi_1[j] = XPi_1[j] + Q[l][M–1];
								break;
							case –1:
								XPi_1[j] = XPi_1[j] – Q[l][0];
								break;
						}
					}
					XPi_1[j] = gamma * XPi_1[j];
				} // if(nLastNode == 0)
			} // if(j < M–1)
			MPI_Barrier(MPI_COMM_WORLD);
```

```
                        // Метод Адамса–Башфорта
                        switch(i)
                        {
                                case 0:
                                        Qi[j] = 0.0;
                                        Pi[j] = 0.0;
                                        break;
                                case 1: // первая итерация
                                        Qi[j] = Q[my_rank][j] + h * Xi_1[j];
                                        Pi[j] = P[my_rank][j+1] + h * XPi_1[j];
                                        break;
                                default: // вторая и последующие итерации
                                        Qi[j] = Q[my_rank][j] + (h/2) * (3*Xi_1[j] – Xi_2[j]);
                                        Pi[j] = P[my_rank][j+1] + (h/2) * (3*XPi_1[j] – XPi_2[j]);
                        } // switch(i)
                        MPI_Barrier(MPI_COMM_WORLD);

                        Xi_2[j] = Xi_1[j];
                        XPi_2[j] = XPi_1[j];
                        Q[my_rank][j] = Qi[j];

                        if((my_rank < m–1) || (j < M–1)) // P[m–1][M] всегда равен 0
                        {
                                P[my_rank][j+1] = Pi[j];
                        }
                        MPI_Barrier(MPI_COMM_WORLD);
                } // for(j)

                MPI_Gather(Qi, M, MPI_DOUBLE, Q, M, MPI_DOUBLE, 0, MPI_COMM_WORLD);
                MPI_Gather(Pi, M, MPI_DOUBLE, Ptmp, M, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
                MPI_Barrier(MPI_COMM_WORLD);

                if(my_rank == 0)
                {
                        for(k = 0; k < m; k++)
                        {
                                for(j = 0; j < M; j++)
                                {
                                        if((k < m–1) || (j < M–1)) // P[m–1][M] всегда равен 0
                                        {
                                                P[k][j+1] = Ptmp[k][j];
                                        }
                                }
                        }

                        for(k = 0; k < (n–1); k++)
                        {
                                nSrc = 0;
                                nDest = 0;
                                nCounter = 0;
                                for(l = 0; l < m; l++)
                                {
                                        if((A[k][l] == 1) && (nCounter != 1))
                                        {
                                                nCounter++;
```

```
                                        nSrc = l;
                                    }else if(A[k][l] == –1)
                                    {
                                        nDest = l;
                                        P[nDest][0] = P[nSrc][M];
                                    }
                                } // for(l)
                            } // for(k)
                        } // if(my_rank == 0)
                        MPI_Barrier(MPI_COMM_WORLD);

                        MPI_Bcast(Q, m * M, MPI_DOUBLE, 0, MPI_COMM_WORLD);
                        MPI_Bcast(P, m * (M+1), MPI_DOUBLE, 0, MPI_COMM_WORLD);
                        MPI_Barrier(MPI_COMM_WORLD);
                        //if ((i%10==0))//&&(my_rank==0))
                           //fprintf(stream[my_rank], "%0.4f;%0.4f\n", Q[my_rank][0], Q[my_rank][M–1]);
                           fprintf(stream, "%0.9f;%0.9f\n", P[my_rank][0], P[my_rank][M–1]);

                        i++;
                } // while(i)
                //fclose(stream[my_rank]);
                fclose(stream);
                if(my_rank == 0)
                {
                        endtime = MPI_Wtime();
                        printf ("\ntime = %f\n", endtime–starttime);
                }

                MPI_Finalize();

                return 0;
} // main()
```

Програмні засоби і параметри МДОРП, що використані при побудові OpenMP–симулятора

Паралельний вирішувач рівнянь був реалізований на мові C++ в рамках середовища Microsoft Visual C++ 2005. Причина такого вибору полягає в інтегрованості OpenMP–стандарту в дане середовище. Активізація реалізованих в компіляторі засобів OpenMP здійснюється за допомогою параметру компілятора /openmp. OpenMP зв’язується з прикладними програмами через бібліотеку імпорту vcomp.lib. Відповідна бібліотека періоду виконання називається vcomp.dll. Версії бібліотек імпорту і періоду виконання (vcompd.lib і vcompd.dll відповідно) для відлагодження підтримують додаткові повідомлення про помилки, що генеруються при деяких неприпустимих

операціях. Можливість використання функцій OpenMP здійснюється шляхом підключення бібліотеки omp.h.

Візуалізація результатів роботи програми була виконана за допомогою бібліотеки MFC.

При моделюванні шахтної вентиляційній мережі були використані наступні параметри та константи:

$\Delta x = 26{,}87$ – крок дискретизації за координатою простору (розрахунок див. в пункті 4.1) [м];

$h = 0{,}006$ – крок метода Адамса–Башфорта [сек];

$\rho = 1{,}25$ – щільність повітря [кг/м^3];

l_i – довжини гілок мережі [м] – див. п. 4.1;

$F = 8{,}9$ – площа перерізу повітроводу [м^2];

$r = 3{,}9*10^{-4}$ – постійний аеродинамічний опір [Нс^2/м^8];

$H = 1000$ – депресія вентилятору, що розташований в кінці восьмої гілки [Па];

$a = 330$ – швидкість повітря [м/с];

$$\alpha = \frac{F}{\rho \cdot \Delta x} = 0{,}257$$

$$\beta = \frac{r \cdot F}{\rho} = 0{,}0027$$

$$\gamma = \frac{\rho \cdot a^2}{F \cdot \Delta x} = 587{,}4$$

Кількість ітерацій: 20000 (таким чином, час моделювання дорівнює 0,006*20000 = 120 секунд)

Код основних модулів програми OpenMP–симулятора

```
#include "stdafx.h"
#include "MAth.h"
#include "2DStatic.h"
#include "math.h"
#include "conio.h"
#include "stdio.h"
```

```
#include "omp.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

double P[NI+3][9][NE+1];
double Q[NI+3][9][NE];
double h = 0.009,F = 8.9,ro = 1.25,r = 3.9E–4,a = 330,dtx = 26.87, alpha, beta, gamma;

int nthreads = 8, chunk = 1000;
int steps[9];
CRect r2;

//_____________________________________________________________
void C2DStatic::Calculate(CComboBox* myCombo1, int it, int branch, FILE * pFile, int tid3)
        {
        int     prevbranch = branch–1, nextbranch = branch+1,
                curSteps,curSteps4, prevSteps, tid2, tid1;
        double  prevP1=0, prevP2=0, nextQ1=0, nextQ2=0, f1, f2;

        curSteps = steps[branch];
        prevSteps = steps[prevbranch];

        switch(branch)
                {
                case 0: {   break;} //Test branch
                case 1: {
                                nextQ1 = Q[it–1][6][0] + Q[it–1][5][0] + Q[it–1][2][0];
                                nextQ2 = Q[it–2][6][0] + Q[it–2][5][0] + Q[it–2][2][0];
                                break;
                                } //Branch 1
                case 2: {
                                prevP1 = P[it–1][1][steps[1]];
                                prevP2 = P[it–2][1][steps[1]];
                                nextQ1 = Q[it–1][3][0] + Q[it–1][4][0];
                                nextQ2 = Q[it–2][3][0] + Q[it–2][4][0];
                                break;
                                } //Branch 2
                case 3: {
                                prevP1 = P[it–1][2][steps[2]];
                                prevP2 = P[it–2][2][steps[2]];
                                nextQ1 = Q[it–1][7][0] – Q[it–1][4][steps[4]–1];
                                nextQ2 = Q[it–2][7][0] – Q[it–2][4][steps[4]–1];
                                break;
                                } //Branch 3
                case 4: {
                                prevP1 = P[it–1][2][steps[2]];
                                prevP2 = P[it–2][2][steps[2]];
                                nextQ1 = Q[it–1][7][0] – Q[it–1][3][steps[3]–1];
                                nextQ2 = Q[it–2][7][0] – Q[it–2][3][steps[3]–1];
                                break;
                                } //Branch 4
                case 5: {
                                prevP1 = P[it–1][1][steps[1]];
```

```
                                prevP2 = P[it–2][1][steps[1]];
                                nextQ1 = Q[it–1][8][0] – Q[it–1][7][steps[7]–1] – Q[it–1][6][steps[6]–1];
                                nextQ2 = Q[it–2][8][0] – Q[it–2][7][steps[7]–1] – Q[it–2][6][steps[6]–1];
                                break;
                                } //Branch 5
                        case 6: {
                                prevP1 = P[it–1][1][steps[1]];
                                prevP2 = P[it–2][1][steps[1]];
                                nextQ1 = Q[it–1][8][0] – Q[it–1][7][steps[7]–1] – Q[it–1][5][steps[5]–1];
                                nextQ2 = Q[it–2][8][0] – Q[it–2][7][steps[7]–1] – Q[it–2][5][steps[5]–1];
                                break;
                                } //Branch 6
                        case 7: {
                                prevP1 = P[it–1][3][steps[3]];
                                prevP2 = P[it–2][3][steps[3]];
                                nextQ1 = Q[it–1][8][0] – Q[it–1][6][steps[6]–1] – Q[it–1][5][steps[5]–1];
                                nextQ2 = Q[it–2][8][0] – Q[it–2][6][steps[6]–1] – Q[it–2][5][steps[5]–1];
                                break;
                                } //Branch 7
                        case 8:  {
                                prevP1 = (P[it–1][6][steps[6]]);
                                prevP2 = (P[it–2][6][steps[6]]);
                                break;
                                } //Branch 8
                        }

                //#pragma omp parallel //2nd level
                        {
                        curSteps4 = (int)(curSteps/4+1);
                        //#pragma omp for
                        for (int i4=0; i4<curSteps4; i4++)
                                {
                                tid2 = omp_get_thread_num();
                                //#pragma omp parallel //1st level
                                        {
                                        //#pragma omp for
                                        for (int i8=0; i8<8; i8++)
                                                {
                                                int i = i4*4+i8/2;
                                                if(i<curSteps)
                                                        {
                                                        if((i8%2)==0)
                                                                {
                                                                tid1 = omp_get_thread_num();
                                                                //fprintf(pFile,"(it%5d) L3: %d, L2: %d, L1: %d
did branch %d step %d\n",it,tid3,tid2,tid1,branch,i);
                                                                if((i==0)&&(branch>1))
                                                                        {
                                                                        f1 = alpha*(prevP1–P[it–
1][branch][i+1])–beta*Q[it–1][branch][i]*abs(Q[it–1][branch][i]);
                                                                        f2 = alpha*(prevP2–P[it–
2][branch][i+1])–beta*Q[it–2][branch][i]*abs(Q[it–2][branch][i]);
                                                                        Q[it][branch][i] = Q[it–1][branch][i] +
h/2*(3*f1–f2);
                                                                        }
                                                                else
                                                                        {
```

```
                                                f1 = alpha*(P[it–1][branch][i]–P[it–
1][branch][i+1])–beta*Q[it–1][branch][i]*abs(Q[it–1][branch][i]);
                                                f2 = alpha*(P[it–2][branch][i]–P[it–
2][branch][i+1])–beta*Q[it–2][branch][i]*abs(Q[it–2][branch][i]);
                                                Q[it][branch][i] = Q[it–1][branch][i] +
h/2*(3*f1–f2);
                                                }
                                        }
                                else
                                        if((i+1)==curSteps)
                                                {
                                                if(branch<8)
                                                        {
                                                        f1 = gamma*(Q[it–1][branch][i]–
nextQ1);
                                                        f2 = gamma*(Q[it–2][branch][i]–
nextQ2);
                                                        P[it][branch][i+1] = P[it–
1][branch][i+1] + h/2*(3*f1–f2);
                                                        }
                                                }
                                        else
                                                {
                                                f1 = gamma*(Q[it–1][branch][i]–Q[it–
1][branch][i+1]);
                                                f2 = gamma*(Q[it–2][branch][i]–Q[it–
2][branch][i+1]);
                                                P[it][branch][i+1] = P[it–1][branch][i+1]
+ h/2*(3*f1–f2);
                                                }
                                        }//if i<curSteps
                                }//for i8
                        }//parallel
                }//for i4
        }//parallel
}//Calculate

//————————————————————————————————————————
void C2DStatic::InitNet()
        {
        int tid;

        steps[0] = 5;
        steps[1] = 30;
        steps[2] = 5;
        steps[3] = 71;
        steps[4] = 53;
        steps[5] = 64;
        steps[6] = 35;
        steps[7] = 5;
        steps[8] = 25;

        //printf("Initializing P and Q...\n");
        for(int b=0; b<9; b++)
                {
                #pragma omp parallel
                        {
```

```
                        tid = omp_get_thread_num();
                        #pragma omp for schedule (dynamic, chunk)
                        for (int it=0; it<(NI+1); it++)
                                {
                                for (int i=0; i<(NE+1); i++)
                                        {
                                        //printf("Thread=%d initialized Q%d\n",tid,i);
                                        P[b][it][i] = 0;
                                        }
                                }
                        }
                }
        for(int b=0;b<9;b++)
                {
                #pragma omp parallel
                        {
                        tid = omp_get_thread_num();
                        #pragma omp for schedule (dynamic, chunk)
                        for (int it=0; it<(NI+1); it++)
                                for (int i=0; i<NE; i++)
                                        {
                                        //printf("Thread=%d initialized Q%d\n",tid,i);
                                        Q[b][it][i] = 0;
                                        }
                        }
                }
        }

//---------------------------------------------------------------------
void C2DStatic::InitFan(int branch, double Pbeg)
        {
        int tid;
        #pragma omp parallel
                {
                tid = omp_get_thread_num();
                #pragma omp for schedule (dynamic, chunk)
                for (int it=0; it<(NI+3); it++)
                        {
                        P[it][branch][steps[branch]] = –Pbeg;
                        }
                }
        }

//---------------------------------------------------------------------
C2DStatic::C2DStatic()
        {
        // Set Default Value
        x_Min = –10;
        x_Max = 10;
        y_Min = –1.5;
        y_Max = 1.5;
        }

//---------------------------------------------------------------------
C2DStatic::~C2DStatic()
        {
        }
```

```
//————————————————————————————————————————
BEGIN_MESSAGE_MAP(C2DStatic, CStatic)
	//{{AFX_MSG_MAP(C2DStatic)
		// NOTE – the ClassWizard will add and remove mapping macros here.
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

//————————————————————————————————————————
// C2DStatic message handlers
CPoint C2DStatic::GetNewPixel(float x,float y)
	{
	CPoint NewPoint;
	NewPoint.x=int((stcWidth/(x_Max–x_Min))*x–((stcWidth*x_Min)/(x_Max–x_Min)));
	NewPoint.y=int(stcHeight/(y_Max–y_Min)*y–(stcHeight*y_Max)/(y_Max–y_Min));
	return NewPoint;
	}

//————————————————————————————————————————
void C2DStatic::SetInitialValue(float xMin,float xMax,float yMin,float yMax)
{
	x_Min=xMin;
	x_Max=xMax;
	y_Min=yMin;
	y_Max=yMax;
}

//————————————————————————————————————————
void C2DStatic::DrawCoords(CDC* pDC, int proc)
	{
	UpdateData(TRUE);
	float index;
	CPoint cp;
	COLORREF crColor;
	char buffer[180];

	crColor=RGB(0,0,255);
	CPen pen (PS_SOLID, 1, crColor);
	pDC–>SelectObject (&pen);
	//draw numbers
	pDC–>Rectangle(&r2);
	for (index=(x_Max–x_Min)/10;index<=x_Max;index+=(x_Max–x_Min)/10)
	{
		cp=GetNewPixel(index,y_Min);
		sprintf_s(buffer, 100, "%2.0f", index);
		pDC–>TextOut(cp.x–20,cp.y,buffer);
	}
	for (index=y_Min;index<=y_Max;index+=(y_Max–y_Min)/10)
	{
		cp=GetNewPixel(0,index);
		sprintf_s(buffer, 100, "%2.0f", index);
		pDC–>TextOut(–20,cp.y+10,buffer);
	}
	//draw Y lines
	cp=GetNewPixel(0,0);
	pDC–>MoveTo(cp);
	for (int it=0;it<=NI;it+=(NI/50))
```

```
        {
                float x1 = it;
                float y1 = 0;
                cp=GetNewPixel(x1,y1);
                if(it%(NI/10)==0)
                        cp.SetPoint(cp.x,0);
                else
                        cp.SetPoint(cp.x,–stcHeight+10);
                pDC–>MoveTo(cp);
                cp.SetPoint(cp.x,–stcHeight);
                pDC–>LineTo(cp);
        }
        //draw X lines
        cp=GetNewPixel(0,0);
        pDC–>MoveTo(cp);
        for (int i=y_Min;i<=y_Max;i+=1)
                {
                float x1 = 0;
                float y1 = i;
                cp=GetNewPixel(x1,y1);
                if(i%5==0)
                        cp.SetPoint(stcWidth,cp.y);
                else
                        cp.SetPoint(10,cp.y);
                pDC–>MoveTo(cp);
                cp.SetPoint(0,cp.y);
                pDC–>LineTo(cp);
                }
        if(proc==1)
                {
                sprintf_s(buffer, 100, "In progress...");
                pDC–>TextOut(10,–10,buffer);
                }
        }

//————————————————————————————————————————
void C2DStatic::ShowFunc(CDC* pDC, int branch)
        {
        UpdateData(TRUE);
        float index;
        CPoint cp;
        COLORREF crColor;

        int red, green, blue;

        switch(branch)
        {
        case 0: red=255, green=0,   blue=0;   break;
        case 1: red=120, green=120, blue=0;   break;
        case 2: red=255, green=0,   blue=255; break;
        case 3: red=0,   green=255, blue=0;   break;
        case 4: red=0,   green=0,   blue=255; break;
        case 5: red=120, green=0,   blue=120; break;
        case 6: red=0,   green=255, blue=255; break;
        case 7: red=120, green=120, blue=120; break;
        case 8: red=0,   green=0,   blue=0;   break;
        default: break;
```

```
		}

		crColor=RGB(red,green,blue);
		CPen pen2 (PS_SOLID, 1, crColor);
		pDC–>SelectObject (&pen2);
		cp=GetNewPixel(0,0);
		pDC–>MoveTo(cp);
		for (int it=0;it<=NI;it+=1)
			{
			float x1 = it;
			float y1 = Q[it][branch][0];
			cp=GetNewPixel(x1,y1);
			if ((cp.x>=0 && cp.y<=0) && (cp.x<=stcWidth && cp.y>=–stcHeight) )
				pDC–>LineTo(cp);
			}
}

//————————————————————————————————————
bool C2DStatic::DrawFunction(CComboBox* myCombo1, int todo)
		{
		CRect rect;
		char buffer[180];
		CPoint p1, p2;
		CClientDC dc (this);
		FILE * pFile;
		pFile =  fopen ("OpenMP Branches & Steps.txt","w");
		int tid, cnt=0, ticks1, ticks2;

		alpha = F/(ro*dtx);
		beta = r*F/(ro);
		gamma = ro*a*a/(F*dtx);
		// printf(" a=%6.4f \n",alpha); printf(" b=%6.4f \n",beta); printf(" g=%6.4f \n",gamma);

		if(todo<0)
			todo = myCombo1–>GetCurSel();
		omp_set_num_threads(nthreads);
		omp_set_nested(1);
		//printf("Starting calculation with %d threads\n",nthreads);

		GetClientRect(&rect);
		stcWidth=rect.Width();
		stcHeight=rect.Height();

		dc.SetMapMode(MM_ISOTROPIC);
		dc.SetWindowExt(stcWidth, stcHeight);
		dc.SetViewportExt(stcWidth, –stcHeight);
		dc.SetViewportOrg(0,0);

		p1.SetPoint(0,0);
		p2.SetPoint(stcWidth,–stcHeight);
		r2.SetRect(p1,p2);

		ticks1 = GetTickCount();
		switch(todo)
			{
			case 0:
				{
```

```
                        DrawCoords(&dc,1);
                        InitNet();
                        InitFan(8,1000);

                        for(int it=2; it<=NI+2; it++)
                                {
                                //h=h–0.005/NI;
                                if(it<NI/2) h=h–0.002/(NI/2);
                                //#pragma omp parallel
                                        {
                                        //#pragma omp for
                                        for(int branch=1; branch<9; branch++)
                                                {
                                                cnt++;
                                                tid = omp_get_thread_num();
                                                //fprintf(pFile,"Thread %d did branch %d;
\n",tid,branch);
                                                Calculate(myCombo1, it, branch, pFile, tid);
                                                if(cnt==7)
                                                        {
                                                        //fprintf(pFile,"\n");
                                                        cnt=0;
                                                        }
                                                }
                                        }
                                }
                        DrawCoords(&dc,0);
                        for(int branch=1; branch<9; branch++)
                                ShowFunc(&dc,branch);
                        break;
                        }
                case 9:
                        {
                        DrawCoords(&dc,0);
                        for(int branch=1; branch<9; branch++)
                                ShowFunc(&dc,branch);
                        break;
                        }
                case 10:
                        {
                        DrawCoords(&dc,0);
                        break;
                        }
                default:
                        {
                        ShowFunc(&dc,todo);
                        break;
                        }
                }
        fclose (pFile);
        ticks2 = GetTickCount();
        sprintf_s(buffer, 100, "Expended time: %d milisecs",(ticks2–ticks1));
        dc.TextOut(10,28,buffer);
        return TRUE;
        }

//————————————————————————————————————————
```

```
void C2DStatic::FileReport()
	{
	FILE * pFile[9];

	pFile[1] = fopen ("Qbranch1.txt","w");
	pFile[2] = fopen ("Qbranch2.txt","w");
	pFile[3] = fopen ("Qbranch3.txt","w");
	pFile[4] = fopen ("Qbranch4.txt","w");
	pFile[5] = fopen ("Qbranch5.txt","w");
	pFile[6] = fopen ("Qbranch6.txt","w");
	pFile[7] = fopen ("Qbranch7.txt","w");
	pFile[8] = fopen ("Qbranch8.txt","w");

	for(int branch=1; branch<9; branch++)
		{
		fprintf(pFile[branch],"Q%2d Iter \\ Steps", branch);
		for (int i=0; i<steps[branch]; i++)
			fprintf(pFile[branch], "%6d  ",i+1);
		fprintf(pFile[branch], "\n");
		for(int it=2; it<=NI+2; it+=10)
			{
			fprintf(pFile[branch], "Iteration %5d: ",it–2);
			for (int i=0; i<steps[branch]; i++)
				fprintf(pFile[branch], "%6.2f  ",Q[it][branch][i]);
			fprintf(pFile[branch], "\n");
			}
		fclose (pFile[branch]);
		}


	pFile[1] = fopen ("Pbranch1.txt","w");
	pFile[2] = fopen ("Pbranch2.txt","w");
	pFile[3] = fopen ("Pbranch3.txt","w");
	pFile[4] = fopen ("Pbranch4.txt","w");
	pFile[5] = fopen ("Pbranch5.txt","w");
	pFile[6] = fopen ("Pbranch6.txt","w");
	pFile[7] = fopen ("Pbranch7.txt","w");
	pFile[8] = fopen ("Pbranch8.txt","w");

	for(int branch=1; branch<9; branch++)
		{
		fprintf(pFile[branch],"Q%2d Iter \\ Steps", branch);
		for (int i=0; i<steps[branch]; i++)
			fprintf(pFile[branch], "%6d  ",i+1);
		fprintf(pFile[branch], "\n");
		for(int it=2; it<=NI+2; it+=10)
			{
			fprintf(pFile[branch], "Iteration %5d: ",it–2);
			for (int i=0; i<=steps[branch]; i++)
				fprintf(pFile[branch], "%6.2f  ",P[it][branch][i]);
			fprintf(pFile[branch], "\n");
			}
		fclose (pFile[branch]);
		}

	}
```

Лістинг MPI програми тестера

```
#include <mpi.h>
#include <stdio.h>
#include <math.h>
#include <unistd.h>
double startTime, endTime, deltaTime;

int exchange(int my_rank){
   //starting parameters
   double xNew, xOld;
   double yNew, yOld;
   double kx1, kx2, kx3, kx4;
   double ky1, ky2, ky3, ky4;
   double ampl = 1;
   double fi = 0;
   double f = 5;
   double dMax = 1.0; //0.1%
   dMax/=100.0;
   double tMax = 100.0;
   double w = 2 * M_PI * f;
   double h = dMax / w;
   double tSim = h;
   MPI_Status status;
   //work with file
   /*FILE *pFile2;
   if(my_rank == 0){
      pFile2 = fopen("sin.txt", "w");
      fclose(pFile2);
      pFile2 = fopen("sin.txt", "a");
   }*/
   //before calculation
   if(my_rank==0){
      xOld = ampl * sin(fi);
   }
   else {
      yOld = ampl * cos(fi);
   }
   //calculation
   do{
      if(my_rank == 0){
         MPI_Send(&xOld, 1, MPI_DOUBLE, 1, 1, MPI_COMM_WORLD);
         MPI_Recv(&yOld, 1, MPI_DOUBLE, 1, 4, MPI_COMM_WORLD, &status);
         kx1=h*w*yOld;
         kx2 = h * (w * yOld + kx1 / 2.0);
         kx3 = h * (w * yOld – (0.5 – sqrt(0.5)) * kx1 + (1 – sqrt(0.5)) * kx2);
         kx4 = h * (w * yOld – sqrt(0.5) * kx2 + (1 + sqrt(0.5)) * kx3);
         xNew = xOld + kx1/6.0 + (1–sqrt(0.5))*kx2/3.0 + (1+sqrt(0.5))*kx3/3.0 + kx4/6.0;
         xOld = xNew;
         //write to file
         /*char buffer [50];
         sprintf (buffer, "%f   %f\n", tSim, ampl * sin(w * tSim + fi));
         fprintf(pFile2, buffer);*/
         tSim += h;
      }
      else {
         MPI_Recv(&xOld, 1, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD, &status);
         MPI_Send(&yOld, 1, MPI_DOUBLE, 0, 4, MPI_COMM_WORLD);
         ky1 = h * (–w * xOld);
         ky2 = h * (–w * xOld + ky1 / 2.0);
         ky3 = h * (–w * xOld – (0.5 – sqrt(0.5)) * ky1 + sqrt(0.5) * ky2);
```

```
            ky4 = h * (–w * xOld – sqrt(0.5) * ky2 + (1 + sqrt(0.5)) * ky3);
            yNew = yOld + ky1/6.0 + (1–sqrt(0.5))*ky2/3.0 + (1+sqrt(0.5))*ky3/3.0 + ky4/6.0;
            yOld = yNew;
            tSim += h;
        }
    } while (tSim <= tMax);

    return 1;
}


int main(int argc, char** argv) {
    int my_rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if(size < 2) {
        MPI_Finalize();
        return 0;
    }
    if(my_rank==0){
        for(int i=1; i<size; i++){

            startTime = MPI_Wtime();
            exchange(my_rank);
            endTime = MPI_Wtime();
            deltaTime = endTime – startTime;
            printf("0:%d=%fsec\n\n", i, deltaTime);
            MPI_Barrier(MPI_COMM_WORLD);
        }
    }
    else{
        for(int i=1; i<size; i++){

            if(my_rank==i){
                exchange(my_rank);
            }
            MPI_Barrier(MPI_COMM_WORLD);
        }
    }
    /*if(my_rank==0){
        for(int i=1; i<size; i++){
            startTime = MPI_Wtime();
            exchange(my_rank, size);
            endTime = MPI_Wtime();
            deltaTime = endTime – startTime;
            printf("0:%d=%fsec\n\n", i, deltaTime);
            //printf("done\n");
        }
    }
    else{
        exchange(my_rank, size);
    }

    if(my_rank==0)
        startTime = MPI_Wtime();
    exchange(my_rank, size);
    if(my_rank==0){
        endTime = MPI_Wtime();
        deltaTime = endTime – startTime;
        printf("time to run = %f sec\n\n", deltaTime);
        printf("done\n");
```

```
    }*/
    MPI_Finalize();
    return 0;
}
```

Лістинг OpenMP програми тестера

```
#include <stdio.h>
#include <math.h>
#include <omp.h>


int main() {
    omp_set_num_threads(2);
    double xCur, xPrev;
    double yCur, yPrev;
    double ampl = 1;
    double fi = 0;
    double f = 5;
    double dMax = 75.0; //0.1%
    dMax/=100.0;
    double tMax = 1.;
    double w = 2 * M_PI * f;
    double h = dMax / w;
    double tSim = h;
    int first=0;
    int count=0;
    double tSim2=0;
    for(;;){
        if(tSim>tMax){
            break;
        }
        count++;
        tSim+=h;
    }
    tSim=h;
    for(int i=0;i<count;i++){
        tSim+=h;
        if(i>=count/2){
            printf("%d %f %f\n", i, tSim, tSim2);
            tSim2=tSim;
            break;
        }
    }
    tSim=h;
    #pragma omp parallel for schedule(static, count/2+1) private(first, xPrev, xCur, yPrev, yCur, tSim)
    for(int i=1; i<count;i++) {
        int thread_id = omp_get_thread_num();
        if(tSim <= (h+0.001)){
            if (thread_id == 1 ){
                xPrev = ampl * sin(tSim2);
                yPrev = ampl * cos(tSim2);
                tSim=tSim2;
            }
            else {
                printf("%f\n\n",tSim);
                xPrev = ampl * sin(tSim);
                yPrev = ampl * cos(tSim);
            }
        }

            double x1 = h * w * yPrev;
```

```
double x2 = h * (w * (yPrev + x1 / 2.0));
double x3 = h * (w * (yPrev – (0.5 – sqrt(0.5)) * x1 + (1 + sqrt(0.5)) * x2));
double x4 = h * (w * (yPrev – sqrt(0.5) * x2 + (1 – sqrt(0.5)) * x3));
xCur = xPrev + x1 / 6.0 + (1 – sqrt(0.5)) * x2 / 3.0 + (1 + sqrt(0.5)) * x3 / 3.0 + x4 / 6.0;

double y1 = h * (–w * xPrev);
double y2 = h * (–w * (xPrev + y1 / 2.0));
double y3 = h * (–w * (xPrev – (0.5 – sqrt(0.5)) * y1 + (1 + sqrt(0.5)) * y2));
double y4 = h * (–w * (xPrev – sqrt(0.5) * y2 + (1 – sqrt(0.5)) * y3));
yCur = yPrev + y1 / 6.0 + (1 – sqrt(0.5)) * y2 / 3.0 + (1 + sqrt(0.5)) * y3 / 3.0 + y4 / 6.0;

yPrev = yCur;
xPrev = xCur;


        tSim+=h;
    }
      return 0;
    }
```

ДОДАТОК В

Охорона праці

В.1 Заходи щодо створення сприятливих умов при роботі з персональним комп’ютером (ПК)

Згідно з НПАОП 0.00-1.31-99 «Правила охорони праці під час експлуатації електронно-обчислювальних машин», експлуатація персонального комп'ютера супроводжується впливом на організм комплексу факторів трудового процесу та виробничого середовища. У результаті наступають зміни функціонального стану центральної системи організму, що характеризуються вираженою напругою. Зміни, що накопичуються в процесі роботи, призводять до ризику розвитку певних захворювань. З метою мінімізації впливу негативних факторів на організм людини при роботі із ПК варто дотримуватися правил охорони праці при роботі із ПК. На рис. В.1 наведено схему приміщення лабораторії.

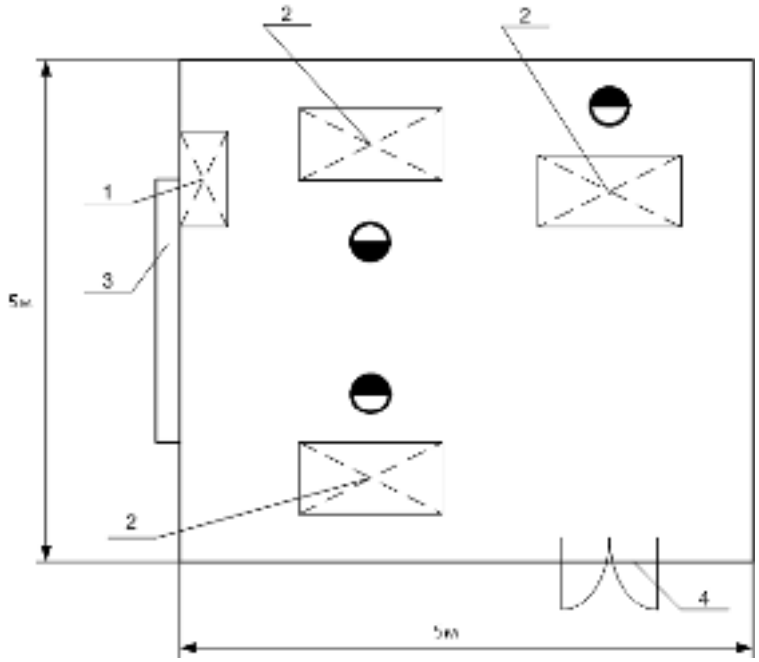


Рисунок В.1 – Схема приміщення лабораторії

Умовні позначення: 1-кондиціонер; 2-комп’ютер (3 шт.); 3-вікно; 4- двері;

У приміщеннях, в яких перебувають робочі місця з ПК, є освітлення, з однобічним розміщенням світопрорізів, площа яких не перевищує 25% від площі стіни. Віконні прорізи в приміщенні з ПК мають регульовані жалюзі, або штори, або інші сонцезахисні пристрої.

Конструкцію робочого місця згідно з ГОСТ 12.2.032-78, «ССБТ. Рабочее место при выполнении работ сидя. Общие эргономические требования», маємо забезпечити підтримання оптимальної робочої пози з такими ергономічними характеристиками: ступні ніг – на підлозі або на підставці для ніг; стегна – в горизонтальній площині; передпліччя - вертикально; лікті – під кутом 90 градусів до вертикальної площини; зап'ястя зігнуті під кутом не більш ніж 20 градусів відносно горизонтальної площини; нахил голови 15 - 20 градусів відносно вертикальної площини.

Екран та клавіатура мають розташовуватися на оптимальній відстані від очей користувача, але не ближче 600 мм, з урахуванням розміру алфавітно-цифрових знаків та символів, загалом 600-1000 мм.

Клавіатуру слід розміщувати на поверхні столу або на спеціальній, регульованій за висотою, робочій поверхні окремо від столу на відстані 100 - 300 мм від краю, ближчого до працівника. Кут нахилу клавіатури має бути в межах 15 градусів.

Висота робочої поверхні столу має бути 800 мм, а ширина – повинна забезпечувати можливість виконання операцій в зоні досяжності моторного поля.

Робочий стіл повинен мати простір для ніг висотою не менше 600 мм, шириною не менше 500 мм, глибиною на рівні колін не менше 450 мм, на рівні витягнутої ноги – не менше 650 мм.

Ширина та глибина сидіння повинні бути не меншими за 400 мм. Висота поверхні сидіння має регулюватися в межах 400 - 500 мм, а кут нахилу поверхні – від 15 градусів вперед до 5 градусів назад.

Висота спинки сидіння має становити 300 ± 20 мм, ширина – не менше 380 мм, радіус кривизни в горизонтальній площині – 400 мм. Кут нахилу спинки

повинен регулюватися в межах 0 - 30 градусів відносно вертикального положення. Відстань від спинки до переднього краю сидіння повинна регулюватись у межах 260 - 400 мм.

Для зниження статичного напруження м'язів рук необхідно застосовувати стаціонарні або знімні підлокітники довжиною не менше 250 мм, шириною – 50 - 70 мм, що регулюються по висоті над сидінням у межах 230 ± 30 мм та по відстані між підлокітниками в межах 350 - 500 мм.

У конструкцію сидіння можуть бути введені додаткові елементи, що не є обов'язковими: підголівник та підставка для ніг.

У приміщеннях, де розташовано монітори, потрібно виконувати заходи щодо боротьби зі статичною електрикою. Найбільш простим способом відповідно до рекомендацій, є підтримка відносної вологості повітря на рівні 50-60% за допомогою кондиціонеру з функцією клімат-контроля.

Згідно з ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень» у приміщеннях з ПК параметри мікроклімату повинні бути такими: температура навколишнього середовища повинна бути в межах 25-22°С, а швидкість руху повітря – 0,1- 0,2м/с.

В.2 Розрахунок кондиціювання повітря

Мікроклімат впливає на самопочуття людини, його працездатність і протікання фізіологічних процесів, від яких залежить підтримка постійності температури тіла. Теплові дії на організм можуть з'явитися причиною швидкого стомлення, зниження працездатності, ослаблення, опірності організму до шкідливих дій, різних захворювань.

Кондиціювання повітря – це створення та автоматична підтримка в приміщеннях постійних чи визначених метеорологічних умов, що змінюються по програмі, найбільш сприятливих для людей, що працюють чи необхідних для нормального протікання технологічного процесу.

При виділенні тепла у виробничому приміщенні визначають необмежений повітрообмін виходячи з рівності виділеного тепла в помешканні та виділення вентиляцією.

$$L = \frac{Q_{\text{надл}}}{c \cdot p_n \cdot (t_{\text{вид}} - t_{\text{пр}})}, \text{м}^3/\text{год} \qquad \text{(В.1)}$$

де $Q_{\text{надл}}$ – надлишкова кількість тепла, тобто різниця між його приходом і відходом, ккал/год;

c – теплообмініість повітря, кДж/кг ºС;

p_n – щільність приточного повітря, кг/м^3;

$t_{\text{пр}}$ і $t_{\text{вид}}$ – відповідно температура приточного повітря і того, що виділяється, ºС.

У зв'язку з тим, що загально обмінна вентиляція повинна забезпечити допустиму температуру повітря на робочі місця, то

$$t_{\text{вид}} = t_{\text{р.з.}} + t\ (H - 2) \qquad \text{(В.2)}$$

де $t_{\text{р.з.}}$ – припустима температура повітря в робочій зоні відповідно до ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень»;

H – висота виробничого приміщення, м;

t – температурний градієнт по висоті приміщення, ºС;

2 – висота робочої зони, м.

Звичайно приймають t = 2-5 ºС.

$$t_{\text{вид}} = 21 + 2(3 - 2) = 23\ ºC$$

Тепловиділення від електричного двигуна при нормальному завантаженні мають бути розраховані за формулою:

$$Q_1 = \frac{N\ (1 - \eta) \cdot 860}{\eta}, \text{ккал/год} \qquad \text{(В.3)}$$

де N – номінальна потужність електродвигуна,

$$Q_1 = \frac{0,7\ (1 - 0,95) \cdot 860}{0,95} = 31,7\ (\frac{\text{ккал}}{\text{год}})$$

Тепловиділення від джерел освітлення, ккал/год

$$Q_2 = q \cdot E_{\text{н}} \cdot S\ (\frac{\text{ккал}}{\text{год}}) \qquad \text{(В.4)}$$

де Q_2 – тепло від джерел світу, ккал/год;

$E_{\text{н}}$ – нормована освітленість, відповідно СНиП II-4-79. «Строительные нормы и правила. Естественное и искусственное освещение. Нормы проектирования».

$q\ =\ 0,05$ ккал – тепло, що виділяється на 1 м^2 освітлюваній поверхні.

$$Q_2 = 0,05 \cdot 400 \cdot 25\ =\ 500\ (\frac{\text{ккал}}{\text{год}})$$

Розрахуємо тепло, що поступає у приміщення від сонячної радіації за формулою:

$$Q_3 = F_{\text{ост}} \cdot q_{\text{ост}} \cdot A_{\text{ост}}\ =\ 500\ (\frac{\text{ккал}}{\text{год}}) \qquad \text{(В.5)}$$

де $F_{\text{ост}}$ – поверхня засклення, м^2;

$q_{\text{ост}}$– величина радіації через 1 м^2, ккал/м^2 год;

$A_{\text{ост}}$ – коефіцієнт, що залежить від характеристики засклення.

$$Q_3 = 3 \cdot 18 \cdot 0,7 = 37,8\ (\frac{\text{ккал}}{\text{год}})$$

Розрахуємо тепловиділення від працюючих за формулою:

$$Q_4 = q_1 \cdot n\ (\frac{\text{ккал}}{\text{год}}) \qquad \text{(В.6)}$$

де q_1 – тепловиділення від одної людини в залежності від праці, що виконується, Вт;

n - кількість працюючих (3 людини).

Відповідно ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень» приміщення з ПК відноситься до категорії робіт по важкості до категорії I-а (легка) q = (90 – 120 ккал/год)

$$Q_4 = 120 \cdot 3\ =\ 360\ (\frac{\text{ккал}}{\text{год}})$$

$$Q_{\text{надл}} = 31{,}7 + 500 + 37{,}8 + 360 = 929{,}5\ (\frac{\text{ккал}}{\text{год}})$$

Тоді L дорівнює:

$$L = \frac{929{,}5}{0{,}24 \cdot 1{,}2 \cdot (25 - 21)} = 806{,}8\ (\frac{\text{м}^3}{\text{год}}) \approx 1000\ \left(\frac{\text{м}^3}{\text{год}}\right)$$

Таким чином, обираємо кондиціонер ALASCA R-1000SE, який забезпечує регулювання температури повітря (L = 1000 м^3/год), вологості та повітрообміну згідно з ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень» для кімнати площею 20 м^2.

В.3 Розрахунок освітлення у приміщенні

Одним з основних питань охорони праці є організація раціонального освітлення виробничих приміщень робочих місць.

Для освітлення приміщення, у якому працюють фахівці, використається змішане освітлення, тобто сполучення природного й штучного освітлення. Природне освітлення здійснюється через вікно. Штучне освітлення використовується при недостатньому природному освітленні й здійснюється за допомогою двох систем: загального й місцевого освітлення.

Розрахунок системи освітлення виробляється методом коефіцієнта використання світлового потоку, що виражається відношенням світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп.

Його величина залежить від характеристик світильника, розмірів приміщення, фарбування стін і стелі, яка характеризується коефіцієнтами відбиття стін і стелі.

Світловий потік при загальному освітленні визначається за формулою:

$$\Phi_{\text{заг}} = E_{\text{н}} \cdot S \cdot k \cdot z / \eta \qquad \text{(В.7)}$$

де Е – нормована освітленість робочого місця по нормі, лк (Е=400 лк);

S – площа приміщення, м^2;

k – коефіцієнт запасу, що враховує зношування й забруднення світильників (k = 1,2);

Z – коефіцієнт, що враховує нерівномірність висвітлення (Z = 1,1);

η – коефіцієнт використовування світлового потоку, що вибирається з таблиць залежно від типу світильника, розмірів приміщення, коефіцієнта відбиття стін і стелі приміщення.

Оберемо з таблиці коефіцієнт використання світлового потоку за наступним даними:

– коефіцієнт відбиття для стелі приймемо $\rho_п = 60\%$;

– коефіцієнт відбиття стін $\rho_з = 30\%$;

$$i = \frac{A \cdot B}{H_c(A+B)} \quad \text{(В.8)}$$

де А і В – сторони приміщення, м;

H_c – розрахункова висота підвісу світильника, м.

Розрахункова висота визначається за формулою:

$$H_c = H - h_c - h_p \quad \text{(В.9)}$$

де Н – висота приміщення, м;

h_c – відстань світильників від перекриття (h_c=0,1 м);

h_p – висота робочої поверхні над підлогою (h_p=0,8 м).

Таким чином, розрахункова висота дорівнює:

$$H_c = 3{,}5 - 0{,}1 - 0{,}8 = 2{,}6 \text{ (м)}$$

Підставляючи значення у формулу (В.9), одержимо:

$$i = \frac{5 \cdot 5}{2{,}6\ (5+5)} = 0{,}962$$

Тоді визначаємо значення коефіцієнта використання світлового потоку $\eta = 0{,}45$.

Загальний світловий потік дорівнює:

$$Ф_{заг} = \frac{400 \cdot 25 \cdot 1{,}3 \cdot 1{,}1}{0{,}45} = 31777 \text{лм}$$

Найбільш прийнятними для приміщення є люмінесцентні лампи ЛБ (білого світла), потужністю 80 Вт які мають потік $Ф_1 = 4320$ лм. Кількість ламп дорівнює:

$$N = \frac{\Phi_{заг}}{\Phi_1} \text{ (шт)} \qquad \text{(В.10)}$$

Підставляючи значення, отримуємо:

$$N = \frac{31777}{4320} = 7{,}35 = 8 \text{ (шт)}$$

Якщо у світильники встановити по 2 лампи 80 Вт, з потоком 4320 лм, то потрібно розташувати 4 світильники на однаковій відстані до центру кімнати (близько 2 м).