**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»**

**Спеціальність: 123 «Комп'ютерна інженерія»**

**Освітня програма: «Комп'ютерна інженерія»**

**Група: 2БКС-28**

# КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

**здобувача освіти денної форми навчання**

**БКС.28.04.000.КРБ**

***ГАЛІТИ***
***ОЛЕГА СЕРГІЙОВИЧА***

**м. Одеса**
**2024 р.**

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітньо-професійна програма: **«Комп'ютерна інженерія»**

Група: **2БКС-28**

# ПОЯСНЮВАЛЬНА ЗАПИСКА

До кваліфікаційної роботи бакалавра на тему: **«Аналіз ефективності алгоритмів розпаралелювання матричних операцій на базі MPI»**

Проектний матеріал складається з пояснювальної записки на 88 сторінках та графічного (презентаційного) матеріалу на 23 аркушах (слайдах)

Виконавець _____ ( Галіта О.С.)

Керівник проекту _____ ( Кривченко А.А.)

**Консультанти:**

з розділу охорони праці та техніки безпеки _____ ( Чорновол Н.І.)

з нормоконтролю _____ ( Петрашова В.І.)

старший консультант _____ ( Кривченко Ю.В.)

**До захисту допущений**

Завідувач кафедри _____ ( Іванова Л.В.)

Завідувач відділення _____ ( Скорнякова О.В.)

Захист «24» 06 2024 р. Протокол ЕК № 1

Оцінка ЕК 5 (відмінно) 95

Секретар ЕК _____

# АНОТАЦІЯ

У випускній кваліфікаційній роботі розглянуто застосування обчислювального кластеру для скорочення часу обчислення матричних операцій на базі технології MPI. Досліджено ефективність застосування обчислювального кластеру та проаналізовано ефективність алгоритмів розпаралелювання матричних операцій на базі MPI.

Проведено аналіз апаратного та програмного забезпечення для високопродуктивних обчислень, розглянуто три типи систем: багатоядерну, розподілену та з графічним адаптером.

Реалізовано побудову та описано організацію обчислювального кластеру для матричних операцій на основі наявного обладнання відповідно до технічного завдання.

Вирішено декілька задач із застосуванням матричних операцій, що використовують паралельні алгоритми для обчислень.

За побудованими паралельними алгоритмами для цих задач та відповідними алгоритмами виконання потоків реалізовано програмне забезпечення мовою програмування C++ на базі технології MPI.

Протестовано розроблені програмні засоби та досліджено ефективність застосування обчислювального кластеру у тестових задачах для відповідних алгоритмів розпаралелювання матричних операцій на базі MPI.

Описано заходи з охорони праці та техніки безпеки.

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»**

Відділення *Комп'ютерних систем* Кафедра *Комп'ютерної інженерії*

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"15" 07 2024 р.

# ЗАВДАННЯ

## на кваліфікаційну роботу бакалавра

здобувачеві освіти *Галіті Олегу Сергійовичу*

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Аналіз ефективності алгоритмів розпаралелювання матричних операцій на базі MPI*

затверджена наказом по коледжу від "02" 11 2023 р. № 244-А2-ОД

2. Термін здачі студентом кваліфікаційної роботи 13.06.2024

3. Вихідні дані до роботи *1. Специфікації технології MPI;*
*2. Обчислювальний кластер має складатись не менш ніж з 4-х комп'ютерів;*
*3. Тестовий стенд – на основі багатоядерних процесорів Intel або AMD;*
*4. Для реалізації багатопотокових застосунків застосовувати мову програмування C++;*
*5. Тестувати ефективність матричних операцій не менш ніж для 3 задач*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
*1. Огляд технологій розпаралелювання обчислень;*
*2. Аналіз можливостей ПЗ для розпаралелювання обчислень;*
*3. Створення обчислювального кластеру та застосування технології MPI;*
*4. Аналіз ефективності алгоритмів розпаралелювання у тестових задачах;*
*5. Питання охорони праці та техніки безпеки*

5. Перелік графічного матеріалу (слайдів мультимедійної презентації) *Архітектура сучасних процесорів Intel та AMD; Топології розподілених систем; Схематичне зображення чіпів CPU і GPU; Основні функції MPI; Схеми взаємодії задач; Схема топології ПАК; Схема структурна процесора; Схема взаємодії потоків у задачах; Час виконання задач; Значення КП для задач; Графік залежності коефіцієнту прискорення від кількості процесорів для задач; Значення КЕ для задач; Графік залежності коефіцієнту ефективності від кількості процесорів для задач; Завантаження ядр процесора при тестуванні*

6. Консультанти по кваліфікаційній роботі, із зазначенням розділів, що їх стосуються

| Розділ | Консультант | ПІДПИС | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| Основний розділ | *Кривченко А.А.* | | |
| Розділ охорони праці | *Чорновол Н.І.* | | |
| Нормоконтроль | *Петрашова В.І.* | | |
| Старший консультант | *Кривченко Ю.В.* | | |

7. Дата видачі завдання 15.01.2024

Керівник роботи *Кривченко А.А.* _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

## КАЛЕНДАРНИЙ ПЛАН

| Пор. № | Назва етапів кваліфікаційної роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| *1.* | *Вступ. Аналіз технічного завдання* | 08.05.24 | виконано |
| *2.* | *Огляд технологій розпаралелювання обчислень* | 09.05.24 | виконано |
| *3.* | *Аналіз сучасних багатоядерних систем* | 11.05.24 | виконано |
| *4.* | *Основні можливості систем з графічним адаптером* | 13.05.24 | виконано |
| *5.* | *Аналіз засобів паралельного програмування* | 19.05.24 | виконано |
| *6.* | *Вивчення можливостей технології MPI* | 25.05.24 | виконано |
| *7.* | *Вивчення можливостей технології PVM* | 28.05.24 | виконано |
| *8.* | *Дослідження технології CUDA та OpenCL* | 30.05.24 | виконано |
| *9.* | *Вибір апаратного забезпечення* | 01.06.24 | виконано |
| *10.* | *Вибір програмних засобів розробки* | 02.06.24 | виконано |
| *11.* | *Розробка тестових програм мовою C++* | 04.06.24 | виконано |
| *12.* | *Тестування продуктивності обчислень* | 06.06.24 | виконано |
| *13.* | *Аналіз результатів тестування* | 08.06.24 | виконано |
| *14.* | *Побудова графіків залежностей* | 10.06.24 | виконано |
| *15.* | *Розробка питань з охорони праці* | 11.06.24 | виконано |
| *16.* | *Оформлення слайдів презентації* | 12.06.24 | виконано |

Здобувач освіти _____
(підпис)

Керівник роботи _____
(підпис)

| Формат | Зона | Поз. | Позначення | Назва | Кіл. | Примітка |
|---|---|---|---|---|---|---|
| | | | | Документація | | |
| | | | БКС 28. 04 000. 00 КРБ | Випускна робота | | |
| А4 | | | БКС 28. 04 000. 00 КРБ ПЗ | Пояснювальна записка | 1 | |

| Зм. | Арк. | № докум. | Підпис | Дата | БКС 28. 04 000. 00 КРБ | | |
|---|---|---|---|---|---|---|---|
| Розробив | | Галіта О.С. | | 10.06 | Аналіз ефективності алгоритмів розпаралелювання матричних операцій на базі MPI | Літ. | Аркуш 4 / Аркушів 88 |
| Перевірив | | Кривченко А.А. | | 10.06 | | Н Д П | |
| Н. Контр. | | Петрашова В.І. | | 10.06 | | ВСП "ОТФК ОНТУ" гр.2БКС-28 | |
| Затверд. | | Іванова Л.В. | | 10.06 | | | |

# ЗМІСТ

# ВСТУП

Відомо, це пришвидшення здійснення послідовності дій поза рахунок розпаралелювання її інструкцій в безлічі рахувальних пристроїв обмежене періодом, необхідним задля здійснення її послідовних інструкцій [1].

Загальна проблема структурних дій, зокрема множення структур, полягає в через це, це вони являється надто ресурсомісткими, потребують здійснення великої чисельності арифметичних дій. У деяких задачах 1 комп'ютер почне декілька днів виконувати підрахунок, через це задля подібних обчислень використовують суперкомп'ютери і комп'ютерні кластери, значно потужніші поза звичайні PC. Задля повного застосування них обчислювальної потужності треба розробляти спеціальні багатопотокові послідовності дій.

Протягом останнього десятиріччя набуло поширення застосування графічних адаптерів задля здійснення арифметичних обчислень. Них застосування задля структурних дій являється доволі перспективним із точки зору пришвидшення обчислень, оскільки в відеоадаптері звичайно являється в десятки разів більше рахувальних ядер, ніж в центральному процесорі. Пристрої задля паралельних векторних обчислень, використовуваних в 3D-графіці, досягають високої пікової продуктивності, що універсальним процесорам не під силу. Звичайно, максимальна потужність досягається виключно в ряді зручних здач та містить деякі обмеження, проте такі пристрої вже почали досить широко застосовувати в сферах, задля яких вони спочатку та не призначалися.

Проте модулю графічного процесору не спроможні одноперіодно виконувати різні дії, зокрема додавання та множення – одноперіодно буде виконуватись виключно одна дія над різними даними в кожному ядрі. Задля структурних дій такий спосіб обчислень не являється недоліком, оскільки над кожним елементом матриці містить виконатись одна та і ж арифметична дія.

Метою даної дії являється аналізування потужності алгоритмів розпаралелювання структурних дій в базі технології MPI, що дає можливість скоротити період підрахунок структурних дій.

# 1 ОСНОВНИЙ РОЗДІЛ

## 1.1 Огляд рахувальних структур в багатоядерних ЦП

Комп'ютерні структури багатоядерного виду будуються в базі обчислювачів, котрі містять більше єдиного комп'ютерного модулю в своїй архітектурі. В сьогоднішній день число ядер в процесорах поступово збільшується біля 64. Існує 3 можливі варіанти багатоядерності:

1. Незалежні процесорні модулю, кожне із своєю кеш-пам'яттю, розташовані в одному кристалі та використовують спільну шину. Приклад виконання – процесор Pentium Д в ядрі Smithfield;
2. Кілька однакових ядер, розташованих в різних кристалах, об'єднані разом в спільному корпусі обчислювача (багаточіповий процесор). Приклад виконання – процесори сімейств Pentium та Xeon в модулюх Presler та Dempsey;
3. Модулю розташовані в одному кристалі та спільно використовують деякі ресурси кристала (шину, кеш-пам'ять). Такими являється майже усі суперіодні багатоядерні процесори.

Поміж модулюми спроможні застосовуватися наступні способи зв'язку:

- шина;
- мережа (Mesh) в каналах точка-точка;
- мережа із комутатором;
- спільна кеш-пам'ять.

Обраний спосіб зв'язку поміж модулюми впливає в потужність передавання значень із єдиного модулю біля іншого. Програмісту, це створює паралельну програму під конкретну систему, задля виконання ефективного алгоритму слід добре знати особливості її архітектури.

Біля 2017 року використовувалась кільцева (ring) структура (рис.1.1). В рис. 1.2 представлена масштабована структура Mesh (сітка), що застосовується в процесорах Intel із 2017 року [1].

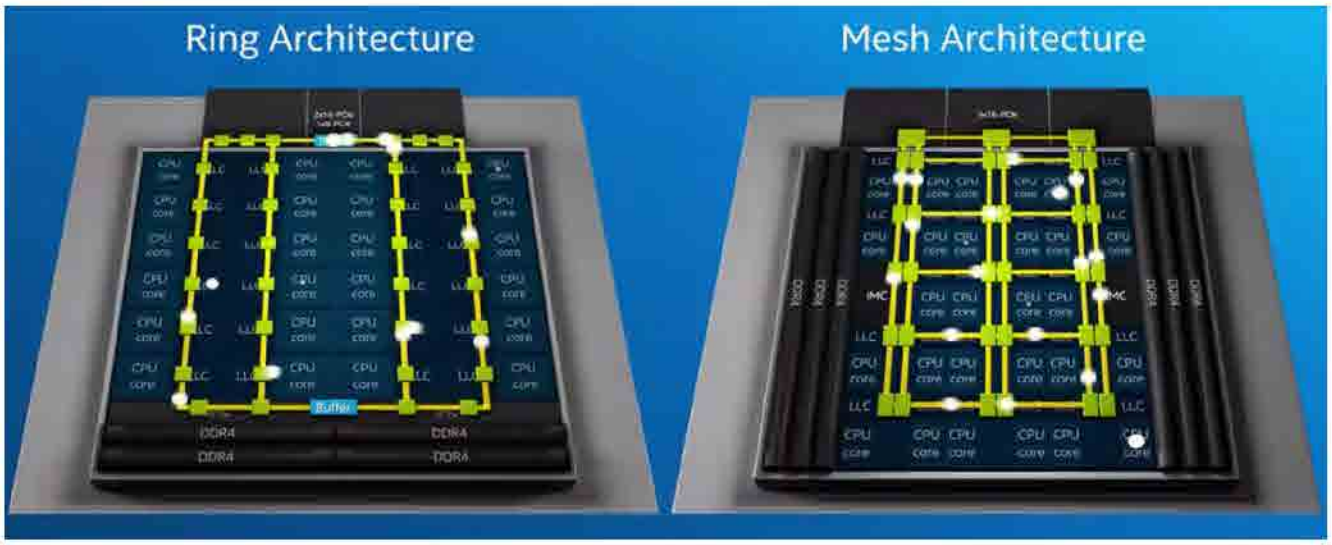



Рисунок 1.1. Порівняння архітектур Intel Ring і Mesh

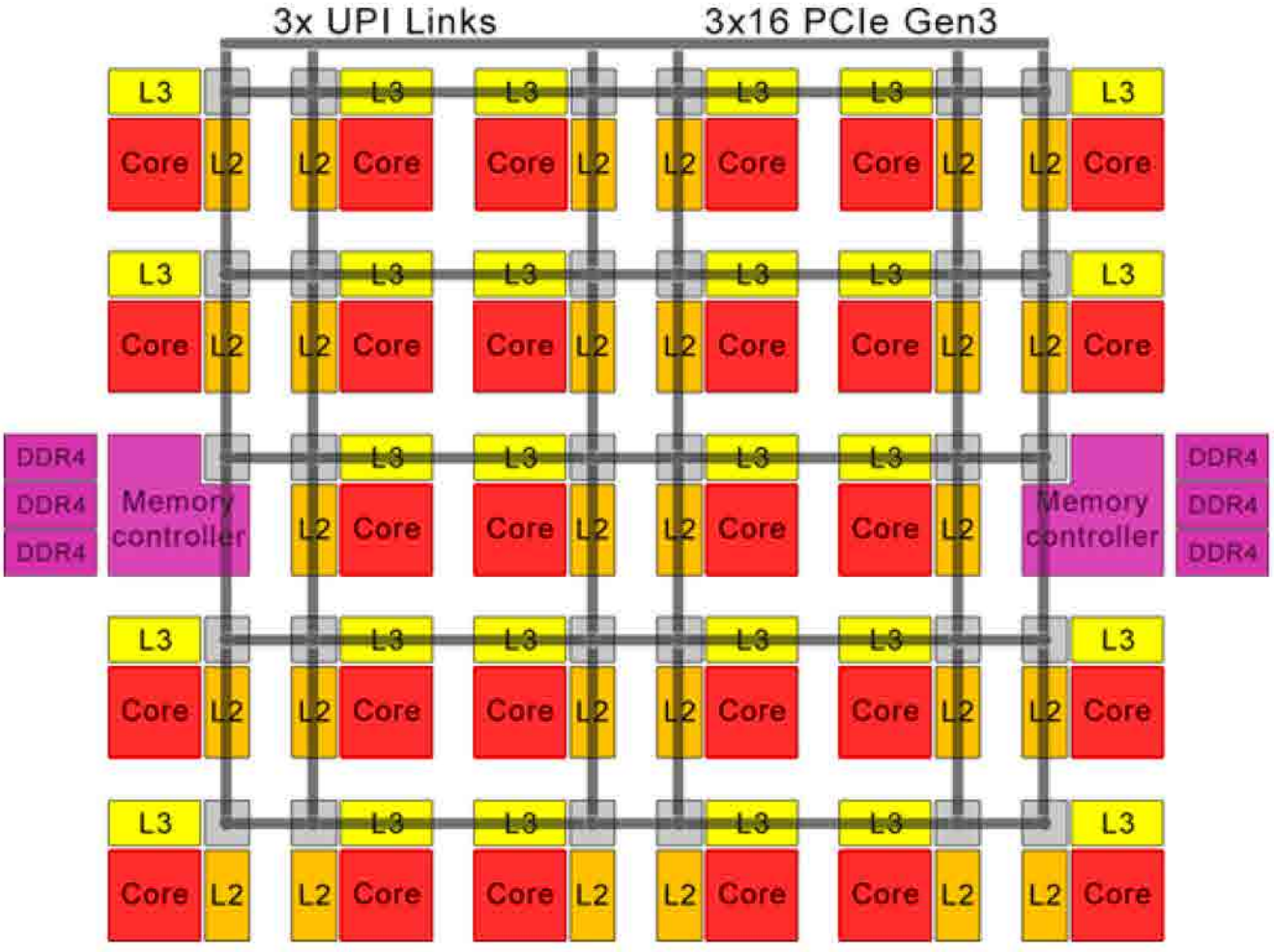



Рисунок 1.2. Структура Intel Mesh

Повнозв'язний комплекс із чотирьох ядер (CCX), це застосовується в процесорах AMD із архітектурою Zen [2], показаний в рис 1.3. Пам'ять L3 являється спільною задля всіх ядер комплексу, проте вона розділена в рівні періодтини задля

будь-якого модулю. Якщо ядру не вистачає своєї періодтини памяті, воно буде використовувати будь-яку L3-память в рамках CCX. Згідно біля архітектури Zen2 кожен кристал складається із двох таких комплексів, проте ядро включає в себе 1 чи кілька таких кристалів.

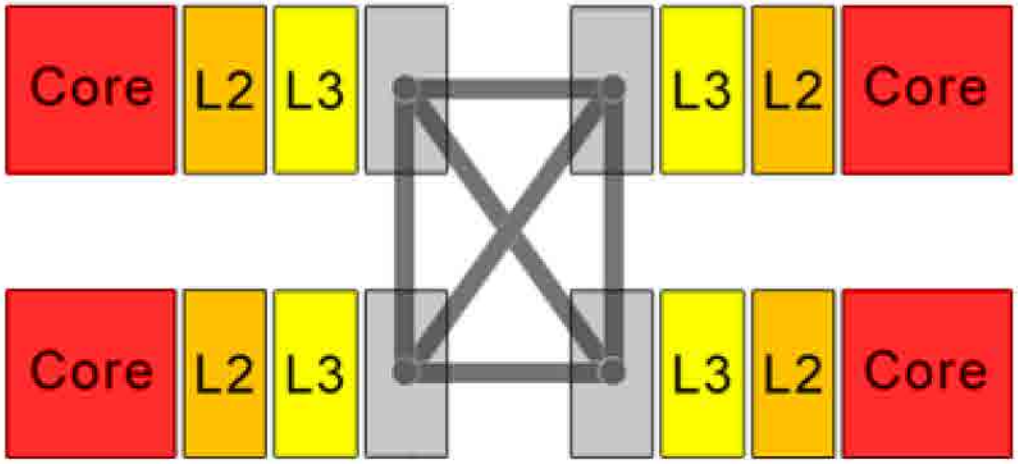



Рисунок 1.3. Структура Core Complex обчислювачів AMD Zen 2

В сьогодні задля багатьох виробників обчислювачів, зокрема Intel, AMD, IBM, ARM, збільшення числа ядер являється одним із пріоритетних напрямків збільшення продуктивності обчислювачів, адже інші напрямки гальмуються нинішнім рівнем розвитку науки та технологій. В таблицях 1.1 – 1.4 наведено властивості деяких багатоядерних обчислювачів задля PC із компаній Intel і AMD.

Таблиця 1.1. Деякі властивості обчислювачів Intel Core X

| *Модель процесору* | *i9-10980XE* | *i9-9960X* | *i9-9980XE* | *i9-7960X* | *i9-7980XE* |
|---|---|---|---|---|---|
| *Число ядер* | *18* | *16* | *18* | *16* | *18* |
| *Число тредів* | *36* | *32* | *36* | *32* | *36* |
| *Тактовий період обчислювача* | *3.0 ГГц/ 4.6 ГГц* | *3.1 ГГц/ 4.4 ГГц* | *3.0 ГГц/ 4.4 ГГц* | *2.8 ГГц/ 4.2 ГГц* | *2.6 ГГц/ 4.2 ГГц* |
| *Кеш-пам'ять* | *24.75 MB Intel® Smart Cache* | *22.00 MB Intel® Smart Cache* | *24.75 MB Intel® Smart Cache* | *22 MB* | *24.75 MB* |
| *Період системної шини* | *8 GT/s* | *8 GT/s* | *8 GT/s* | *8 GT/s* | *8 GT/s* |

Таблиця l.2. Деякі властивості обчислювачів Intel Core i9

| *Модель процесору* | *i9-9900KS* | *i9-9900T* | *i9-9900* | *i9-9900K* |
|---|---|---|---|---|
| *Число ядер* | *8* | *8* | *8* | *8* |
| *Число тредів* | *l6* | *l6* | *l6* | *l6* |
| *Тактова періодтота обчислювача* | *4.00 ГГц/ 5.00 ГГц* | *2.l0 ГГц/ 4.40 ГГц* | *3.l0 ГГц/ 5.00 ГГц* | *3.60 ГГц/ 5.00 ГГц* |
| *Кеш-пам'ять* | *l6 MB Intel® Smart Cache* | *l6 MB Intel® Smart Cache* | *l6 MB Intel® Smart Cache* | *l6 MB Intel® Smart Cache* |
| *Періодтота системної шини* | *8 GT/s* | *8 GT/s* | *8 GT/s* | *8 GT/s* |

Таблиця l.3. Деякі властивості обчислювачів AMD Threadripper 3rd Gen

| *Модель процесору* | *3960X* | *3970X* | *3990X* |
|---|---|---|---|
| *Число ядер* | *24* | *32* | *64* |
| *Число тредів* | *48* | *64* | *l28* |
| *Тактова періодтота обчислювача* | *3.80 ГГц/ 4.50 ГГц* | *3.70 ГГц/ 4.50 ГГц* | *2.90 ГГц/ 4.30 ГГц* |
| *Кеш-пам'ять* | *l40 MB* | *l44 MB* | *288 MB* |

Таблиця l.4. Деякі властивості обчислювачів AMD Threadripper 2rd Gen

| *Модель процесору* | *2950X* | *2970WX* | *2990WX* |
|---|---|---|---|
| *Число ядер* | *l6* | *24* | *32* |
| *Число тредів* | *32* | *48* | *64* |
| *Тактова періодтота обчислювача* | *3.50 ГГц/ 4.40 ГГц* | *3.00 ГГц/ 4.20 ГГц* | *3.00 ГГц/ 4.20 ГГц* |
| *Кеш-пам'ять* | *40 MB* | *80 MB* | *80 MB* |

Задля коректного порівняння перелічених в таблицях обчислювачів компаній Intel і AMD треба пам'ятати, це вони націлені в різні сегменти ринку обчислювачів задля PC. Одними із головних показників продуктивності обчислювача являється того тактова системної шини (чим вища– тим вища потужність передавання значень). Період ядер обчислювача показує, із якою швидкістю модулю виконують дії над даними. Проте процесор із більшою кількістю ядер та із меншою тактовою

період буде існувати продуктивнішим, ніж процесор із меншою кількістю ядер в більшою тактовою періодичністю. У виконанні громіздких обчислень важливу роль грає об'єм кеш-памяті. Дії запису-читання біля кеш-памяті обчислювача виконуються швидше, ніж дії запису-читання в оперативну пам'ять.

## І.2 Огляд рахувальних структур розподіленого виду

Комп'ютерні структури розподіленого виду представляють собою набір незалежних комп'ютерів, котрий задля користувачів виглядає єдиною системою. У цьому користувачам не відомі відмінності поміж комп'ютерами та способи зв'язку поміж ними. Структури зі спільною пам'яттю називаються мультипроцесорами (multiprocessors), проте із розподіленою пам'яттю – мультикомп'ютерами (multicomputer). Типовий приклад такого мультикомп'ютера – кілька персональних комп'ютерів, об'єднаних в єдину локальну мережу. Типовий приклад розподіленої структури – мережа інтернет [9].

Загальні властивості конфігурації мережі передавання значень являється такими:

- число вузів/обчислювачів (P);
- діаметр (Д) – відстань поміж найвіддаленішими вузлами мережі. Ця величина буде охарактеризувати максимально-необхідний період задля передавання значень поміж вузлами/процесорами, оскільки період передавання зазвичай прямо пропорційний довжині шляху;
- зв'язність (S) – число зав'язків біля єдиного вузла;
- вартість (R) – загальна число зв'язків (ліній передавання значень) в мережі.

Далі розглянуто конфігурації і властивості розподілених структур.

### І.2.1 Лінійна конфігурація

Конфігурація лінія набула популярності завдяки своїй низькій вартості та легкості розширення мережі. Проте вона містить 1 важливий недолік: фізичне пошкодження виключно єдиного кабелю буде вивести період структури із ладу. Структура конфігурації "Лінія" наведена в рис.І.4.

Число обчислювачів: $P = 4;$

Зв'язність: $S \leq 2;$

Діаметр: $D = P - 1 = 3;$

Вартість: $R = P - 1 = 3.$

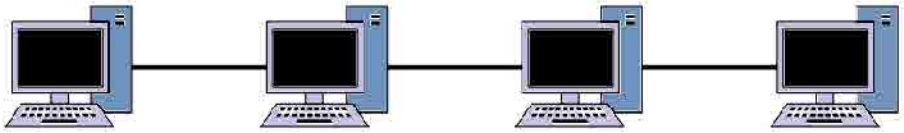


Рисунок 1.4. Лінійна конфігурація

### 1.2.2 Конфігурація "Кільце"

Біля переваг конфігурації кільце можна теж віднести низьку вартість і легкість розширення, простоту установки. В такій структурі шлях передавання значень поміж найвіддаленішими вузлами скорочується вдвічі. Серед недоліків: вразливість біля фізичних пошкоджень кабелю. Структура конфігурації "Кільце" наведена в рис.1.5.

Число обчислювачів: $P = 4;$

Зв'язність: $S = 2;$

Діаметр: $D = \lfloor P/2 \rfloor = 2;$

Вартість: $R = P = 4.$

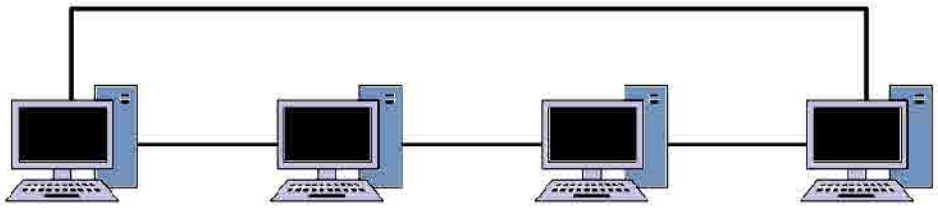


Рисунок 1.5. Конфігурація "кільце"

### 1.2.3 Конфігурація "Зірка"

В конфігурації "Зірка" у виході із ладу єдиного кабелю з'єднання обірветься виключно із одним користувачем, що наслідок – перевагою конфігурації являється

простий пошук несправностей та обривів. Серед переваг разом з цим простота підключення нових PC. Недоліки: несправність центрального вузла виведе із ладу всю систему; велика довжина сполучних кабелів; у великій чисельності підключених PC в центральний вузол почне припадати великий трафік та він почне виконувати виключно процедуру передавання сповіщень поміж вузлами, проте не обчислень, через це в такій структурі почне доцільніше замінити центральний PC в комутатор. Структура конфігурації “Зірка” наведена в рис.1.6.

Число обчислювачів: $P = 5$;

Зв’язність: $S \leq P - 1 = 4$;

Діаметр: $D = 2$;

Вартість: $R = P - 1 = 4$.

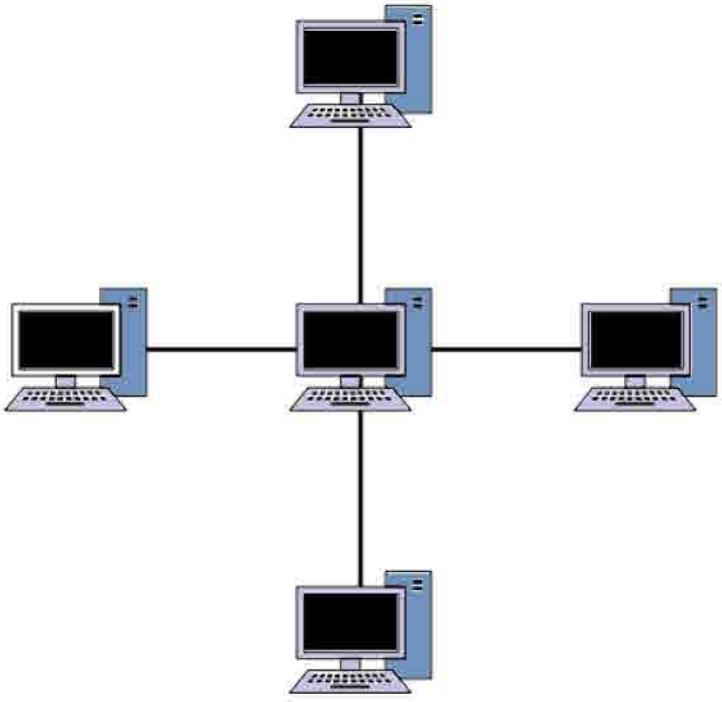


Рисунок 1.6. Конфігурація “зірка”

### 1.2.4 Конфігурація “Решітка”

Конфігурація “решітка” складається із n1 x n2 робочих станцій. Така система являється доволі надійною в порівнянні із попередніми. В ній легко шукати несправності. Застосовується оптимальна число з’єднувальних кабелів. Серед недоліків: розширення структури можливе виключно квантами розміру n1 чи n2.

Структура конфігурації “Решітка” наведена в рис.1.7.

Число обчислювачів: $P = 9;$

Зв’язність: $S \leq 4;$

Діаметр: $D = 2(\sqrt{P} - 1) = 4;$

Вартість: $R = 2(P - \sqrt{P}) = 12.$

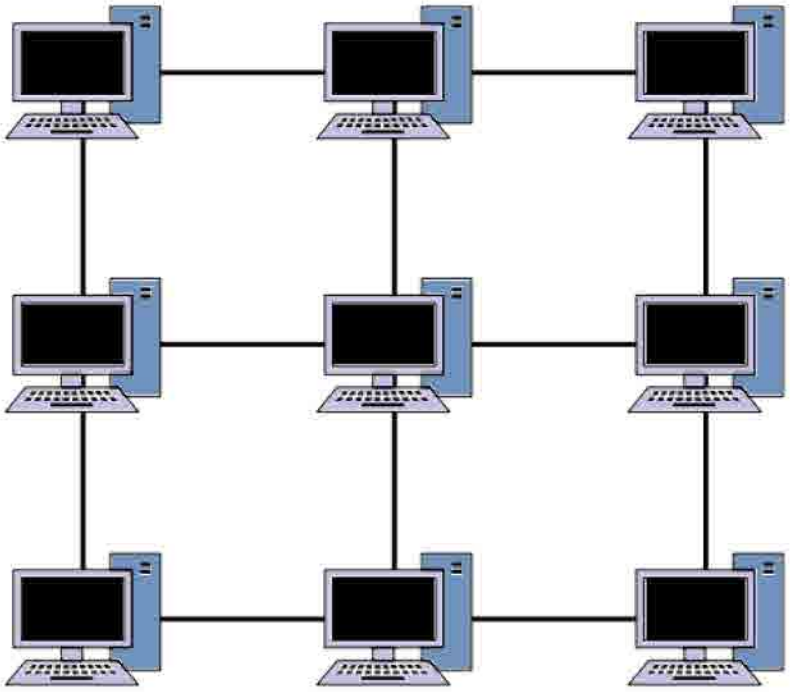


Рисунок 1.7. Конфігурація “решітка”

### 1.2.5 Конфігурація “Тор”

Конфігурація “тор” дуже схожа в решітку, поза винятком додаткових зв’язків, через це містить ідентичні переваги і недоліки. Структура конфігурації “Тор” наведена в рис.1.8.

Число обчислювачів: $P = 9;$

Зв’язність: $S = 4;$

Діаметр: $D = 2 * \lfloor \sqrt{P}/2 \rfloor = 2;$

Вартість: $R = 2 * P = 18.$

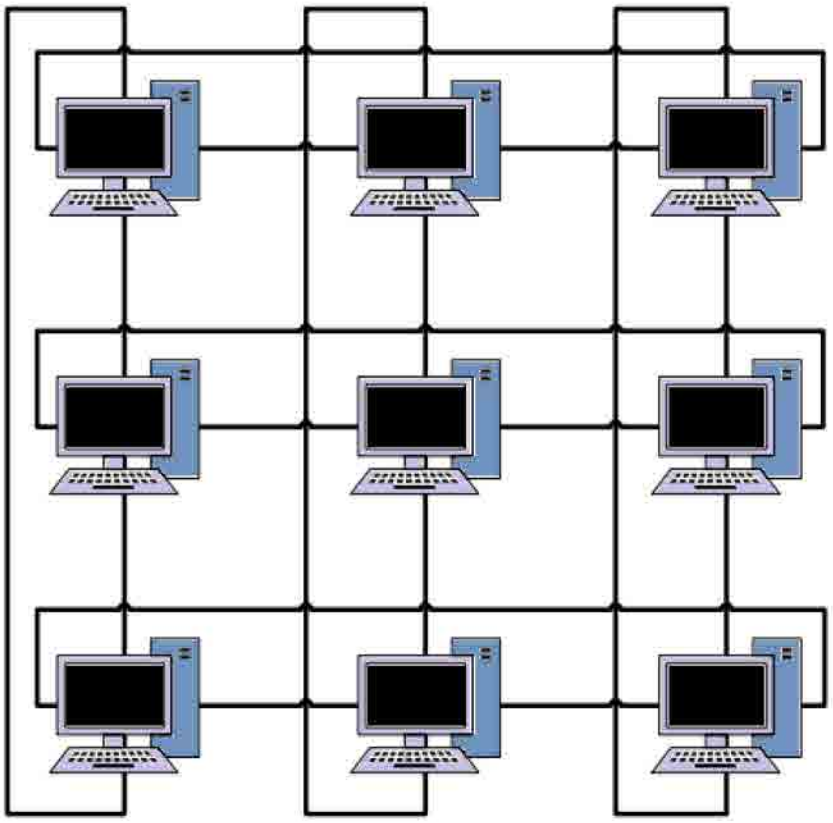


Рисунок 1.8. Конфігурація “тор”

### 1.2.6 Повнозв’язна конфігурація

Єдиним плюсом повнозв’язної конфігурації являється її діаметр: кожен вузол напряму з’єднаний із усіма іншими. Не зважаючи в свою логічну простоту, такий варіант з’єднання являється громіздким і неефективним. Виникають труднощі у підключенні нового вузла. Найчастіше така конфігурація застосовується задля з’єднання невеликої чисельності робочих станцій. Структура повнозв’язної конфігурації наведена в рис.1.9.

Число обчислювачів: $P = 4;$

Зв’язність: $S = P - 1 = 3;$

Діаметр: $D = 1;$

Вартість: $R = \frac{P(P-1)}{2} = 6$

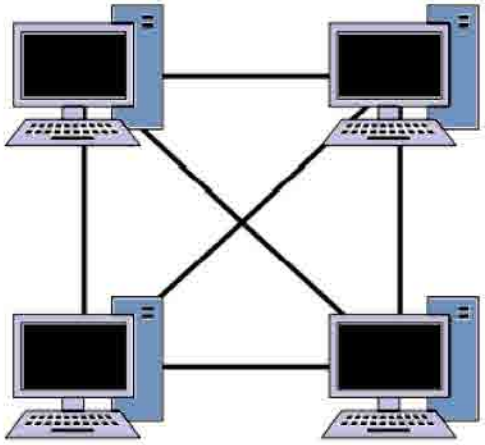


Рисунок 1.9. Повнозв'язна конфігурація

### 1.2.7 Конфігурація "Гіперкуб"

Структура "гіперкуб" дає малу число зв'язків поміж процесорами. Така конфігурація являється популярною у об'єднанні паралельних обчислювачів. Серед недоліків –великі затрати у розширенні структури: із кожним порядком число зв'язків та зв'язків подвоюється. Структура конфігурації "Гіперкуб" наведена в рис.1.10.

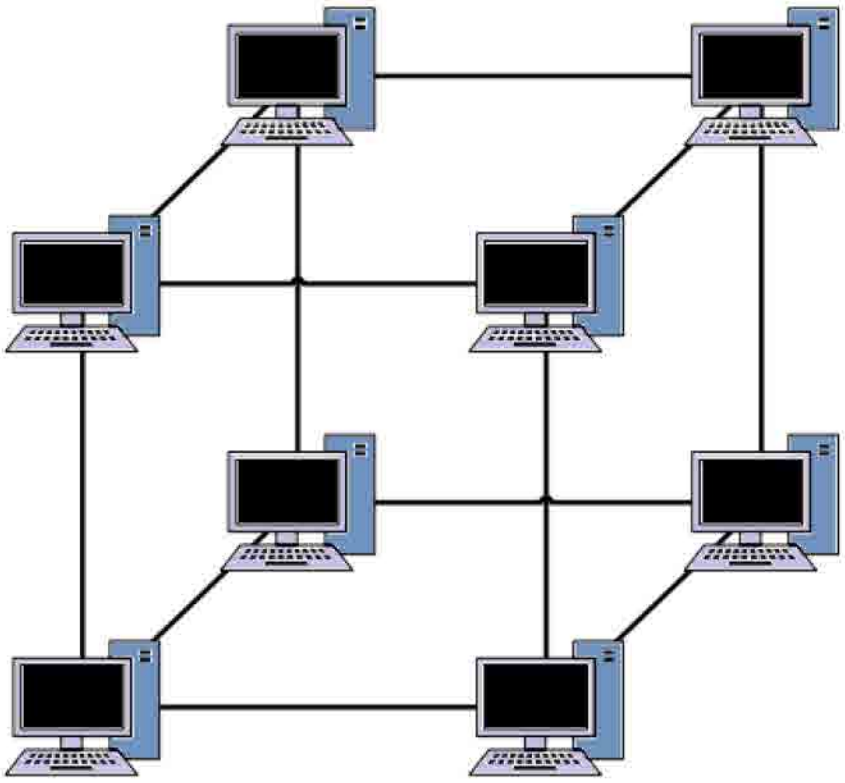


Рисунок 1.10. Конфігурація “гіперкуб“

Число обчислювачів: $P = 8;$

Зв’язність: $S = \log_2 P = 3;$

Діаметр: $D = \log_2 P = 3;$

Вартість: $R = \frac{P(\log_2 P)}{2} = 12$

### 1.2.8 Основні властивості розподілених структур

Основними характеристиками розподілених структур являється:

- спільне застосування ресурсів – зокрема жорстких дисків, принтерів, файлів, компіляторів, зв'язаних поза сприянням мережі;
- відкритість – можливість розширювати систему шляхом додавання нових ресурсів (апаратного та програмного утворення із різних виробників);
- паралельність – можливість одноперіодного здійснення кількох операцій в різних комп'ютерах в мережі;
- масштабованість – можливість нарощувати систему поза сприянням додавання нових рахувальних ресурсів;

– відмовостійкість – стійкість біля певних апаратних та програмних помилок. Більшість розподілених структур в разі помилки, що правило, спроможні підтримувати хоча б періодткову функціональність. Повний збій в роботі структури відбувається виключно в разі мережевих помилок;
– прозорість – означає, це користувачам надано повністю прозорий доступ біля ресурсів та в той же період із них прихована інформація про розподіл ресурсів в структурі.

В розподілених структур являється такі недоліки:

– складність в проектуванні, побудові та відлагодженні них дії;
– відмова компонентів структури. ІЗ періодом деякі компоненти зношуються та відмовляють в роботі. Це призводить біля перерозподілу завдань в вузлах і змінення шляхів передавання значень, через це деякі ділянки такої структури спроможні виявитися перевантаженими, проте інші – простоювати;
– змінення в конфігурації. Розподілена система підлягає еволюції, отримує нові комп'ютерні вузли. Це призводить біля тих же проблем, це та відмова вузла в відлагодженій структурі.

## І.3 Огляд структури із прискоренням в базі відео-адаптеру

Відео-адаптер РС містить в собі графічний процесор – пристрій, призначений задля здійснення дій із обробки графіки та обчислень із плаваючою крапкою.

В основі високої продуктивності графічних обчислювачів (GPU) лежить них здатність виконувати програмний код паралельно в великій чисельності однорідних рахувальних елементів зі спільною пам'яттю. Подібна структура називається SIMT – Single Instruction Multiple Threads: код однаковий, проте оброблювані дані – різні [13]. Кожен обчислювальний елемент здатний виконувати тисячі тредів, перемикання поміж якими не містить накладних витрат. Треди спроможні існувати згруповані в блоки, котрі містять загальний кеш та швидку пам'ять, це розділяється, та явно контролюється користувачем.

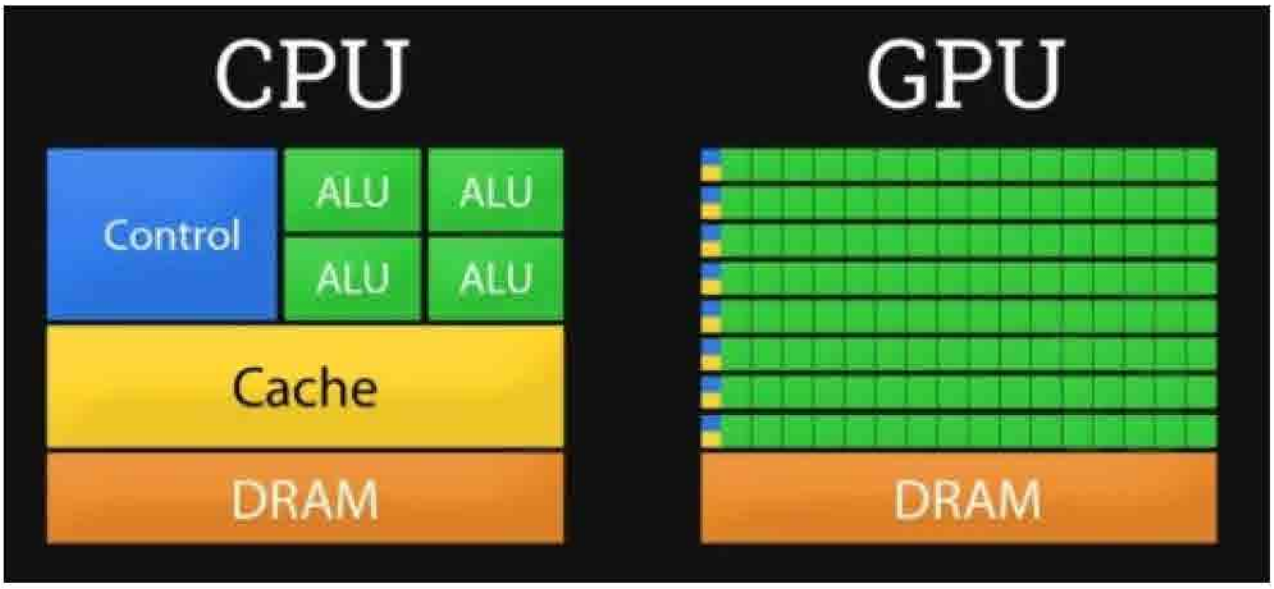



Рисунок 1.11. Розташування логічних блоків в CPU та GPU

В CPU велика період площі чіпу зайнята під буфери команд, апаратне передбачення розгалуження та величезні обсяги кеш-памяті, проте в GPU велика період площі зайнята виконавчими блоками. Схематичне зображення чіпів CPU та GPU показане в рис. 1.11.

Властивості відеоадаптерів, це впливають в них продуктивність, являється такими:

- пропускна здатність відео-пам'яті;
- тип відео-пам'яті (GDDR4, GDDR5, GDDR6, HBM, HBM2 і інші) показує біля якого покоління належить пам'ять GPU. Кожне наступне покоління являється досконалішим поза попереднє та забезпечує більш високу періоду;
- об'єм відео-пам'яті. Якщо того недостатньо, тоді відеокарта починає використовувати оперативну пам'ять комп'ютера, доступ біля якої займає значно більше періоду. Проте правило «чим більше – тим краще» в цьому випадку теж не працює. Треба щоб усі властивості були збалансованими;
- властивості графічного модулю: тактова частота, число шейдерних обчислювачів. Тут вже діє правило «чим більше – тим краще».

В ринку із виробництва дискретних графічних обчислювачів задля

відеокарт фігурують перш поза все дві компанії – NVIDIA та AMD.

Розглянемо деякі популярні рішення виробників [6].

Таблиця 1.5. Властивості відеокарт в GPU із AMD

| *Відеокарта* | *ASUS ROG STRIX (RX5700XT-8G)* | *ASRock Phantom Gaming X Radeon VII* | *SAPPHIRE RADEON RX 5700 XT Nitro+* |
|---|---|---|---|
| *GPU* | *RADEON RX 5700 XT* | *RADEON VII* | *RADEON RX 5700 XT* |
| *Періодтота GPU* | *1840/2035 МГц* | *1400/1750 МГц* | *1840/2035 МГц* |
| *Число шей-дерних обчислювачів* | *2560* | *3840* | *2560* |
| *Пропускна здатність памяті* | *1024 Gb/s* | *1024 Gb/s* | *1024 Gb/s* |
| *Тип відео-пам'яті* | *GDDR6* | *HBM2* | *GDDR6* |
| *Об'єм відео-пам'яті* | *8 Gb* | *16 Gb* | *8 Gb* |
| *Розрядність шини відео-пам'яті* | *256 bit* | *4096 bit* | *256 bit* |
| *Періодтота відео-пам'яті* | *3500 MGz (14 ГГц QDR)* | *1000 МГц* | *3500 MGz (14 ГГц QDR)* |

Таблиця 1.6. Властивості відеокарт в GPU із NVIDIA

| *Відеокарта* | *MSI GAMING TRIO (RTX 2080 Ti)* | *GIGABYTE AORUS Waterforce Xtreme (GV-N208TAORUS)* | *ASUS ROG STRIX (RTX2080TI)* |
|---|---|---|---|
| *GPU* | *GeForce RTX 2080 Ti* | *GeForce RTX 2080 Ti* | *GeForce RTX 2080 Ti* |
| *Періодтота GPU* | *1350/1635 МГц* | *1770 МГц* | *1350/1665 МГц* |
| *Число шей-дерних обчислювачів* | *4352* | *4352* | *4352* |
| *Пропускна здатність памяті* | *616 Gb/s* | *616 Gb/s* | *616 Gb/s* |
| *Тип відео-пам'яті* | *GDDR6* | *GDDR6* | *GDDR6* |
| *Об'єм відео-пам'яті* | *11 Gb* | *11 Gb* | *11 Gb* |
| *Розрядність шини відео-пам'яті* | *352 bits* | *352 bits* | *352 bits* |
| *Періодтота відео-пам'яті* | *3500 МГц (14 ГГц QDR)* | *3535 МГц (14.14 ГГц QDR)* | *3500 МГц (14 ГГц QDR)* |

В цілому, графічні рішення із компанії NVIDIA дещо швидше виконують

підрахунок, ніж графічні рішення AMD, містять менше енергоспоживання та незначне виділення тепла. Відеокарти із AMD містять більшу пропускну здатність памяті, проте споживають більше енергії, виділяють дещо більше тепла.

## І.4 Аналізування програмних засобів розпаралелювання обчислень в багатоядерних системах

Багато суперіодних мов кодування підтримують багатопотоковість та містять власні засоби виконання тредів. Мови високого рівня (Java, Python, .NET) надають багатопотоковість розробнику в вигляді абстрактної специфічної платформи, це відрізняється із виконання тредів в середовищі здійснення. Ряд інших мов кодування (бібліотек) разом з цим намагається повністю абстрагувати концепцію паралелізму та потоковості із розробника (Cilk, OpenMP, MPI). Бібліотеки WinAPI, PThread, OpenMP надають розробникові інтерфейс задля виконання тредів.

### І.4.1 Синхронізація тредів в системі кодування Java

В системі кодування Java задля синхронізації тредів застосовується високорівневий механізм моніторів. Методи класів, це містять модифікатор synchronized виконуються в режимі взаємного виключення. Одноперіодно із єдиного класу буде виконуватися виключно 1 синхронізований метод. Інкапсуляція спільного ресурсу в такому класі-моніторі виконується поза сприянням модифікатора private. Ніякий інший тред не буде увійти в блок, захищений монітором, поки перший тред не вийде із synchronized-блоку. Якщо інші треди викликають синхронізований метод, це вже виконується, вони почнуть блоковані біля завершення здійснення цього синхронізованого методу.

```
public class Monitor {
private Object obj = new Object();
public void doSmth () {
//... логіка, доступна задля всіх тредів
synchronized (obj) {
//логіка, одноперіодно доступна виключно задля єдиного треду
```

```
}
}
}
```

Якщо змінна являється спільним ресурсом, оголошеним поза сприянням модифікатора private чи являється volatile-змінною, дії читання-запису такої змінної являється атомарними. Доступи біля volatile-змінних впорядковані глобально, тобто кожен тред, котрий звертається біля volatile-змінної, прочитає заново її результат перед тим, що продовжити здійснення послідовності дій, замість того, щоб (по можливості) використати кешоване результат.

### 1.4.2 Треди в системі C# задля паралельного кодування

Суперіодна мова кодування C# забезпечує кодування операцій через треди. Програма мовою C# завжди являється багатопотоковою, адже здійснення послідовності дій розпочинається із виконання нового треду (“main”-треду) та породження додаткових тредів.

В системі кодування C# задля виконання треду використовують клас Thread. Через конструктор класу передається метод, котрий почне виконуватись потоком (потоковий метод). Метод join() дає можливість організувати очікування завершення викликаного треду. Механізм семафорів в системі C# реалізований поза сприянням класу Semaphore, котрий містить 4 конструктори. Семафор створюється що об’єкт класу Semaphore:

*Semaphore S = new Semaphore(p1, p2),*

тут p1 – початкове результат, p2 – максимальне результат семафору.

Методи S.WaitOne(), S.Resume() аналогічні операціям P(S), V(S). Задля семафору визначено 2l метод. Мютекс створюється що об’єкт класу Mutex (існує 5 конструкторів класу).

*Mutex M = new Mutex(false),*

тут false – початковий момент мютексу.

Методи M.WaitOne(), M.ResumeMutex() аналогічні операціям P(M), V(M).

Задля мютексів визначено 21 метод [9].

Найпростішим та фундаментальним об'єктом задля синхронізації тредів являється подія. Події із ручним скиданням реалізовані поза сприянням класу ManualResetEvent.

*ManualResetEvent E = new ManualResetEvent(false);*

Початкове результат події передається в конструктор. Методи E.WaitOne(), E.Set() аналогічні операціям P(E), V(E). являється разом з цим класи AutoResetEvent і EventWaitHandle [10].

В системі C# механізм критичних секцій реалізовано поза сприянням

*lock(Object O){ <блок набоору команд> }*

Здійснення блоку набоору команд, описаного в тілі, дозволяється виключно одному набору дій. Якщо процес виконує оператор lock, тоді він почне заблокований в випадку, якщо інший процес вже розпочав здійснення блоку набоору команд із оператора lock із однаковим параметром Object. Інша реалізація – поза сприянням класу Monitor, в якому являється методи Monitor.Enter(Object Mon), Monitor.Exit(Object Mon) [7].

### 1.4.3 Робота із процесами засобами бібліотеки WinAPI

Біля складу операційної структури Windows входить бібліотека Win32 (Windows API), це містить набір функцій, котрі призначені задля дії із процесами. Задля виконання набору дій в WinAPI використовують процедуру CreateThread(). Процедура обертає ідентифікатор набору дій, котрий являється унікальним, та визначає того в структурі. Під період виконання набору дій визначається початковий адрес набоору команд, із якого містить виконуватись тред. Зазвичай, це назва процедури (потокової процедури), що вже створена та почне виконуватись що процес:

*HANDLE Ім'я_Треду = CreateThread(*
*LPSECURITY_ATTRIBUTES amp, // атрибут безпеки*
*SIZE_T pc, // розмір стека*

*LPTHREAD_STERT_ROUTUNE фп, // процедура треду*

*LPVOID афп, // аргумент процедури треду*

*DWORD пр, // прапорець здійснення*

*LPDWORD іп); // ідентифікатор треду*

Зазначена процедура створює тред, задля якого фактичні параметри визначають ім'я потокової процедури і її параметри, атрибут безпеки, початковий розмір стека треду, прапорець здійснення: негайного (0) чи відкладеного (Create_Suspended), ім'я треду. Тред виконується доти, доки не відпочнеться одна із таких подій:

- процедура обертає результат треду;
- тред викликає процедуру ExitThread();
- інший тред викликає процедуру ExitProcess();
- інший тред викликає процедуру TerminateThread() із дескриптором треду;
- інший тред викликає процедуру TerminateProcess() із дескриптором набору дій.

Поки тред не закінчить роботу та усі того дескриптори не почнуть закритими поза сприянням процедури CloseHandle(), він залишається в структурі [5]. В бібліотеці Win32 семафори являється змінними спеціального виду HANDLE, поза якими слідкує сама система. В цій бібліотеці лічильники семафорів разом з цим являється багатозначними.

Задля семафорів визначені наступні дії.

1) дія, це створює семафор:

*HANDLE CreateSemaphore (LPSECURITY_ATTRIBUTES lpSemaphore Attributes, LONG lInitialCount, LONG lMaximumCount, LPCTSTR lpName),*

- lpSemaphoreAttributes – параметри захисту, зазвичай встановлено результат 0;
- lInitialCount – початкове результат, яке містить існувати більше поза 0 чи рівним 0, і меншим поза lMaximumCount чи рівним йому. Момент семафору сигнальний, якщо це число більше поза 0, і не сигнальний, якщо

рівне 0. Результат семафору зменшується в l, якщо процедура очікування розблоковує тред, котрий чекав в цей семафор. Результат семафору збільшується в визначене результат викликанням процедури ReleaseSemaphore;

- lMaximumCount – максимальне результат, яке містить існувати більше поза 0;
- lpName – ім'я семафору, яке містить існувати не більше поза MAX_PATH і буде містити в собі будь-котрі символи, окрім зворотного слешу(\);
- BOOL ReleaseSemaphore (HANDLE hSemaphore, LONG lReleaseCount, LPLONG lpPreviousCount) – реалізація дії V(S);
- hSemaphore – семафор, котрий повертається функцією CreateSemaphore;
- lReleaseCount – число, в яке того потрібно збільшити. Це число повинне існувати більше поза нуль. Якщо додавання цього числа біля результат, яке вже являється, зробить того більше поза максимальне результат семафору, тоді додавання не відпочнеться та процедура поверне FALSE;
- lpPreviousCount – попереднє результат семафору. Буде існувати 0, якщо попереднє результат не потребується. Якщо процедура успішна, тоді результат, яке вона обертає, являється ненульовим.

Порівняння імен регістрозалежне. Якщо lpName співпадає із вже існуючим семафором, тоді lInitialCount і lMaximumCount ігноруються, через це це вони вже були встановлені у створенні. Якщо lpName являється 0, тоді семафор створюється без імені. Якщо процедура успішна, тоді вона обертає семафор виду HANDLE. Якщо іменований об'єкт семафору існував біля виклику процедури, тоді процедура GetLastError обертає ERROR_ALREADY_EXISTS. В інших випадках GetLastError обертає нуль. Якщо CreateSemaphore являється невдалою, тоді обертає 0. Задля поширених відомостей про помилку потрібно викликати процедуру GetLastError.

2) дія, це створює мютекс:

*HANDLE CreateMutex(LPSECURITY_ATTRIBUTES lpMutexAttributes, BOOL bInitialOwner, LPCTSTR lpName)*, тут:

– lpMutexAttributes – атрибут безпеки;

– bInitialOwner – початковий момент (прапор початкового власника);

– lpName – ім'я об'єкту [12].

Подія – це виключно прапор, якому процедурами SetEvent / ResetEvent можна задати момент: сигналізуючий чи нейтральний.

3) дія, це створює подію:

*HANDLE CreateEvent (LPSECURITY_ATTRIBUTES lpEventAttributes, BOOL bManualReset, bInitialState, LPCTSTR lpName)*, тут:

– lpEventAttributes – атрибут безпеки;

– bManualReset – тип скидання: TRUE – ручний;

– bInitialState – початковий момент: TRUE – сигнальний;

– lpName – ім'я об'єкту.

У параметрі bInitialState=FALSE почне створено подію із автоскиданням. Це означає, це що виключно якийсь тред, це очікує цієї події, почне звільнений сигналом із цієї події, вона автоматично почне скинута назад в нейтральний момент [6]. Задля дії із критичною секцією потрібна змінна виду критичної секції (CRITICAL_SECTION). Біля того, що почати використовувати критичну секцію, її треба ініціалізувати:

*VOID InitializeCriticalSection(LPCRITICAL_SECTION lpCriticalSection).*

Задля оголошення початку критичної секції застосовується процедура:

*VOID EnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection).*

Задля виходу із критичної секції застосовується процедура:

*VOID EnterCriticalSection(LPCRITICAL_SECTION lpCriticalSection),*

тут lpCriticalSection – вказівник в змінну виду CRITICAL_SECTION.

Класи CLock та CAutoLock зручно використовувати задля синхронізації доступу біля змінних класу, проте CScopeLock призначений задля застосування в процедурах. Компілятор сам подбає про виклик :: LeaveCriticalSection() через деструктор.

4) реалізація дії P(S):

*HANDLE WaitForSingleObject (HANDLE hHandle, DWORD dwMilliseconds)*,

тут:

- hHandle – ідентифікатор об'єкта;
- dwMilliseconds – визначає період тайм-ауту в мілісекундах. Процедура обертає результат якщо інтервал періоду скінчується, навіть якщо момент семафору несигнальний. Якщо dwMilliseconds дорівнює нулю, тоді процедура визначає момент семафору і повертається миттєво. Якщо dwMilliseconds дорівнює INFINITE, таймаут процедури не наступає.

У успішному виконанні процедури результат, це вона обертає, визначає, що подія змусила її повернутися. ЯВЛЯЄТЬСЯ два такі результат: WAIT_OBJECT_0, яке повертається якщо момент семафору стає сигнальним, і WAIT_TIMEOUT, яке повертається, якщо трапляється тайм-аут і момент семафору являється несигнальним. WAIT_FAILED вказує в неуспішне здійснення процедури. Очікування в некоректний семафор змушує процедуру повернути WAIT_FAILED. Задля поширених відомостей про помилку потрібно викликати процедуру GetLastError.

5) реалізація дії P(S), якщо треба чекати в кілька об'єктів).

## 1.5 Аналізування програмних засобів розпаралелювання обчислень в розподілених системах

Розглянемо основні проблеми виконання програмного утворення задля розподілених структур:

- затримки. Дані із різних рахувальних зв'язків приходять не одноперіодно та в різному порядку;
- відмова компонентів структури у збереженні працездатності структури в цілому. Можливі випадки, якщо 1 тред нескінченно довго чекає відповіді із іншого, нестабільного треду;
- виникнення «вузьких місць»;
- змінення в конфігурації. Розподілена система підлягає еволюції. Це буде впливати в пропускну здатність, викликати проблеми із затримками;

– гетерогенність. Структура зв'язків структури буде використовувати різні конфігурації, механізми, програмне утворення.

### 1.5.1 Бібліотека Message Passing Interface (MPI)

Інтерфейс MPI містить набір функцій, це дозволять організувати роботу із процесами в мовах, котрі не містять вбудованих засобів кодування операцій. Число використовуваних обчислювачів буде існувати задана вручну у запуску через командний рядок, проте буде визначатися в момент запуску послідовності дій засобами MPI-середовища. Основні процедури MPI, необхідні задля виконання операцій, наведені в табл. 1.7.

Таблиця 1.7. Основні процедури MPI задля виконання операцій

| *Процедури* | *Дії* |
|---|---|
| *MPI Init()* | Підключити біля MPI (параметри args та argv визначають аргументи послідовності дій). Завжди викликається першою. |
| *MPI Comm size()* | Визначити число операцій, котрі потрібно запустити |
| *MPI Comm rank()* | Отримати ранг (номер) набору дій в групі (комунікаторі) |
| *MPI Finalize()* | Завершити здійснення послідовності дій |

В MPI процес називається задачею. Усі завдання послідовно пронумеровані, починаючи із нуля. Номер завдання разом з цим називають рангом. Завдання в MPI спроможні існувати об'єднані в іменовану групу. Об'єкт-група дає можливість звертатися біля групи що єдиного цілого (зокрема, відправляти повідомлення всім задачам групи), проте разом з цим визначати дії, котрі виконуються виключно членами групи.

Усі завдання в рамках групи містять унікальні ідентифікатори (ранги) – впорядковані числа, котрі розпочинаються із нуля. Спочатку усі завдання належать біля однієї базової групи, із якої потім формуються нові групи – MPI надає набір функцій задля дії із групами. Комунікатор – опис ділянки зв'язку операцій, котрі об'єднані в групу [10]. Програма буде вміщувати кілька ділянок зв'язку. Нумерація операцій всередині ділянки зв'язку незалежна. Комунікатор буде існувати

використаний що параметр MPI-процедури задля обмеження сфери її дії виключно заданою ділянкою зв'язку. Крім того, комунікатор забезпечує вимоги безпеки. MPI автоматично створює комунікатор MPI_COMM_WORLD, котрий являється базовим задля будь-якого додатку та створюється автоматично під період виклику процедури MPI.Init().

*int MPI_Comm_size(); int MPI_Comm_rank();* – задля комунікатора повертають результат: size – розміру групи (число завдань, це приєднані біля ділянки зв'язку) та rank – порядковий номер завдання, що викликає цю процедуру. Одною із основних переваг в MPI являється процес передавання і прийому сповіщень. Процеси в межах єдиного комунікатора спроможні посилати і одержувати повідомлення. Передавання сповіщень відбувається поза сприянням процедури: *MPI_Comm comm);* – процедура виконує передачу count елементів виду dtype повідомлення із ідентифікатором tag набору дій destrank в області зв'язку комунікатора comm. Прийом сповіщень відбувається поза сприянням процедури:

*int MPI_Recv(void* buf, int count, MPI_Datatype dtype, int sourcerank, int tag,*

*MPI_Comm comm, MPI_Status *status);* – процедура виконує получення count елементів виду dtype повідомлення із ідентифікатором tag із набору дій sourcerank в області зв'язку комунікатора comm.

### 1.5.2 Бібліотека Paralel Virtual Machine (PVM)

Бібліотека PVM являється однією із перших структур кодування, це базується в механізмі передавання сповіщень та отримала широке поширення. Вона проектувалася задля того, щоб зв'язати окремі незалежні комп'ютери в віртуальну обчислювальну систему, що була б єдиним керованим обчислювальним ресурсом. Бібліотека PVM орієнтована в роботу із неоднорідними системами із розподіленою пам'яттю. В її основу покладена концепція віртуальної паралельної обчислювальної машини, що об'єднує в єдиний ресурс безліч різнорідних зв'язків. Цікаво, це віртуальна машина допускає динамічне конфігурування (причому користувачем, проте не адміністратором). ЯВЛЯЄТЬСЯ виконання PVM задля всіх основних операційних структур та підтримується широкий набір мов кодування –

Fortran, C / C ++, Tcl / Tk, Perl, Python [9].

PVM складається із двох періодів: PVM-сервера (pvmd) та призначених задля користувача бібліотек (libpvm3.a libfpvm3.a). Сервер pvmd забезпечує комунікації поміж комп'ютерами та управління процесами. В кожному комп'ютері віртуальної паралельної машини запускається 1 pvmd-сервер. Перший pvmd-сервер, це запускається користувачем, стає майстер-процесом, в той період що усі інші pvmd-процеси, це запускаються майстром, називаються робітниками. PVM-бібліотеки дозволяють процесам-робітникам взаємодіяти із pvmd-серверами в інших вузлах. Вони містять процедури задля упакування та розпакування сповіщень та процедури задля передавання сповіщень. Зокрема, процедура pvm_mytid() обертає ідентифікатор набору дій, із якого її викликано.

*pvm_spawn(char *task, char **argv, int flag, char *where, int ntask, int *tids)* – запускає новий процес. В PVM-середовищі задля передавання значень поміж процесами використовуються буфери сповіщень. Кожен процес буде мати 1 чи декілька таких буферів, проте виключно 1 із них являється активним. Перед відправкою будь-якого повідомлення викликається процедура, що дає можливість підготувати чи сформувати активний буфер сповіщень:

*pvm_initsend (int encoding).*

Задля упаковки масиву заданого виду значень в активний буфер відправлення використовуються процедури pvm_pk*. Зокрема:

*pvm_pkfloat( float *fp, int nitem, int stride)*

Задля відправки та получення сповіщень використовуються процедури send та recv, відповідно.

*pvm_send(int tid, int msgtag) pvm_recv(int tid, int msgtag)*

Виймають отримане повідомлення із активного буферу сповіщень та розпаковують того, зберігаючи в масиві вказаного виду, процедури pvm_upk*. Зокрема:

*pvm_upkfloat( float *fp, int nitem, int stride)*

Процедура pvm_exit() викликається у завершенні здійснення набору дій Це дає можливість із'єднати того із PVM-середовища, проте не просто завершити

процес.

### 1.5.3 Реалізацію співдружності завдань в системі ADA

Реалізацію співдружності завдань в системі Ада через посилання сповіщень визначає механізм Randevu. Оператори входу entry, прийняття виклику входу accept, оператор відбору select дозволяють ефективно реалізувати різноманітні можливості моделі Randevu. Треба звернути увагу в несиметричність такого механізму співдружності. Це означає, це в процесі співдружності одна із завдань розглядається що сервер, проте інша – що клієнт, причому задача-сервер не буде існувати ініціатором початку співдружності.

Задача-клієнт та задача-сервер виконуються незалежно 1 із єдиного, через це немає ніякої гарантії. Через це, якщо задача-сервер опинилась в точці Randevu, проте у цьому немає жодного звернення біля входу (запиту в взаємодію), тоді вона повинна чекати появи такого звернення. Аналогічно, якщо задача-клієнт звертається біля входу, проте задача-сервер не готова обслужити таке звернення, тоді задача-клієнт містить чекати, поки задача-сервер обслужить це звернення. В процесі очікування, що задача-клієнт, так та задача-сервер не займають ресурси обчислювача, перебуваючи в стані, котрий називають станом блокування [10].

Розглянемо приклад застосування механізму рандеву задля організації співдружності завдань T1 та T2, тут задача T1 передає в задачу T2 ціле число x. Задля співдружності завдань в специфікації завдання T2 описаний вхід DataIn, специфікація якого дає можливість приймати дані в завдання ВСтруктурну схему співдружності завдань, під період якої здійснюють обмін даними зображено в рис. 1.12. У цьому застосовано два входи: DataIn задля передавання значень та ResultOut задля повернення результату. Задля запуску завдань, них вкладено в процедуру Run_Tasks, виклик якої приведе біля старту завдань T1 та T2.

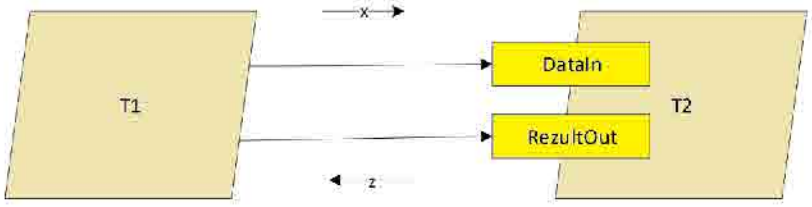


Рисунок 1.12. Двостороння схема співдружності завдань

Взаємодію трьох завдань T1, T2 та T3 зображено в структурній схемі (рис. 1.13). Задача T3 приймає данні із завдань T1 та T2 поза сприянням входів: DataInT1 та DataInT2. Особливістю виконання співдружності завдань являється застосування в тілі завдання T3 оператору select. Це дає можливість завдання T3 приймати виклик входів DataInT1 та DataInT2 в будь якій послідовності в міру них надходження [11].

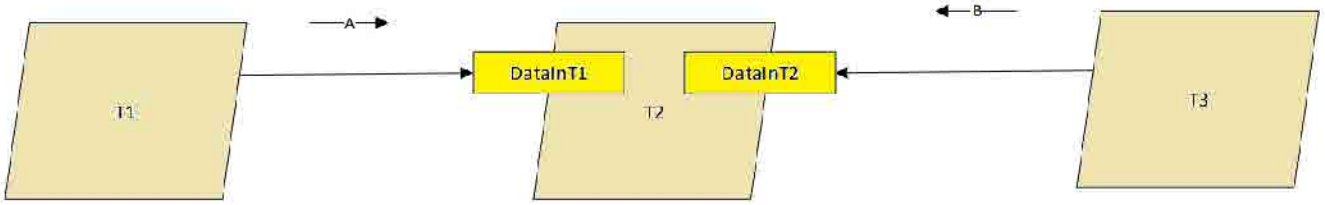



Рисунок 1.13. Структурна схема співдружності трьох завдань

## 1.6 Аналізування програмних засобів пришвидшення обчислень в системах із графічним адаптером

Застосування графічного обчислювача задля здійснення арифметичних дій потребує представлення вхідних значень в графічному вигляді та перетворення вихідних графічних значень в числові. Такі перетворення підтримуються двома програмними інтерфейсами задля графічних обчислювачів: OpenGL та DirectX. Ці громіздкі перетворення були усунені із появою мов кодування загального призначення і таких інтерфейсів що: Sh/RapidMind, Brook і Accelerator. Суперіодні GPU спроможні виконувати будь-котрі дії із великими даними та повноцінно використовувати свою потужність, не вимагаючи представлення значень в графічній формі. Суперіодні програмні інтерфейси задля дій GPGPU:

- OpenCL – відкритий стандарт задля розробки програм що в CPU, так та в GPU, розроблений в системі кодування C, апаратно та програмно незалежна платформа;
- DirectCompute – прикладна мова кодування, це являється періодтиною DirectX та підтримується починаючи із версії DirectX 10;
- C++ AMP – бібліотека, розроблена компанією Microsoft в системі C++, задля

дії необхідний DirectX ll;

- CUDA – фреймворк задля програмно-апаратних обчислень із компанії NVIDIA в системі C. Апаратно-залежна платформа, призначена виключно задля відеокарт із компанії NVIDIA;
- AMD FireStream – фреймворк задля програмно-апаратних обчислень із компанії AMD, апаратно незалежна платформа.

### 1.6.1 Пришвидшення обчислень поза технологією NVIDIA CUDA

В архітектурі CUDA послідовності дій пишуться «розширеною» мовою C, проте задля них компіляції застосовується спеціальний компілятор nvcc. Розширення мови включає специфікатори функцій (вони показують, тут почне виконуватися процедура та звідки буде існувати викликана), додаткові типи значень, специфікатори змінних (вони визначають, що пам'ять почне використовуватися задля них розміщення), синтаксис запуску ядер, вбудовані змінні, це зберігають інформацію про кожну нитку, проте разом з цим математичні процедури. Незважаючи в те, це поза поняттям thread в літературі закріпився переклад “тред”, тут (та в перекладній літературі по технології CUDA) застосовується переклад “нитка”, оскільки в рамках CUDA поняття “тред” (stream) разом з цим застосовується.

Виконувані багатьма потоками-нитками процедури почнуть позначені в програмі специфікаторами __global__ та спроможні існувати викликані в здійснення поза сприянням спеціального нового синтаксису:

```
<<. . . >>:
// визначення процедури-модулю
__global__ void MyKernel( . . . )
{
. . .
}
. . .
// виклик модулю в здійснення
```

*MyKernel<<M, N>>(...);*

. . .

В здійснення виклик ядер ініціюється в програмі вказанням них імені із так званими параметрами конфігурації запуску ядер (котрі вказуються в потрійних кутових дужках <<< ... >>>) поряд зі звичайними параметрами, необхідними задля дії будь-якого окремого модулю. Задля позначення того, це процедура являється ядром (тобто, повинна виконуватися в GPU та буде існувати запущена в здійснення із CPU), застосовується специфікатор __global__ . Специфікатори __host__ та __device__ позначають процедури, виконувані в CPU та GPU відповідно, причому викликані вони спроможні існувати виключно звідти, тут виконуються. Задля завдання розміщення в памяті GPU змінних використовуються специфікатори __device__ (змінна знаходиться в глобальній памяті та доступна всім ниткам), __constant__ (розміщена в так званій константної памяті, звідки вона буде існувати виключно прочитана будь-якою із ниток), __shared__ (змінна розміщена в розподіленій памяті GPU, тут доступна виключно всім ниткам «свого» блоку).

Окрім того, передбачені додаткові типи значень CUDA – 1/2/3/4-вимірні вектори із базових типів мови C (char, short, int, long - unsigned та signed, longlong, float та double), імена яких утворюються додаванням цифри із числом компонент: char1, char2, char3, char4. Звернення біля компонентів здійснюється поза іменами (x, y, z, w), проте виконання значень векторів – поза сприянням виклику функцій make_<TypeName>, тут <TypeName> – 1 із вищенаведених типів значень, зокрема:

*int2 a = make_int2(1,4);*

*float3 u = make_float3(1,2.72f,3.14f);*

Задля завдання розмірності являється тип dim3, котрий містить разом з цим нормальний конструктор:

*dim3 Blocks(16,16); // <=> Blocks(16,16);*

*dim3 Grid(256); // <=> Grid(256,1,1);*

Повний синтаксис запуску ядер в здійснення виглядає так:

*kernelName<<Dg, Db, Ns, S>>(args);*

тут kernelName – ім'я __global_-процедури, Dg – змінна виду dim3 (параметри

сітки блоків), Db – змінна виду dim3 (параметри блоку ниток), Ns – додатковий обсяг розподіленої памяті в байтах задля будь-якого блоку (необов'язковий параметр, котрий поза замовчуванням дорівнює нулю), S – задає тред (CUDAstream), в якому повинен відіснуватися виклик (тред 0 поза замовчуванням), args – параметри процедури-модулю (них буде існувати декілька, проте загальний розмір цих параметрів поки обмежений 256 байтами).

Існують такі способи виділення розподіленої памяті в ядрі:

- явно вказати розмір масиву зі специфікатором __shared__;
- у запуску модулю задати додатковий обсяг розподіленої памяті в байтах, котрий треба виділити кожному блоку (третій параметр конфігурації процедури- модулю). Задля доступу в модулюх біля такої памяті застосовується опис масиву в процедури модулю без явно заданого розміру, зокрема:

  *__shared__ float buf[];*

  Сам виклик модулю (із назвою kernel) у цьому виглядає так:

  *kernel<<< ... , ... , k*sizeof(float) >>>( ... );*

Процедура __syncthreads() – точка синхронізації, що працює що бар'єр, перед яким усі нитки блоку містять зупинитись в очікуванні завершення дії інших ниток блоку. Кожна копія процедури модулю після запуску в здійснення містить доступ біля спеціальних змінних, проте разом з цим біля положень конкретної копії модулю в блоці та цього блоку в усій сітці у виконанні набоору команд (threadIdx, blockIdx, обидві змінні містять тип int3). Використовуючи ці змінні, кожна копія модулю буде сформувати унікальні індекси задля доступу біля значень [ll].

### I.6.2 Пришвидшення обчислень поза технологією OpenCL

Open Computing Language (OpenCL) – відкритий стандарт задля паралельного кодування та дії із широким набором суперіодних паралельних обчислювачів (багатоядерних обчислювачів, GPU, FPGA). Спочатку OpenCL був запропонований фірмою Apple, проте згодом отримав підтримку багатьох представників галузі, в через це числі Intel, AMD, IBM, NVIDIA, ARM, Samsung та ін.

Програма, це використовує OpenCL, працює приблизно так: опитуються наявні комп'ютерні ресурси, обираються ті, це почнуть далі використовуватися, із вихідного набоору команд створюються комп'ютерні модулю та розподіляються задля запуску в комп'ютерні ресурси. Таким чином, розробка OpenCL-послідовності дій зводиться біля написання ядер та host-додатку задля PC, котрий розподіляє потрібні модулю в доступні пристрої. Такий додаток повинен використовувати п'ять структур: cl_device_id, cl_kernel, cl_program, cl_command_queue, cl_context. Він розподіляє модулю (cl_kernel), отримані із вихідного набоору команд (cl_program) в прилади (cl_device_id), ці модулю потрапляють в пристрої через чергу команд (cl_command_queue); контекст (cl_context) дає можливість пристроям отримувати модулю та обмінюватися даними. Подібний додаток містить отримати дані про пристрій, це почне далі виконувати процедуру-ядро, зокрема, що описано нижче.

Перша виявлена платформа:

*clGetPlatformIDs(l, &platform, 0);*

Перший пристрій (GPU):

*clGetDeviceIDs(platform, CL_DEVICE_TYPE_GPU, l, &device, 0);*

Після цього додаток створює контекст, зокрема, виключно із одним виявленим пристроєм:

*context = clCreateContext(0, l, &device, 0, 0, &err);*

тут &err – адреса змінної, в яку запишеться код помилки.

Додаток після цього повинен отримати програму із набоору команд модулю, це міститься, зокрема, в файлі hello.cl, задля чого вміст цього файлу зчитується в масив, котрий передається процедури:

*clCreateProgramWithSource:*

*program=clCreateProgramWithSource(context,l,(const char**)&*

*program_buffer, &program_size, &err);*

*clBuildProgram(program, 0, &device, 0, 0, 0);*

Параметри, відсутні в останньому виклику, спроможні визначати варіанти компіляції. Після неї із заданої процедури створюється ядро (тут "hello" – назва

процедури модулю):

*kernel = clCreateKernel(program, "hello", &err);*

Задля розподілу ядер в прилади треба створювати черги:

*queue = clCreateCommandQueue(context, device, 0, &err);*

Треба разом з цим створити буфер памяті, оскільки ядру знадобиться пам'ять задля виведення:

*mem = clCreateBuffer(context, CL_MEM_READ_WRITE,*

*MEM_SIZE * sizeof(char), 0, &ret);*

Після того, що усі компоненти оточення (тобто, структури cl_device_id, cl_kernel, cl_program, cl_command_queue, cl_context) створені, слід підготувати ядру необхідні параметри виклику, зокрема, в разі модулю hello – це 1 параметр (адреса буфера памяті задля виведення):

*ret = clSetKernelArg(kernel, 0, sizeof(cl_mem), (void *)&mem);*

Тепер можна відправляти ядро в чергу в здійснення:

*global_size = 8;*

*local_size = 4;*

*clEnqueueNDRangeKernel(queue, kernel, l, 0, &*

*global_size, &local_size, 0, 0, 0);*

Дана процедура не виключно забезпечує запуск модулю в пристрої, проте й визначає, що багато робочих одиниць містить існувати створено (параметр global_size), проте разом з цим скільки робочих одиниць почне в робочій групі (параметр local_size). Задля читання отриманих результатів із буфера памяті mem в масив (в даному випадку – символів) string викликається процедура clEnqueueReadBuffer:

*ret = clEnqueueReadBuffer(queue, mem, CL_TRUE, 0,*

*MEM_SIZE * sizeof(char), string, 0, 0, 0);*

Дана процедура обертає результат набоору команд можливої помилки (чи результат CL_SUCCESS) [9].

## 1.7 Вибір апаратних засобів задля комп'ютерного сукупності

### І.7.1 Визначення виду комп'ютерного сукупності

Проаналізуємо переваги та недоліки будь-якого із розглянутих типів комп'ютерних структур задля паралельних обчислень задля обрання виду створюваного комп'ютерного сукупності.

У виконанні зборки багатоядерної структури потрібен виключно 1 набір комплектуючих задля PC. Задля такої структури треба обрати потужний процесор. Розробка ПЗ задля такої структури почне простішою, ніж задля інших розглянутих типів комп'ютерних структур, проте її продуктивність почне обмежена продуктивністю виключно єдиного обчислювача.

Задля виконання комп'ютерного сукупності (розподіленої структури) треба зібрати декілька PC та обрати обладнання задля них із'єднання. Продуктивність цієї структури можна нарощувати із періодом підключенням нових зв'язків. У розробці ПЗ треба враховувати еволюційні змінення в конфігурації.

Задля обчислювальної структури із графічним адаптером треба обрати потужну відеокарту. Здійснення обчислень в відеокарті не означає, це CPU не почне брати уперіодть в обчисленнях, через це CPU треба обирати із відповідною продуктивністю.

ІЗ урахуванням описаних особливостей обрано розподілений тип комп'ютерного сукупності – в основі мережі декількох PC. Наразі існують інструменти API, це полегшують розробку програм задля розподілених комп'ютерних структур. Усі PC почнуть об'єднані в мережу із топологією зірка, тут роль центрального вузла почне виконувати комутатор (рис.1.14). Перевагою такої конфігурації являється те, це передачею сповіщень почне займатись виключно комутатор, проте комп'ютерні вузли не почнуть переривати підрахунок задля передавання повідомлення із єдиного іншому.

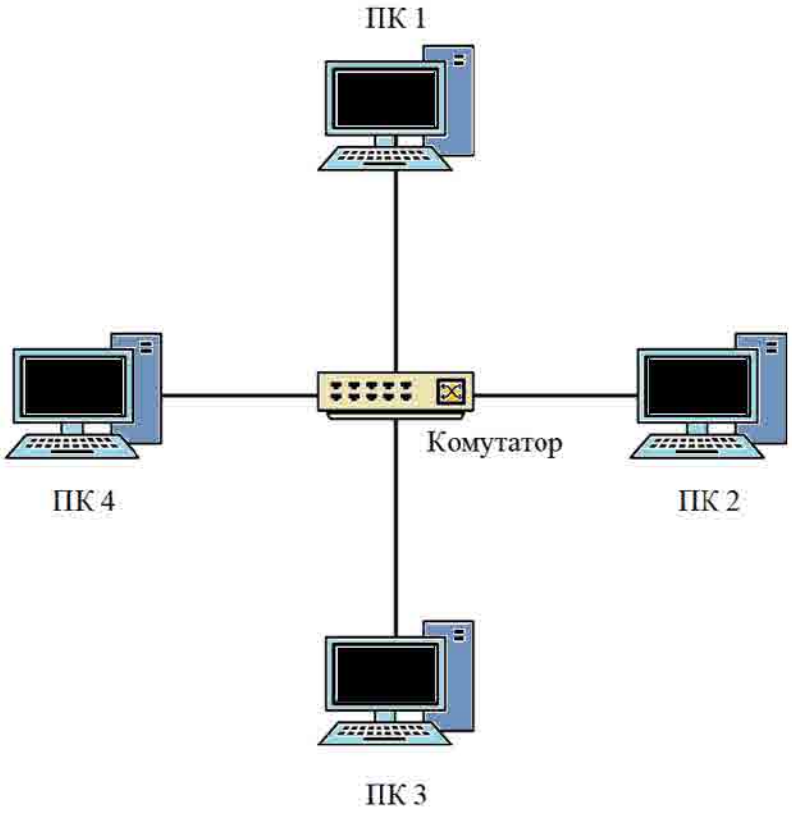



Рисунок 1.14. Схема конфігурації комп'ютерного сукупності задля структурних дій

### 1.7.2 Вибір комп'ютерних складових задля комп'ютерного сукупності

Обчислювальна система почне побудована в базі 16-ядерного процесору AMD Ryzen Threadripper 2rd Gen 2950X. Процесор містить 4 канали памяті, через це обрано 4 планки ОЗП по 8Гб кожна.

В процесорі AMD Ryzen Threadripper відсутній вбудований відеоадаптер, через це треба обрати дискретну відеокарту. Задля зазначених потреб обрано одну із найпростіших і найдешевших суперіодних відеокарт GeForce GT 710. В якості материнської плати обрано Gigabyte X399 Aorus Pro поза необхідним сокетом Socket sTR4 центрального обчислювача. В якості накопичувача обрано SSD-накопичувач із інтерфейсом M.2, це підтримується материнською платою, адже потужність передавання значень інтерфейсу M.2 являється більшою, ніж в звичного інтерфейсу SATA.

Процесори із AMD містять значне тепловиділення під період дії, проте даний екземпляр обчислювача Threadripper 2950X поза специфікацією виробника містить TDP 250 Вт, через це систему охолодження треба обрати відповідну. Рідинне

охолодження являється ефективнішим поза повітряне, через це обрано одну із рекомендованих компанією AMD рідинних структур задля цього обчислювача: ID-COOLING Zoomflow 240. Блок живлення підібрано необхідної потужності (табл.1.8). Теоретично можна обрати будь-яку число персональних комп'ютерів задля об'єднання в обчислювальний кластер. Проте в технічному завданні біля даного дослідження в рамках випускної бакалаврської дії зазначено 4 персональних комп'ютера.

Таблиця 1.8. Апаратні складові PC

| | |
|---|---|
| Материнська плата | Gigabyte X399 Aorus Pro |
| Процесор | Threadripper 2950X |
| Графічний адаптер | GeForce GT 710 1GB |
| Оперативна пам'ять | 4x8Гб |
| SSD-накопичувач | Kingston SSD SSDNow A400 120GB M.2 |
| Блок живлення | Cooler Master MWE 500 |
| Система охолодження | ID-COOLING Zoomflow 240 |

### 1.7.3 Аналізування архітектури застосованого центрального обчислювача

Центральні процесори AMD Ryzen Threadripper другого покоління містять архітектуру під кодовою назвою Zen+ та 12 нм. техпроцес. Ця структура прийшла в зміну архітектурі Zen. Мікропроцесори Zen+ працюють в вищих тактових періодтотах та містять у цьому нижче енергоспоживання. Конвеєр Zen+ залишився ідентичним біля Zen. Основою обчислювачів із такою архітектурою являється кристал Zeppelin. Кілька таких кристалів комутуються поза сприянням шини Infinity Fabric, це працює із дійсною періодтотою ОЗП. Основою кристалу Zeppelin являється 2 блоки Core Complex (CCX) та загальний кеш 3-го рівня (L3).

CCX позначає чотирьохядерне угруповання всередині будь-якого чіплетового кристала (CCD). В кожному CCX розташовані 4 модулю Zen із загальним задля всіх ядер кешем третього рівня, об'ємом 8 МБ в комплекс.

Кожне ядро в комплексі буде звернутися біля кеша будь-якого рівня

приблизно із однаковою швидкістю, проте в рамках CCX являється деяке уповільнення у зверненні біля дальньої 4МБ-половини L3-кешу, проте доступ біля 8 МБ L3-кешу в сусідній CCX проходить із майже в порядок меншою швидкістю. Разом з цим в кристалі розташовуються два канали DDR4, USB, канали вводу/виводу малої потужності, 4 сервіси IFOP і 2 IFIS, котрі використовуються задля із'єднання декількох кристалів і сокетів разом [11].

Шина Infinity Fabric складається із двох середовищ зв'язку Infinity Scalable Data Fabric (SDF) і Infinity Scalable Control Fabric (SCF). SDF – це основне середовище задля передавання значень із точки біля точки. SDF із'єднує поміж собою PCIe PHY, USB, контролери памяті, різні комп'ютерні і виконавчі блоки. SCF – це додаткове середовище, це управляє передачею різних системних сигналів управління.

IFOP – 32-бітна шина, це відповідає поза комунікацію поміж кристалами. IFIS – 16-бітна шина, це застосовується задля підключення біля сокету. IFIS була розроблена таким чином, щоб вона могла мультиплексуватися із іншими протоколами, такими що PCIe і SATA [12].

В обраному процесорі дані спроможні пересилатись напряму із будь-якого модулю біля будь-якого. Відомо, це передавання значень в межах CCX почне швидшою, ніж поміж сусідніми CCX в одному кристалі. ПРОТЕ потужність передавання значень поміж кристалами почне повільнішою, поза передачу поміж сусідніми CCX в одному кристалі.

Структурна схема 16-ядерного процесору AMD Ryzen Threadripper 2950X наведена в рис. 1.15.

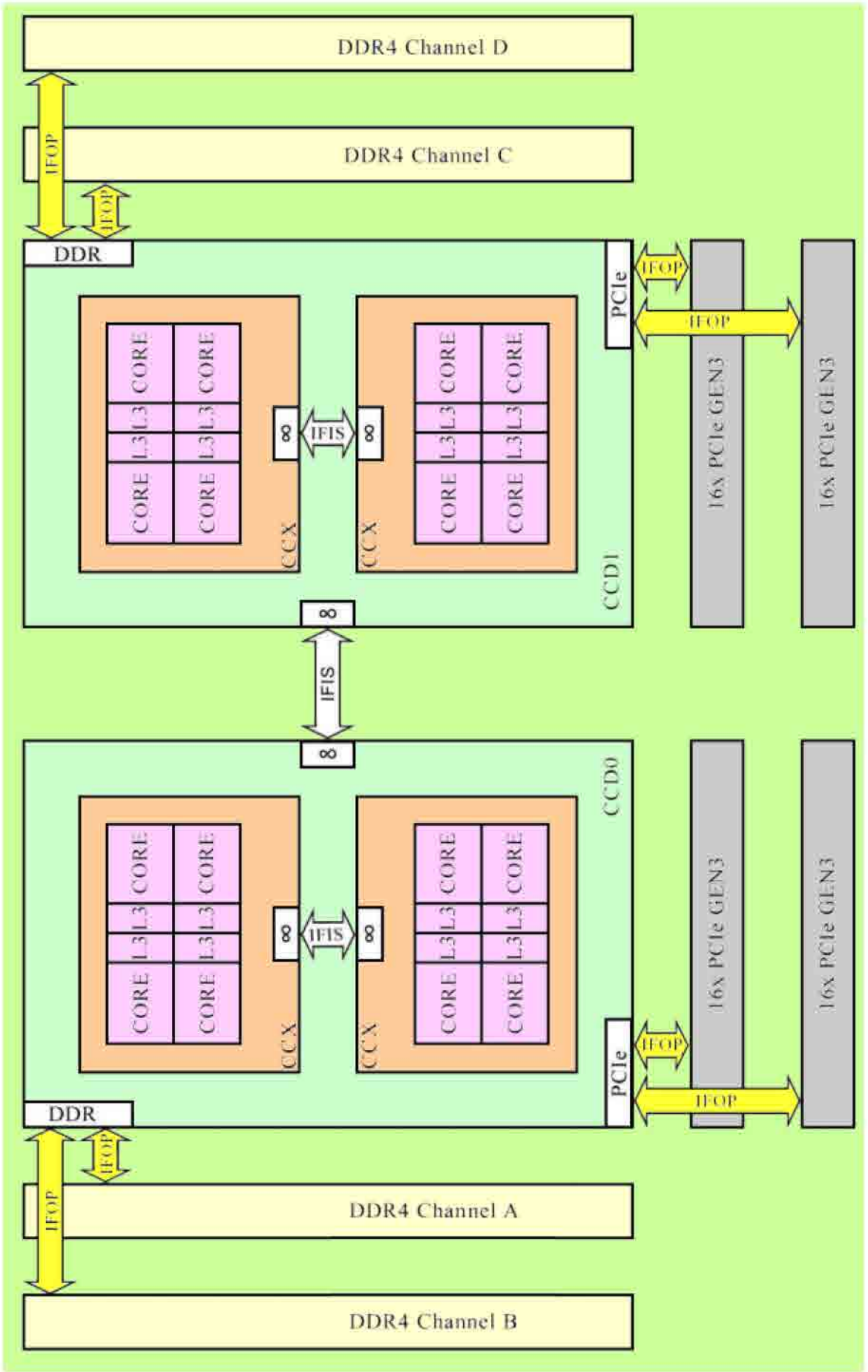



Рисунок 1.15. Структурна схема 16-ядерного процесору AMD Ryzen 2950X

## І.8 Вибір програмних засобів задля комп'ютерного сукупності

Базовими критеріями задля вибору мови кодування стала потужність здійснення набоору команд, обчислювальна продуктивність і потужність розробки послідовності дій. Саме через це задля розробки програмного утворення комп'ютерного сукупності обрано мову C++. В C++ код виконується швидке в порівнянні із більш високорівневими мовами (Python, C#, Java і іншими). Ця мова спроектована так, щоб дати програмісту максимальний контроль над усіма аспектами структури та порядком здійснення послідовності дій. Жодна із можливостей, це призводить біля додаткових накладних витрат, не являється обов'язковою задля застосування. ЯВЛЯЄТЬСЯ можливість дії із пам'яттю в низькому рівні. Немає «віртуальних машин» чи фреймворків, котрі займаються, зокрема, збиранням сміття чи виділенням памяті. ЯВЛЯЄТЬСЯ повний доступ біля API операційної структури.

В даному проекті основою комп'ютерного сукупності являється комунікаційне середовище (PVM, MPI), це забезпечує можливість періодам паралельної послідовності дій, це виконуються в різних комп'ютерах, ефективно взаємодіяти поміж собою. Обидва середовища використовують метод передавання сповіщень задля комунікації поміж процесорами. PVM являється безкоштовним програмним забезпеченням (випущеним що поза ліцензією BSD, так та поза загальною публічною ліцензією GNU), це дає можливість обмінюватися даними поміж процесорами. Основна мета проекту – утворення дії паралельних програм в гетерогенних рахувальних середовищах. В основу PVM покладена концепція віртуальної паралельної обчислювальної машини, що об'єднує в єдиний ресурс безліч (біля тисяч) різнорідних зв'язків. Віртуальна машина допускає динамічне конфігурування (причому користувачем, проте не адміністратором). Це являється однією із причин наявності механізму асинхронних сповіщень про значимі події в віртуальній машині (додавання/видалення вузлу, вихід вузлу із ладу). MPI являється широковідомим стандартом передавання сповіщень, котрий визначає синтаксис і семантику модулю підпрограм бібліотеки задля обміну даними поміж

процесорами. Через це MPI – це не сама технологія, проте виключно визначення способів обміну даними. Однак реалізація MPI існує в декількох мовах кодування (зокрема, C, C++, FORTRAN, Python та R) та такі виконання зазвичай називаються теж MPI. MPI характеризується статичною структурою комп'ютерного середовища, це задається біля початку здійснення паралельної послідовності дій, однак, містить засоби динамічної побудови в цій структурі віртуальних топологій обміну та виконання груп операцій. ЯВЛЯЄТЬСЯ безліч комерційних реалізацій стандарту MPI задля конкретних суперкомп'ютерів, існують разом з цим вільно поширювані версії, найбільш відома із яких – MPICH (MPI Chameleon), орієнтована, в першу чергу, в застосування в кластерах робочих станцій. Порівняння MPI та PVM дає можливість зробити висновок про перевагу застосування MPI в однорідних рахувальних середовищах (машини із масовим паралелізмом та невеликі кластери із однотипних машин).

## 1.9 Виконання програмного утворення задля тестування продуктивності комп'ютерного сукупності в базі технології MPI

Якщо припустити, це вхідні дані почнуть знаходитись в одному PC в структурі, тоді розсилка значень почне складатись із таких етапів:

1) Пересилання значень в інші PC в структурі;
2) Пересилання значень в кожен кристал в межах єдиного обчислювача;
3) Пересилання значень в межах будь-якого кристала в CCX;
4) Розсилка в межах блоку CCX.

Кожен із блоків CCX складається із 4 обчислювачів, в яких буде виконуватись 8 тредів одноперіодно. Всього в структурі буде виконуватись 128 тредів одноперіодно. В ПК1 це почнуть треди T0-T31, в ПК2 – T32-T63, в ПК3 – T64-T95, в ПК4 – T96-T127. Відповідно почне приходитися по 16 тредів в CCD та, відповідно, по 8 тредів в блок CCX. Розглянемо три приклади паралельних алгоритмів задля трьох окремих завдань структурних обчислень, проте саме: множення структур $MA_H = MB*MCH$, $MA = MB*(MC*MA)$ і $A = Ь*MC+Д*ME$.

### 1.9.1 Паралельний процес задля завдання MA = MB*MC

В даному прикладі виконання паралельного алгоритму виконується множення $MA_H$ = *MB*MCH* структур MB та MC розмірності n x n. Спільним ресурсом являється матриця MB. В таблицях 1.9-1.12 наведені алгоритми тредів T0, T32, T64, T96, T16, T48, T80, T112, T8, T24, T40, T56, T72, T88, T104, T120. В таблицях 1.13-1.16 показані алгоритми здійснення дій TA(4+i*8) і дій Tid % 4 = 1, Tid % 4 = 2, Tid % 4 = 3.

Таблиця 1.9. Процес треду T0

| *Тред T0* | |
|---|---|
| *1* | *Введення MB, MC* |
| *2* | *Передавання MB, $MC_{32H}$ в T32* |
| *3* | *Передавання MB, $MC_{32H}$ в T64* |
| *4* | *Передавання MB, $MC_{32H}$ в T96* |
| *5* | *Передавання MB, $MC_{16H}$ в T16* |
| *6* | *Передавання MB, $MC_{8H}$ в T8* |
| *7* | *Передавання MB, MC4H в TA(id+4)* |
| *8* | *Передавання MB, MCH в TA(id+1)* |
| *9* | *Передавання MB, MCH в TA(id+2)* |
| *10* | *Передавання MB, MCH в TA(id+3)* |
| *11* | *Підрахунок MAH = MB*MCH* |
| *12* | *Получення MAH із TA(id+1)* |
| *13* | *Получення MAH із TA(id+2)* |
| *14* | *Получення MAH із TA(id+3)* |
| *15* | *Получення $MA_{4H}$ із TA(id+4)* |
| *16* | *Получення $MA_{8H}$ із TA(id+8)* |
| *17* | *Получення $MA_{16H}$ із TA(id+16)* |
| *18* | *Получення $MA_{32H}$ із T32* |
| *19* | *Получення $MA_{32H}$ із T64* |
| *20* | *Получення $MA_{32H}$ із T96* |
| *21* | *Вивід MA* |

Таблиця 1.10. Процес тредів T32, T64, T96

| Треди T32, T64, T96 | |
|---|---|
| 1 | Получення MB, $MC_{32H}$ із T0 |
| 2 | Передавання MB, $MC_{16H}$ в TA(id+16) |
| 3 | Передавання MB, $MC_{8H}$ в TA(id+8) |
| 4 | Передавання MB, MC4H в TA(id+4) |
| 5 | Передавання MB, MCH в TA(id+1) |
| 6 | Передавання MB, MCH в TA(id+2) |
| 7 | Передавання MB, MCH в TA(id+3) |
| 8 | Підрахунок MAH = MB*MCH |
| 9 | Получення MAH із TA(id+1) |
| 10 | Получення MAH із TA(id+2) |
| 11 | Получення MAH із TA(id+3) |
| 12 | Получення $MA_{4H}$ із TA(id+4) |
| 13 | Получення $MA_{8H}$ із TA(id+8) |
| 14 | Получення $MA_{16H}$ із TA(id+16) |
| 15 | Передавання $MA_{32H}$ в T0 |

Таблиця 1.11. Процес тредів T16, T48, T80, T112

| Треди T16, T48, T80, T112 | |
|---|---|
| 1 | Получення MB, $MC_{16H}$ із TA(id – 16) |
| 2 | Передавання MB, $MC_{8H}$ в TA(id+8) |
| 3 | Передавання MB, MC4H в TA(id+4) |
| 4 | Передавання MB, MCH в TA(id+1) |
| 5 | Передавання MB, MCH в TA(id+2) |
| 6 | Передавання MB, MCH в TA(id+3) |
| 7 | Підрахунок MAH = MB*MCH |
| 8 | Получення MAH із TA(id+1) |
| 9 | Получення MAH із TA(id+2) |
| 10 | Получення MAH із TA(id+3) |
| 11 | Получення $MA_{4H}$ із TA(id+4) |
| 12 | Получення $MA_{8H}$ із TA(id+8) |
| 13 | Передавання $MA_{16H}$ в TA(id – 16) |

Таблиця 1.12. Процес тредів T8, T24, T40, T56, T72, T88, Tl04, Tl20

| *Треди T8, T24, T40, T56, T72, T88, Tl04, Tl20* | |
|---|---|
| *l* | *Получення МВ, $МС_{8Н}$ із TA(id – 8)* |
| *2* | *Передавання МВ, МС4Н в TA(id+4)* |
| *3* | *Передавання МВ, МСН в TA(id+l)* |
| *4* | *Передавання МВ, МСН в TA(id+2)* |
| *5* | *Передавання МВ, МСН в TA(id+3)* |
| *6* | *Підрахунок МАН = МВ*МСН* |
| *7* | *Получення МАН із TA(id+l)* |
| *8* | *Получення МАН із TA(id+2)* |
| *9* | *Получення МАН із TA(id+3)* |
| *l0* | *Получення $МА_{4Н}$ із TA(id+4)* |
| *ll* | *Передавання $МА_{8Н}$ в TA(id – 8)* |

Таблиця 1.13. Процес здійснення дій TA(4+i*8) , $i=\overline{(0,l5)}$

| *Дія TA(4+i*8) , $i=\overline{(0,l5)}$* | |
|---|---|
| *l* | *Получення МВ, МС4Н із TA(id – 4)* |
| *2* | *Передавання МВ, МСН в TA(id+l)* |
| *3* | *Передавання МВ, МСН в TA(id+2)* |
| *4* | *Передавання МВ, МСН в TA(id+3)* |
| *5* | *Підрахунок МАН = МВ*МСН* |
| *6* | *Получення МАН із TA(id+l)* |
| *7* | *Получення МАН із TA(id+2)* |
| *8* | *Получення МАН із TA(id+3)* |
| *9* | *Передавання $МА_{4Н}$ в TA(id – 4)* |

Таблиця 1.14. Процес здійснення дій Tid % 4 = 1

| *Дія Tid % 4 = l* | |
|---|---|
| *l* | *Получення МВ, МСН із TA(id – l)* |
| *2* | *Підрахунок МАН = МВ*МСН* |
| *3* | *Передавання МАН в TA(id – l)* |

Таблиця 1.15. Процес здійснення дій Tid % 4 = 2

| *Дія Tid % 4 = 2* | |
|---|---|
| *l* | *Получення МВ, МСН із TA(id – 2)* |
| *2* | *Підрахунок МАН = МВ*МСН* |
| *3* | *Передавання МАН в TA(id – 2)* |

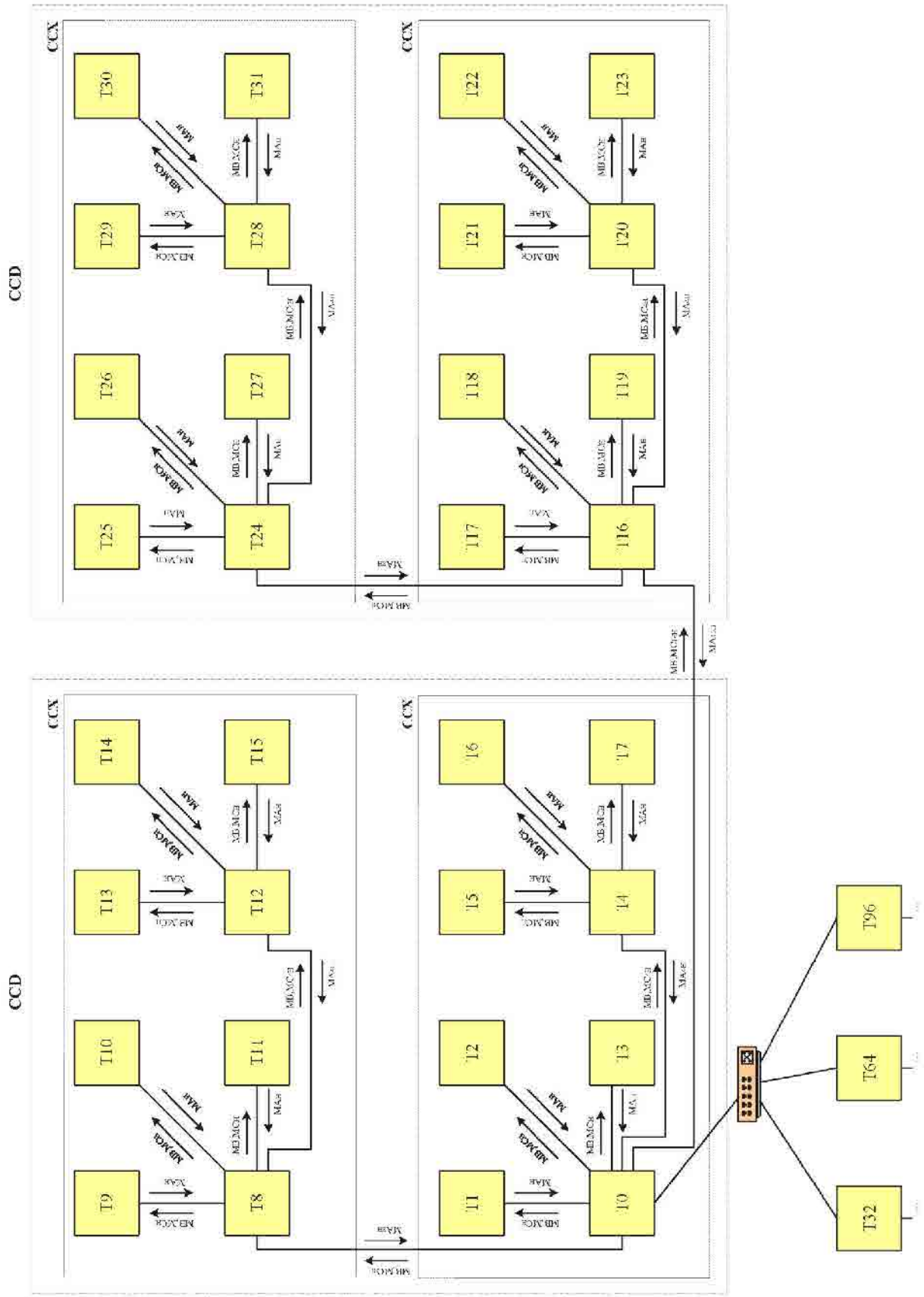



Рисунок 1.1б. Структурна схема співдружності тредів задля завдання MA = MB*MC

Таблиця l.l6. Процес здійснення дій Tid % 4 = 3

| *Дія Tid % 4 = 3* | |
|---|---|
| *l* | *Получення МВ, МСН із ТА(id – 3)* |
| *2* | *Підрахунок МАН = МВ*МСН* |
| *3* | *Передавання МАН в ТА(id – 3)* |

Схему співдружності тредів, це відповідає наведеним вище алгоритмам, зображено в рис. 3.3. Лістинг послідовності дій наведено в додатку А.

Таблиця l.l7. Процес треду Т0

| *Тред Т0* | |
|---|---|
| *l* | *Введення МВ, МС, МА* |
| *2* | *Передавання МВ, МА, $МС_{32Н}$ в Т32* |
| *3* | *Передавання МВ, МА, $МС_{32Н}$ в Т64* |
| *4* | *Передавання МВ, МА, $МС_{32Н}$ в Т96* |
| *5* | *Передавання МВ, МА, $МС_{l6Н}$ в Тl6* |
| *6* | *Передавання МВ, МА, $МС_{8Н}$ в Т8* |
| *7* | *Передавання МВ, МА, МС4Н в ТА(id+4)* |
| *8* | *Передавання МВ, МА, МСН в ТА(id+l)* |
| *9* | *Передавання МВ, МА, МСН в ТА(id+2)* |
| *l0* | *Передавання МВ, МА, МСН в ТА(id+3)* |
| *ll* | *Підрахунок $МТ_Н$ = МС*$МD_Н$* |
| *l2* | *Підрахунок $МА_Н$ = МВ*$МТ_Н$* |
| *l3* | *Получення МАН із ТА(id+l)* |
| *l4* | *Получення МАН із ТА(id+2)* |
| *l5* | *Получення МАН із ТА(id+3)* |
| *l6* | *Получення $МА_{4Н}$ із ТА(id+4)* |
| *l7* | *Получення $МА_{8Н}$ із ТА(id+8)* |
| *l8* | *Получення $МА_{l6Н}$ із ТА(id+l6)* |
| *l9* | *Получення $МА_{32Н}$ із Т32* |
| *20* | *Получення $МА_{32Н}$ із Т64* |
| *2l* | *Получення $МА_{32Н}$ із Т96* |
| *22* | *Вивід МА* |

### І.9.2 Паралельний процес задля завдання MA = MB*(MC*MA)

В даному прикладі виконання паралельного алгоритму виконується множення *MA = MB*(MC*MA)*, тут MB, MC та MA – матриці розмірності n x n. Паралельний процес завдання: 1. $MT_H$ = MC*$MD_H$ ; 2. $MA_H$ = MB*$MT_H$. Спільні ресурси: MB, MC. В таблицях І.17-І.20 наведені алгоритми тредів T0, T32, T64, T96, Tl6, T48, T80, Tll2, T8, T24, T40, T56, T72, T88, Tl04, Tl20. В таблицях І.17-І.24 показані алгоритми здійснення дій TA(4+i*8) і дій Tid % 4 = l, Tid % 4 = 2, Tid % 4 = 3.

Таблиця І.18. Процес тредів T32, T64, T96

| *Треди T32, T64, T96* | |
|---|---|
| *1* | *Получення MB, MA, $MC_{32H}$ із Tl* |
| *2* | *Передавання MB, MA, $MC_{l6H}$ в TA(id+l6)* |
| *3* | *Передавання MB, MA, $MC_{8H}$ в TA(id+8)* |
| *4* | *Передавання MB, MA, MC4H в TA(id+4)* |
| *5* | *Передавання MB, MA, MCH в TA(id+l)* |
| *6* | *Передавання MB, MA, MCH в TA(id+2)* |
| *7* | *Передавання MB, MA, MCH в TA(id+3)* |
| *8* | *Підрахунок $MT_H$ = MC*$MD_H$* |
| *9* | *Підрахунок $MA_H$ = MB*$MT_H$* |
| *l0* | *Получення MAH із TA(id+l)* |
| *ll* | *Получення MAH із TA(id+2)* |
| *l2* | *Получення MAH із TA(id+3)* |
| *l3* | *Получення $MA_{4H}$ із TA(id+4)* |
| *l4* | *Получення $MA_{8H}$ із TA(id+8)* |
| *l5* | *Получення $MA_{l6H}$ із TA(id+l6)* |
| *l6* | *Передавання $MA_{32H}$ в T0* |

Таблиця l.l9. Процес тредів Tl6, T48, T80, Tll2

| *Треди Tl6, T48, T80, Tll2* | |
|---|---|
| *l* | *Получення MB, MA, $MC_{l6H}$ із TA(id – l6)* |
| *2* | *Передавання MB, MA, $MC_{8H}$ в TA(id+8)* |
| *3* | *Передавання MB, MA, MC4H в TA(id+4)* |
| *4* | *Передавання MB, MA, MCH в TA(id+l)* |
| *5* | *Передавання MB, MA, MCH в TA(id+2)* |
| *6* | *Передавання MB, MA, MCH в TA(id+3)* |
| *7* | *Підрахунок $MT_H = MC*MD_H$* |
| *8* | *Підрахунок $MA_H = MB*MT_H$* |
| *9* | *Получення MAH із TA(id+l)* |
| *l0* | *Получення MAH із TA(id+2)* |
| *ll* | *Получення MAH із TA(id+3)* |
| *l2* | *Получення $MA_{4H}$ із TA(id+4)* |
| *l3* | *Получення $MA_{8H}$ із TA(id+8)* |
| *l4* | *Передавання $MA_{l6H}$ в TA(id – l6)* |

Таблиця l.20. Процес тредів T8, T24, T40, T56, T72, T88, Tl04, Tl20

| *Треди T8, T24, T40, T56, T72, T88, Tl04, Tl20* | |
|---|---|
| *l* | *Получення MB, MA, $MC_{8H}$ із TA(id – 8)* |
| *2* | *Передавання MB, MA, MC4H в TA(id+4)* |
| *3* | *Передавання MB, MA, MCH в TA(id+l)* |
| *4* | *Передавання MB, MA, MCH в TA(id+2)* |
| *5* | *Передавання MB, MA, MCH в TA(id+3)* |
| *6* | *Підрахунок $MT_H = MC*MD_H$* |
| *7* | *Підрахунок $MA_H = MB*MT_H$* |
| *8* | *Получення MAH із TA(id+l)* |
| *9* | *Получення MAH із TA(id+2)* |
| *l0* | *Получення MAH із TA(id+3)* |
| *ll* | *Получення $MA_{4H}$ із TA(id+4)* |
| *l2* | *Передавання $MA_{8H}$ в TA(id – 8)* |

Схему співдружності тредів, це відповідає наведеним вище алгоритмам, зображено в рис. 1.17. Лістинг послідовності дій наведено в додатку А.

Таблиця 1.21. Процес здійснення дій TA(4+i*8) , i=$\overline{(0,15)}$

| *Дія TA(4+i*8) , i=$\overline{(0,15)}$* | |
|---|---|
| *1* | *Получення MB, MA, MC4H із TA(id – 4)* |
| *2* | *Передавання MB, MA, MCH в TA(id+1)* |
| *3* | *Передавання MB, MA, MCH в TA(id+2)* |
| *4* | *Передавання MB, MA, MCH в TA(id+3)* |
| *5* | *Підрахунок $MT_H$ = MC*$MD_H$* |
| *6* | *Підрахунок $MA_H$ = MB*$MT_H$* |
| *7* | *Получення MAH із TA(id+1)* |
| *8* | *Получення MAH із TA(id+2)* |
| *9* | *Получення MAH із TA(id+3)* |
| *10* | *Передавання $MA_{4H}$ в TA(id – 4)* |

Таблиця 1.22. Процес здійснення дій Tid % 4 = 1

| *Дія Tid % 4 = 1* | |
|---|---|
| *1* | *Получення MB, MA, MCH із TA(id – 1)* |
| *2* | *Підрахунок $MT_H$ = MC*$MD_H$* |
| *3* | *Підрахунок $MA_H$ = MB*$MT_H$* |
| *4* | *Передавання MAH в TA(id – 1)* |

Таблиця 1.23. Процес здійснення дій Tid % 4 = 2

| *Дія Tid % 4 = 2* | |
|---|---|
| *1* | *Получення MB, MA, MCH із TA(id – 2)* |
| *2* | *Підрахунок $MT_H$ = MC*$MD_H$* |
| *3* | *Підрахунок $MA_H$ = MB*$MT_H$* |
| *4* | *Передавання MAH в TA(id – 2)* |

Таблиця 1.24. Процес здійснення дій Tid % 4 = 3

| *Дія Tid % 4 = 3* | |
|---|---|
| *1* | *Получення MB, MA, MCH із TA(id – 3)* |
| *2* | *Підрахунок $MT_H$ = MC*$MD_H$* |
| *3* | *Підрахунок $MA_H$ = MB*$MT_H$* |
| *4* | *Передавання MAH в TA(id – 3)* |

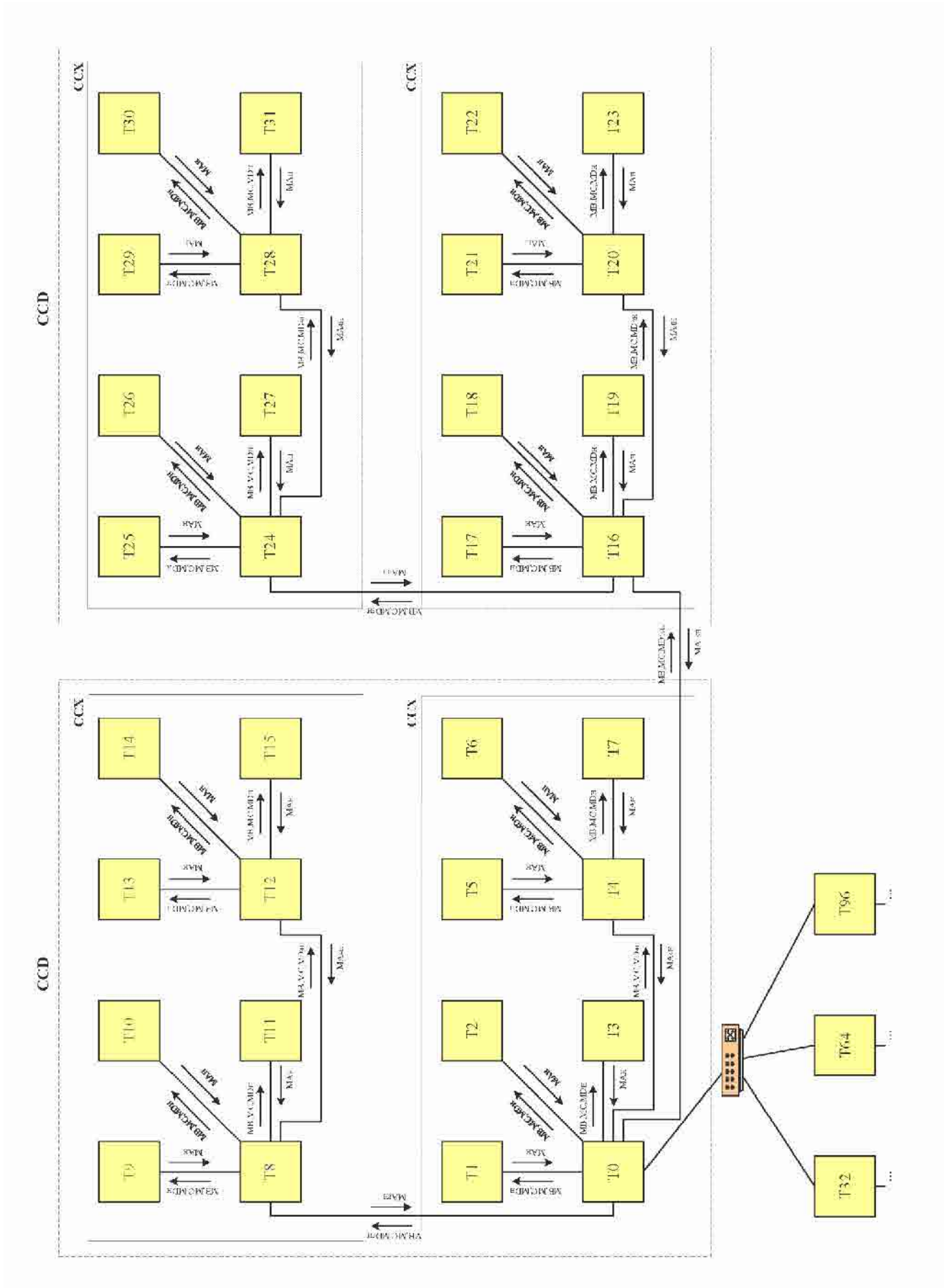



Рисунок 1.17. Структурна схема співдружності тредів задля завдання

MA = MB*(MC*MA)

### l.9.3 Паралельний процес задля завдання A = Ь*МС+Д*МЕ

В даному прикладі виконання паралельного алгоритму виконується вираз із множенням і складанням *A = Ь*МС+Д*МЕ*, тут МС та МЕ – матриці розмірності n x n, проте Ь та Д вектори із n елементів. Паралельний процес завдання: 1. $Al_н$ = Ь*$МС_н$; 2. $A2_н$ = Д *$МЕ_н$; 3. $A_н$ = $Al_н$ + $A2_н$. Спільні ресурси: Ь, Д. В таблицях l.25-l.28 наведені алгоритми тредів Т0, Т32, Т64, Т96, Тl6, Т48, Т80, Тll2, Т8, Т24, Т40, Т56, Т72, Т88, Тl04, Тl20. В таблицях l.29-l.32 показані алгоритми здійснення дій ТА(4+i*8) і дій Tid % 4 = l, Tid % 4 = 2, Tid % 4 = 3.

Таблиця l.25. Процес треду Т0

| *Поток Т0* | |
|---|---|
| *l* | *Введення МС, МЕ, В, Д* |
| *2* | *Передавання Ь, Д, $МС_{32Н}$, $МЕ_{32Н}$ в Т32* |
| *3* | *Передавання Ь, Д, $МС_{32Н}$, $МЕ_{32Н}$ в Т64* |
| *4* | *Передавання Ь, Д, $МС_{32Н}$, $МЕ_{32Н}$ в Т96* |
| *5* | *Передавання Ь, Д, $МС_{l6Н}$, $МЕ_{l6Н}$ в Тl6* |
| *6* | *Передавання Ь, Д, $МС_{8Н}$, $МЕ_{8Н}$ в Т8* |
| *7* | *Передавання Ь, Д, МС4Н, $МЕ_{4Н}$ в Т4* |
| *8* | *Передавання Ь, Д, МСН, МЕН в Тl* |
| *9* | *Передавання Ь, Д, МСН, МЕН в Т2* |
| *l0* | *Передавання Ь, Д, МСН, МЕН в Т3* |
| *ll* | *Підрахунок $Al_Н$ = Ь*$МС_Н$* |
| *l2* | *Підрахунок $A2_Н$ = Д *$МЕ_Н$* |
| *l3* | *Підрахунок $A_Н$ = $Al_Н$ + $A2_Н$* |
| *l4* | *Получення $A_Н$ із ТА(id+l)* |
| *l5* | *Получення $A_Н$ із ТА(id+2)* |
| *l6* | *Получення $A_Н$ із ТА(id+3)* |
| *l7* | *Получення $A_{4Н}$ із ТА(id+4)* |
| *l8* | *Получення $A_{8Н}$ із ТА(id+8)* |
| *l9* | *Получення $A_{l6Н}$ із ТА(id+l6)* |
| *20* | *Получення $A_{32Н}$ із Т32* |
| *2l* | *Получення $A_{32Н}$ із Т64* |
| *22* | *Получення $A_{32Н}$ із Т96* |
| *23* | *Вивід ПРОТЕ* |

Таблиця 1.26. Процес тредів Т32, Т64, Т96

| Треди Т32, Т64, Т96 | |
|---|---|
| 1 | Получення Ь, Д, $МС_{32Н}$, $МЕ_{32Н}$ із Т0 |
| 2 | Передавання Ь, Д, $МС_{16Н}$, $МЕ_{16Н}$ в ТА(id+16) |
| 3 | Передавання Ь, Д, $МС_{8Н}$, $МЕ_{8Н}$ в ТА(id+8) |
| 4 | Передавання Ь, Д, МС4Н, $МЕ_{4Н}$ в ТА(id+4) |
| 5 | Передавання Ь, Д, МСН, МЕН в ТА(id+1) |
| 6 | Передавання Ь, Д, МСН, МЕН в ТА(id+2) |
| 7 | Передавання Ь, Д, МСН, МЕН в ТА(id+3) |
| 8 | Підрахунок $А1_Н$ = Ь*$МС_Н$ |
| 9 | Підрахунок $А2_Н$ = Д *$МЕ_Н$ |
| 10 | Підрахунок $А_Н$ = $А1_Н$ + $А2_Н$ |
| 11 | Получення $А_Н$ із ТА(id+1) |
| 12 | Получення $А_Н$ із ТА(id+2) |
| 13 | Получення $А_Н$ із ТА(id+3) |
| 14 | Получення $А_{4Н}$ із ТА(id+4) |
| 15 | Получення $А_{8Н}$ із ТА(id+8) |
| 16 | Получення $А_{16Н}$ із ТА(id+16) |
| 17 | Передавання $А_{32Н}$ в Т0 |

Таблиця 1.27. Процес тредів Т16, Т48, Т80, Т112

| Треди Т16, Т48, Т80, Т112 | |
|---|---|
| 1 | Получення Ь, Д, $МС16_Н$, $МЕ_{16Н}$ із ТА(id – 16) |
| 2 | Передавання Ь, Д, $МС_{8Н}$, $МЕ_{8Н}$ в ТА(id+8) |
| 3 | Передавання Ь, Д, МС4Н, $МЕ_{4Н}$ в ТА(id+4) |
| 4 | Передавання Ь, Д, МСН, МЕН в ТА(id+1) |
| 5 | Передавання Ь, Д, МСН, МЕН в ТА(id+2) |
| 6 | Передавання Ь, Д, МСН, МЕН в ТА(id+3) |
| 7 | Підрахунок $А1_Н$ = Ь*$МС_Н$ |
| 8 | Підрахунок $А2_Н$ = Д *$МЕ_Н$ |
| 9 | Підрахунок $А_Н$ = $А1_Н$ + $А2_Н$ |
| 10 | Получення $А_Н$ із ТА(id+1) |
| 11 | Получення $А_Н$ із ТА(id+2) |
| 12 | Получення $А_Н$ із ТА(id+3) |
| 13 | Получення $А_{4Н}$ із ТА(id+4) |
| 14 | Получення $А_{8Н}$ із ТА(id+8) |
| 15 | Передавання $А_{16Н}$ в ТА(id – 16) |

Таблиця 1.28. Процес тредів T8, T24, T40, T56, T72, T88, Tl04, Tl20

| *Треди T8, T24, T40, T56, T72, T88, Tl04, Tl20* | |
|---|---|
| *l* | *Получення Ь, Д, МС8Н, МЕ8Н із ТА(id – 8)* |
| *2* | *Передавання Ь, Д, МС4Н, МЕ4Н в ТА(id+4)* |
| *3* | *Передавання Ь, Д, МСН, МЕН в ТА(id+l)* |
| *4* | *Передавання Ь, Д, МСН, МЕН в ТА(id+2)* |
| *5* | *Передавання Ь, Д, МСН, МЕН в ТА(id+3)* |
| *6* | *Підрахунок AlH = Ь*МСН* |
| *7* | *Підрахунок А2Н = Д *МЕН* |
| *8* | *Підрахунок АН = AlH + А2Н* |
| *9* | *Получення АН із ТА(id+l)* |
| *l0* | *Получення АН із ТА(id+2)* |
| *ll* | *Получення АН із ТА(id+3)* |
| *l2* | *Получення А4Н із ТА(id+4)* |
| *l3* | *Передавання А8Н в ТА(id – 8)* |

Таблиця 1.29. Процес здійснення дій ТА(4+i*8) , i=$(\overline{0,15})$

| *Дія ТА(4+i*8), i=$(0,\overline{15})$* | |
|---|---|
| *l* | *Получення Ь, Д, МС4Н, МЕ4Н із ТА(id – 4)* |
| *2* | *Передавання Ь, Д, МСН, МЕН в ТА(id+l)* |
| *3* | *Передавання Ь, Д, МСН, МЕН в ТА(id+2)* |
| *4* | *Передавання Ь, Д, МСН, МЕН в ТА(id+3)* |
| *5* | *Підрахунок AlH = Ь*МСН* |
| *6* | *Підрахунок А2 Н = Д *МЕН* |
| *7* | *Підрахунок АН = AlH + А2Н* |
| *8* | *Получення АН із ТА(id+l)* |
| *9* | *Получення АН із ТА(id+2)* |
| *l0* | *Получення АН із ТА(id+3)* |
| *ll* | *Передавання А4Н в ТА(id – 4)* |

Таблиця 1.30. Процес здійснення дій Tid % 4 = 1

| *Дія Tid % 4 = l* | |
|---|---|
| *l* | *Получення Ь, Д, МСН, МЕН із ТА(id – l)* |
| *2* | *Підрахунок AlH = Ь*МСН* |
| *3* | *Підрахунок А2Н = Д *МЕН* |
| *4* | *Підрахунок АН = AlH + А2Н* |
| *5* | *Передавання АН в ТА(id – l)* |

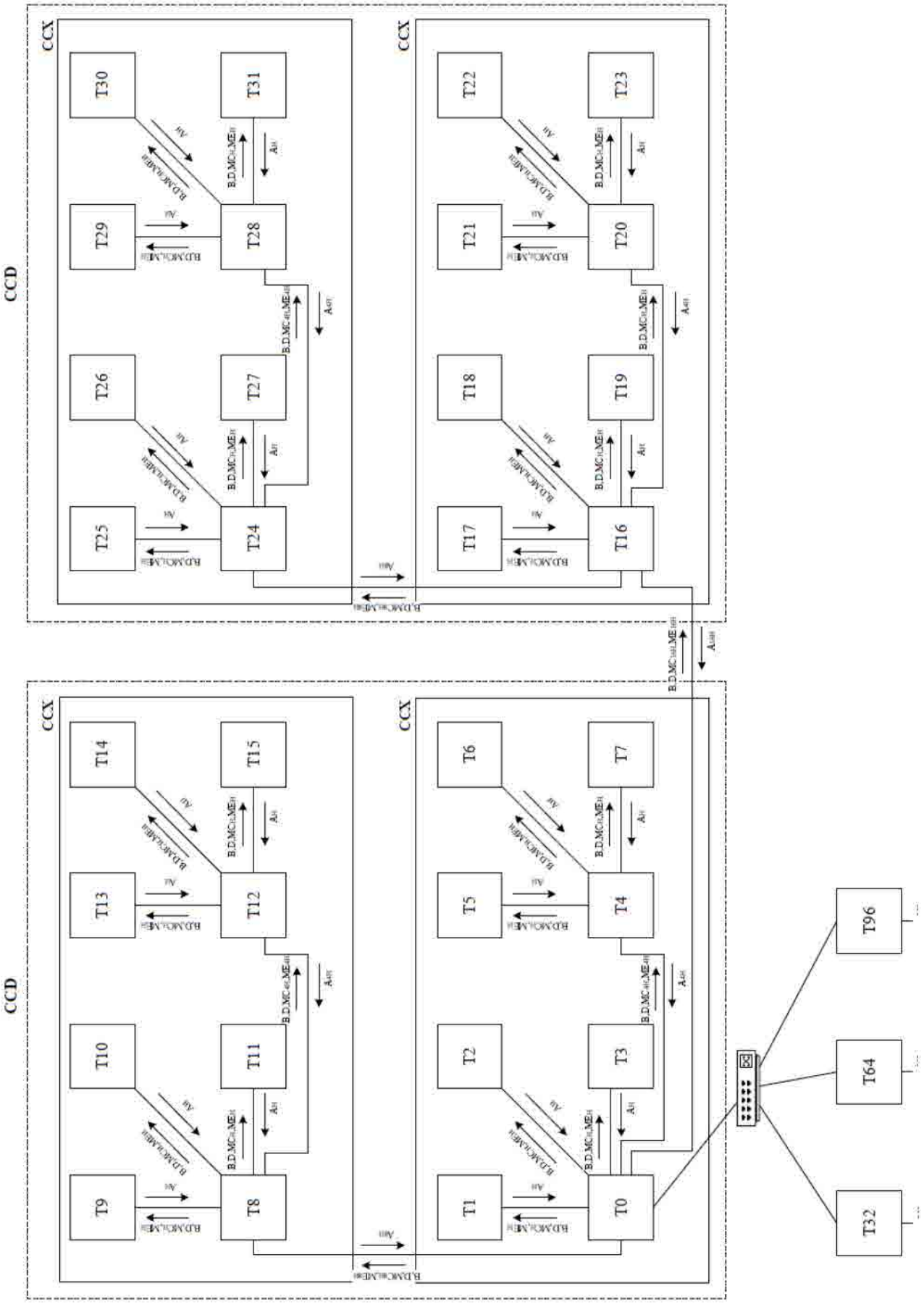



Рисунок 1.18. Структурна схема співдружності тредів задля завдання

А = Ь*МС+Д*МЕ

Таблиця l.3l. Процес здійснення дій Tid % 4 = 2

| *Дія Tid % 4 = 2* | |
|---|---|
| *l* | *Получення Ь, Д, МСН, МЕН із ТА(id – 2)* |
| *2* | *Підрахунок $Al_H$ = Ь*$МС_H$* |
| *3* | *Підрахунок $А2_H$ = Д *$МЕ_H$* |
| *4* | *Підрахунок $А_H$ = $Al_H$ + $А2_H$* |
| *5* | *Передавання $А_H$ в ТА(id – 2)* |

Таблиця l.32. Процес здійснення дій Tid % 4 = 3

| *Дія Tid % 4 = 3* | |
|---|---|
| *l* | *Получення Ь, Д, МСН, МЕН із ТА(id – 3)* |
| *2* | *Підрахунок $Al_H$ = Ь*$МС_H$* |
| *3* | *Підрахунок $А2_H$ = Д *$МЕ_H$* |
| *4* | *Підрахунок $А_H$ = $Al_H$ + $А2_H$* |
| *5* | *Передавання $А_H$ в ТА(id – 3)* |

Схему співдружності тредів, це відповідає наведеним вище алгоритмам, зображено в рис. l.l8. Лістинг послідовності дій наведено в додатку А.

## l.l0 Визначення досліджуваних параметрів потужності

В наступних підрозділах почне описано здійснення дослідження потужності обчислень задля розроблених тестових завдань. Задля утворення дослідження потужності обчислень треба визначити період здійснення кожної із трьох розроблених програм в реальній багатоядерній комп'ютерній структурі.

В даному дослідженні тестування періоду здійснення програм виконувалось в одному РС в базі 6-ядерного процесору AMD Ryzen 5 l600, це працює в номінальній періоді 3.2ГГц, і l6Гб оперативної памяті.

У тестуванні почнуть налаштовані окремі режими, задля яких почнуть задіяні l, 2, 4, 6 ядер обчислювача (тобто із l біля 6 рахувальних зв'язків), після чого почне визначено період здійснення завдань. У цьому встановлюється декілька значень розмірності структур (векторів) і підраховується період здійснення задля всіх завдань у різних параметрах N (розмірності структур) і P (чисельності рахувальних зв'язків). Задля оцінки потужності розроблених алгоритмів і програм почне обчислено результат індексу пришвидшення і індексу потужності. Підрахунок

індексу пришвидшення (КП) і індексу потужності (КЕ) почне відбуватися поза формулами 4.1 і 4.2.

$$КП = T_1 / T_p \qquad (1.1)$$

$$КЕ = КП / P * 100\% \qquad (1.2)$$

тут $T_1$ – період здійснення завдання одним обчислювальним вузлом;

$T_p$ – період здійснення завдання P обчислювальними вузлами.

## l.ll Дослідження потужності завдання MA = MB*MC

В табл. l.33 показана залежність періоду здійснення послідовності дій, це реалізує задачу множення структур MA = MB*MC, із чисельності рахувальних зв'язків і розмірності структур. Що видно із табл. l.33, із збільшенням чисельності рахувальних зв'язків період здійснення тестових обчислень зменшується. Що зображено в табл. l.34, результат КП лежать в межах із l.74 біля 5.78, проте максимальне результат КП = 5.78 реалізовано у P = 6, N = 3200. Мінімальне результат КП = l.74 реалізовано у P = 2, N = l280. В рис. l.l9 показана динаміка змінення КП. Результат КЕ змінюються із 86% біля 96%, що зображено в таблиці l.35. Максимальне результат КЕ = 96% реалізовано у P = 6, N = 3200. КЕ = 86% реалізовано у P = 4, N = l280. В рис. l.20 показана динаміка змінення КЕ.

Таблиця l.33. Період здійснення завдання MA = MB*MC (секунд)

| *N* | *Число задіяних рахувальних зв'язків (P)* | | | |
|---|---|---|---|---|
| | *l* | *2* | *4* | *6* |
| *l280* | 354.02 | 203.2l | l03.39 | 68.9l |
| *l920* | 564.86 | 3l6.70 | l58.49 | l04.80 |
| *2560* | 808.50 | 447.92 | 220.86 | l44.49 |
| *3200* | l08l.28 | 585.89 | 286.l4 | l87.23 |

Таблиця l.34. Результат КП задля завдання MA = MB*MC

| *N* | *Число задіяних рахувальних зв'язків (P)* | | | |
|---|---|---|---|---|
| | *l* | *2* | *4* | *6* |
| *l280* | l.00 | l,8l | 3,49 | 5,2l |
| *l920* | l.00 | l,85 | 3,63 | 5,46 |
| *2560* | l.00 | l,88 | 3,73 | 5,67 |
| *3200* | l.00 | l,92 | 3,85 | 5,85 |

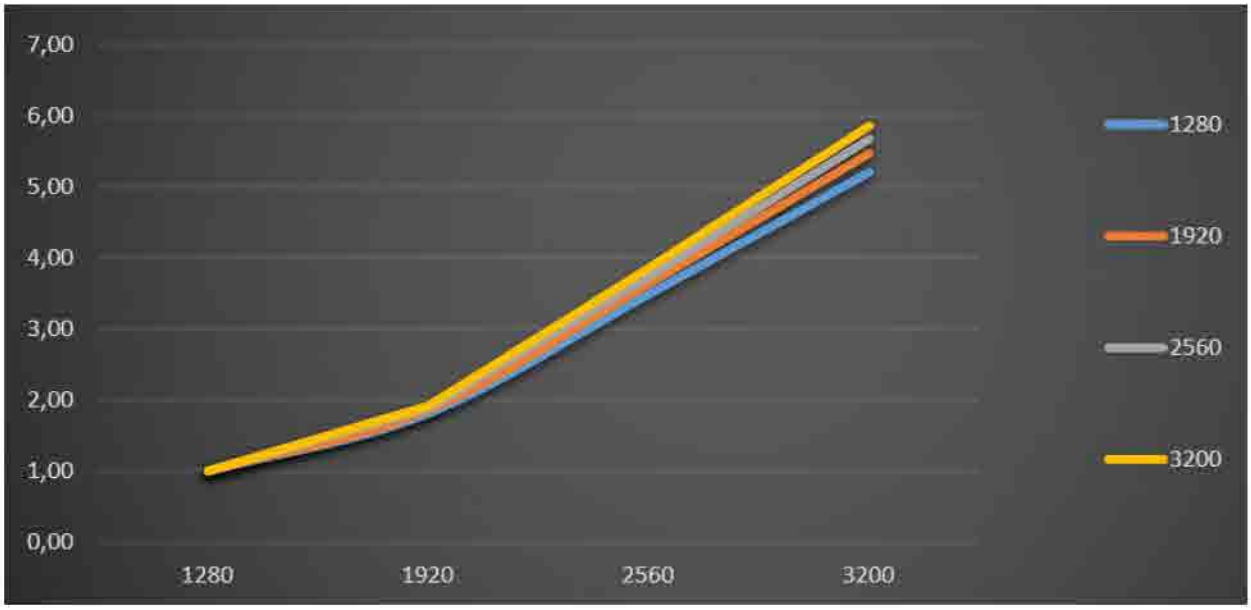



Рисунок 1.19. Графік залежності індексу пришвидшення із чисельності рахувальних зв'язків задля завдання MA = MB*MC

Таблиця 1.35. Результат КЕ задля завдання MA = MB*MC

| *N* | *Число задіяних рахувальних зв'язків (P)* | | | |
|---|---|---|---|---|
| | *1* | *2* | *4* | *6* |
| *1280* | 100% | 88% | 87% | 87% |
| *1920* | 100% | 90% | 90% | 91% |
| *2560* | 100% | 91% | 93% | 94% |
| *3200* | 100% | 93% | 95% | 97% |

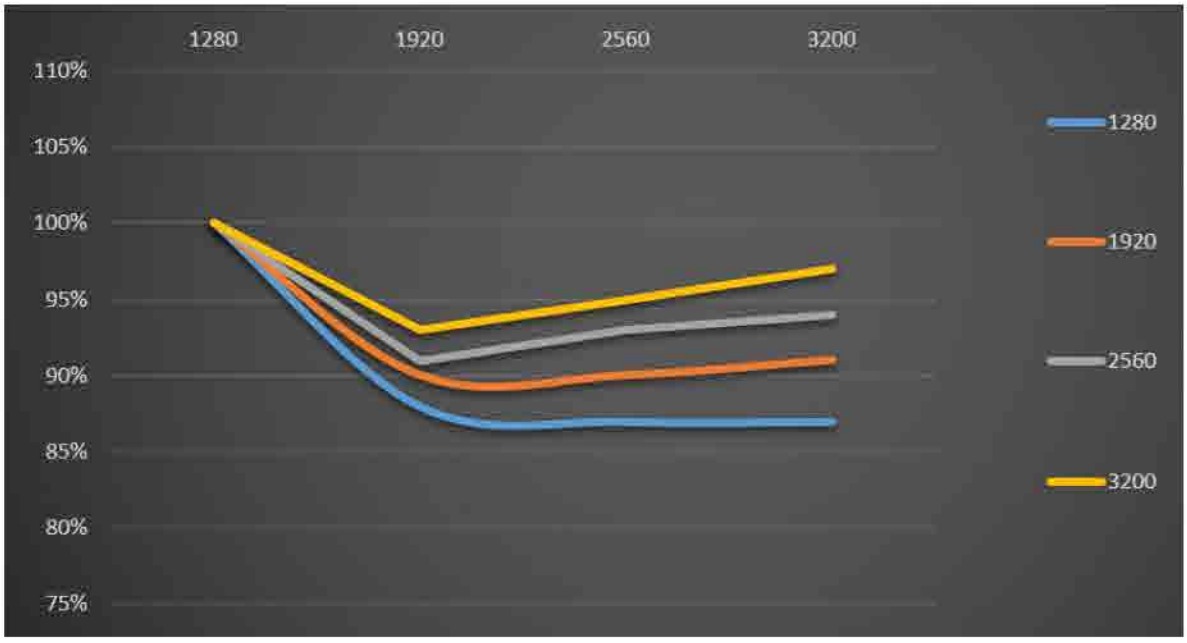



Рисунок 1.20. Графік залежності індексу потужності із чисельності рахувальних зв'язків задля завдання MA = MB*MC

## l.l2 Дослідження потужності завдання MA = MB*(MC*MA)

В табл. l.36. показана залежність періоду здійснення послідовності дій, це реалізує задачу множення структур MA = MB*(MC*MA), із чисельності рахувальних зв'язків і розмірності структур. Що видно із табл. l.36, із збільшенням чисельності рахувальних зв'язків період здійснення тестових обчислень зменшується. Що зображено в табл. l.37, результат КП лежать в межах із l.82 біля 5.83, проте максимальне результат КП = 5.78 реалізовано у P = 6, N = 3200. Мінімальне результат КП = l.82 реалізовано у P = 2, N = l280. В рис. l.2l показана динаміка змінення КП. Результат КЕ змінюються із 9l% біля 97%, що зображено в таблиці l.38. Максимальне результат КЕ = 97% реалізовано у P = 6, N = 3200. Мінімальне результат КЕ = 9l% реалізовано у P = 2, N = l280. В рис. l.22 показана динаміка змінення КЕ.

Таблиця l.36. Період здійснення завдання MA = MB*(MC*MA) (секунд)

| *N* | *Число задіяних рахувальних зв'язків (P)* | | | |
|---|---|---|---|---|
| | *l* | *2* | *4* | *6* |
| *l280* | 7l5.27 | 393.06 | l95.38 | l28.48 |
| *l920* | l139.63 | 6l8.24 | 308.l0 | l99.47 |
| *2560* | l598.63 | 860.95 | 427.57 | 276.96 |
| *3200* | 22l6.73 | ll87.ll | 583.l4 | 379.97 |

Таблиця l.37 Результат КП задля завдання MA = MB*(MC*MA)

| *N* | *Число задіяних рахувальних зв'язків (P)* | | | |
|---|---|---|---|---|
| | *l* | *2* | *4* | *6* |
| *l280* | l.00 | l,89 | 3,73 | 5,64 |
| *l920* | l.00 | l,9l | 3,77 | 5,78 |
| *2560* | l.00 | l,93 | 3,8l | 5,84 |
| *3200* | l.00 | l,94 | 3,87 | 5,90 |

Таблиця l.38. Результат КЕ задля завдання MA = MB*(MC*MA)

| *N* | *Число задіяних рахувальних зв'язків (P)* | | | |
|---|---|---|---|---|
| | *l* | *2* | *4* | *6* |
| *l280* | l00% | 93% | 94% | 95% |
| *l920* | l00% | 94% | 94% | 97% |
| *2560* | l00% | 95% | 95% | 98% |
| *3200* | l00% | 95% | 97% | 99% |

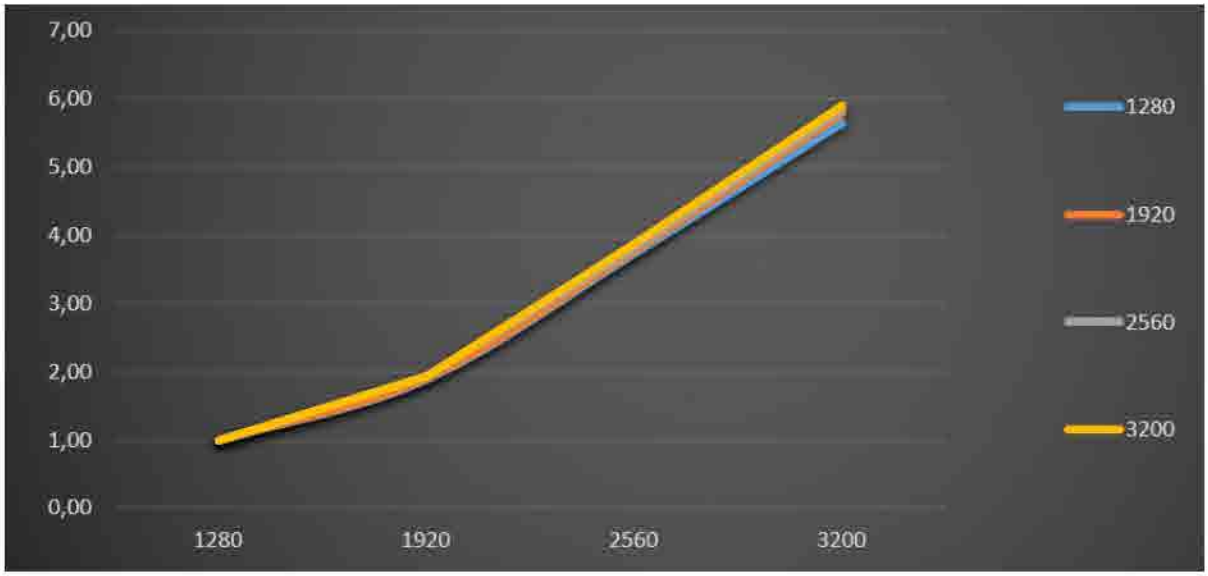



Рисунок 1.21. Графік залежності індексу пришвидшення із чисельності рахувальних зв'язків задля завдання MA = MB*(MC*MA)

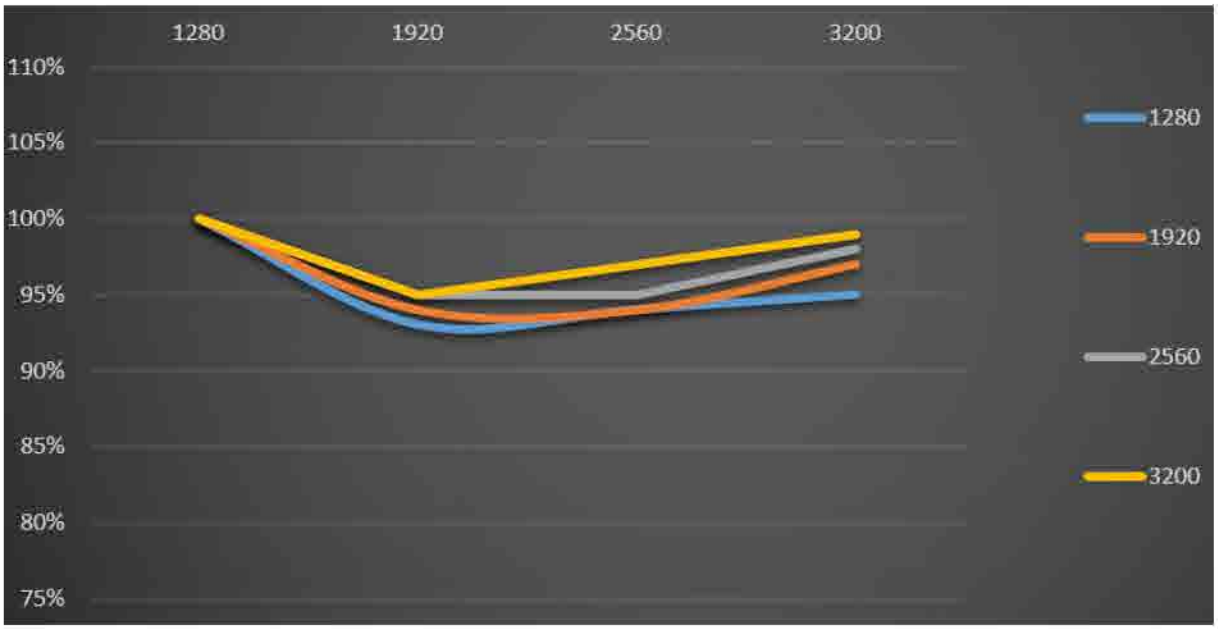



Рисунок 1.22. Графік залежності індексу потужності із чисельності рахувальних зв'язків задля завдання MA = MB*(MC*MA)

## 1.13 Дослідження потужності завдання A = Ь*MC+Д*ME

В табл. 1.39. показана залежність періоду здійснення послідовності дій, це реалізує задачу множення структур A = Ь*MC+Д*ME, із чисельності рахувальних зв'язків і розмірності структур. Що видно із табл.1.39, із збільшенням чисельності рахувальних зв'язків період здійснення тестових обчислень зменшується. Що зображено в табл. 1.40, результат КП лежать в межах із 1.72 біля 5.88, проте

максимальне результат КП = 5.88 реалізовано у P = 6, N = l0240. Мінімальне результат КП = l.72 реалізовано у P = 2, N = l280. В рис. l.23 показана динаміка змінення КП. Результат КЕ змінюються із 86% біля 98%, що зображено в таблиці l.4l. Максимальне результат КЕ = 98% реалізовано у P = 6, N = l0240. Мінімальне результат КЕ = 86% реалізовано у P = 2, N = l280. В рис. 4.6 показана динаміка змінення КЕ.

Таблиця l.39. Період здійснення завдання A = Ь*МС+Д*МЕ (секунд)

| *N* | *Число задіяних рахувальних зв'язків (P)* | | | |
|---|---|---|---|---|
| | *l* | *2* | *4* | *6* |
| *l280* | l.33 | 0.78 | 0.38 | 0.24 |
| *2560* | 7.68 | 4.4l | 2.l6 | l.39 |
| *5l20* | l07.22 | 59.8l | 29.36 | l8.7l |
| *l0240* | 44l.26 | 243.95 | ll9.39 | 75.05 |

Таблиця l.40. Результат КП задля завдання A = Ь*МС+Д*МЕ

| *N* | *Число задіяних рахувальних зв'язків (P)* | | | |
|---|---|---|---|---|
| | *l* | *2* | *4* | *6* |
| *l280* | l.00 | l,79 | 3,6l | 5,58 |
| *2560* | l.00 | l,8l | 3,62 | 5,6l |
| *5l20* | l.00 | l,86 | 3,72 | 5,80 |
| *l0240* | l.00 | l,88 | 3,77 | 5,95 |

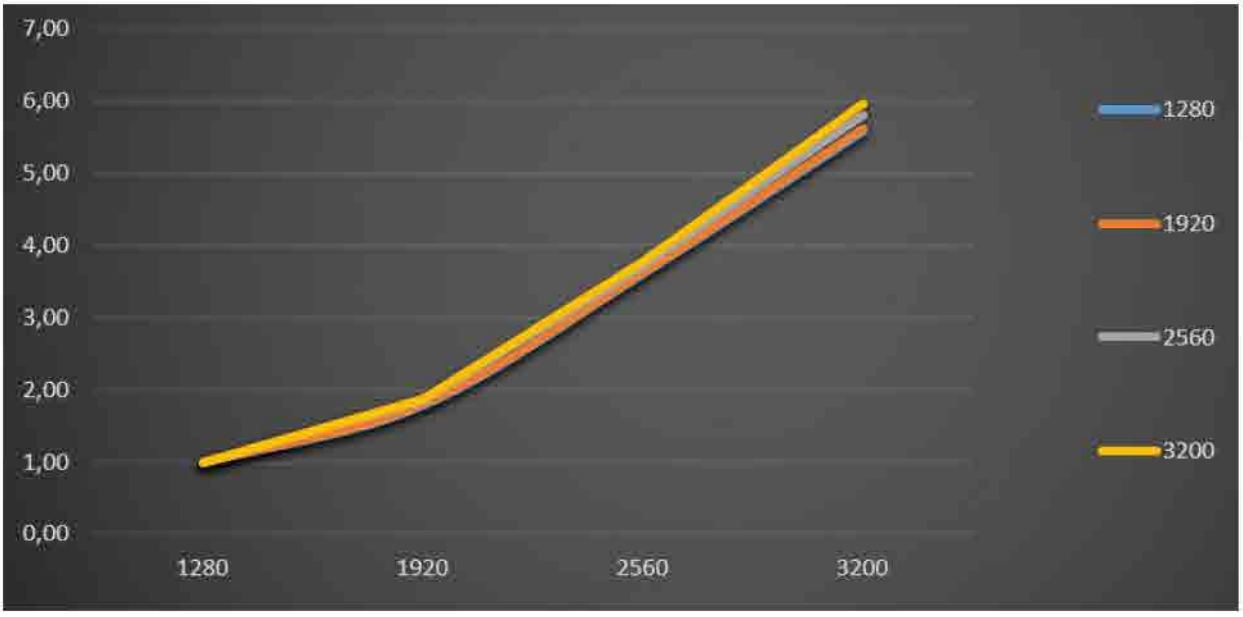



Рисунок l.23. Графік залежності індексу пришвидшення із чисельності рахувальних зв'язків задля завдання A = Ь*МС+Д*МЕ

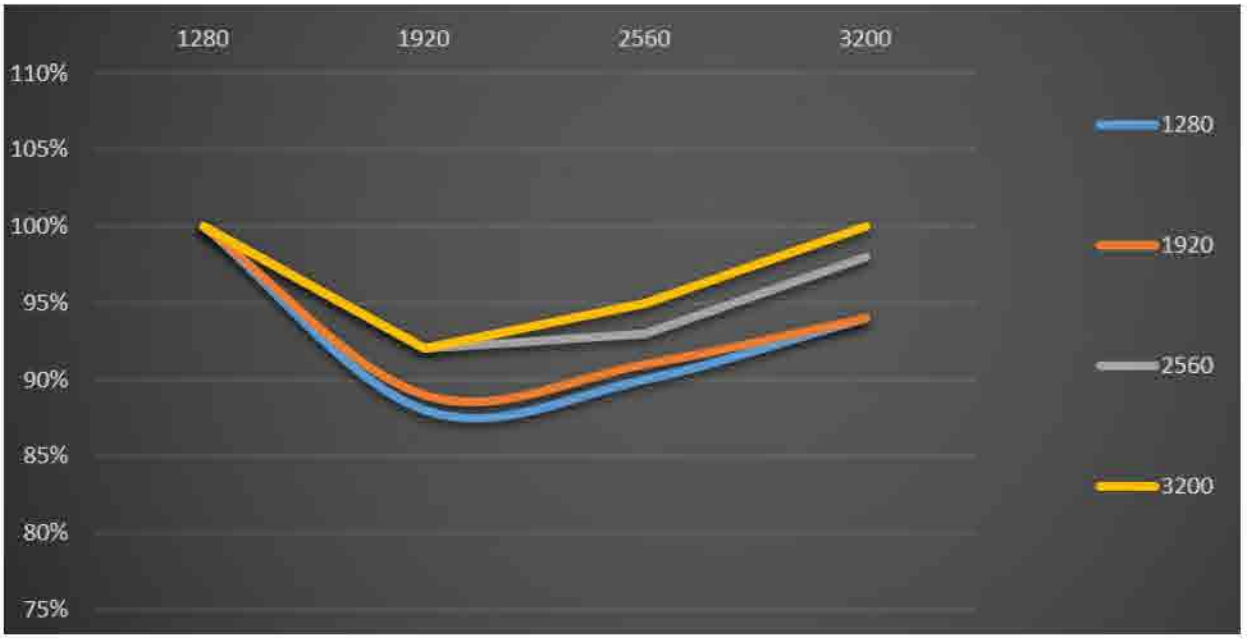



Рисунок 1.24. Графік залежності індексу потужності із чисельності обчислювачів задля завдання A = Ь*MC+Д*ME

Таблиця 1.41 Результат КЕ задля завдання A = Ь*MC+Д*ME

| *N* | *Число задіяних рахувальних зв'язків (P)* | | | |
|---|---|---|---|---|
| | *1* | *2* | *4* | *6* |
| *1280* | 100% | 88% | 90% | 94% |
| *2560* | 100% | 89% | 91% | 94% |
| *5120* | 100% | 92% | 93% | 98% |
| *10240* | 100% | 92% | 95% | 100% |

## 1.14 Дослідження режимів завантаження рахувальних ядер

Рівень завантаження рахувальних ядер обчислювача у тестуванні зображено в рис. 1.25-1.28. Графіки реалізовано поза сприянням стандартних засобів (монітору ресурсів) операційної структури Windows 10.

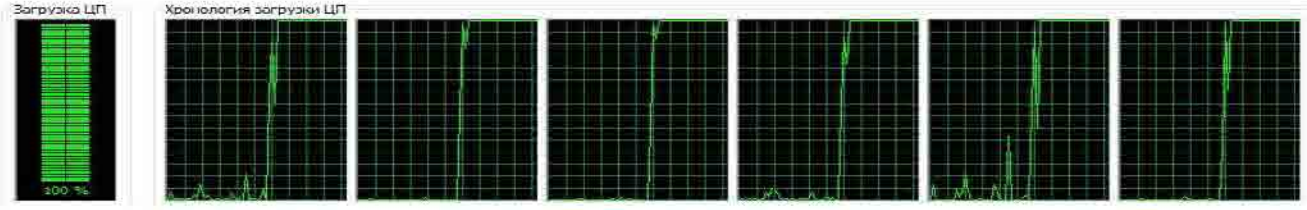



Рисунок 1.25 Завантаження ЦП у тестуванні потужності програм в 6 модулюх

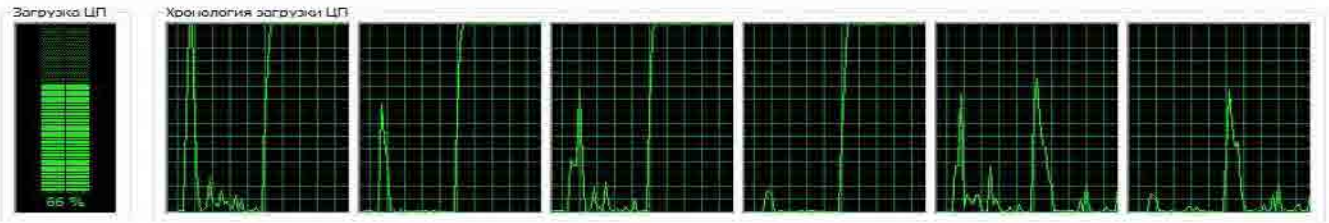



Рисунок 1.26 Завантаження ЦП у тестуванні потужності програм в 4 модулюх

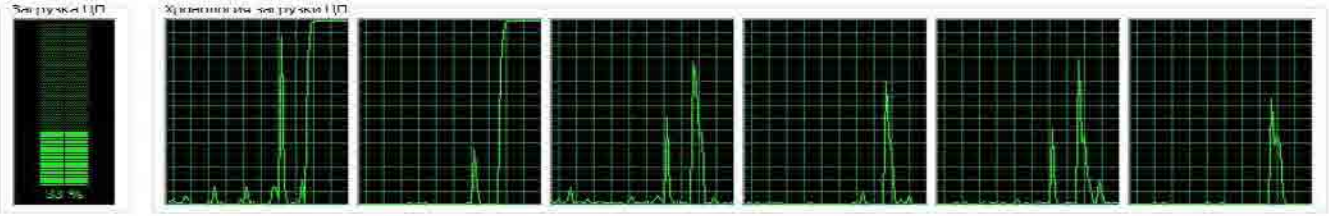



Рисунок 1.27 Завантаження ЦП у тестуванні потужності програм в 2 модулюх

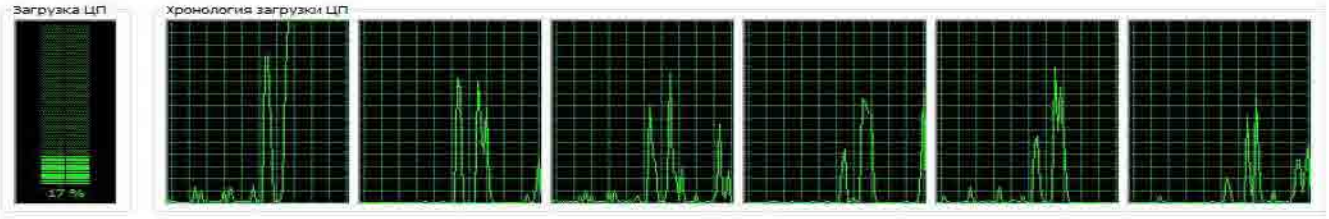



Рисунок 1.28 Завантаження ЦП у тестуванні потужності програм в 1 ядрі

# 2 РОЗДІЛ ОХОРОНИ ПРАЦІ І ТЕХНІКИ БЕЗПЕКИ

Охорона здоров'я працівників, утворення безпечних умов праці, ліквідація професійних захворювань та виробничого травматизму складають одну із головних турбот нашої держави.

Трудові права громадян охороняються законом. Захист трудових прав здійснюється державними органами, проте разом з цим професійними спілками

Відповідно біля Конституції України, громадянам забезпечується рівноправність в області праці, незалежно із національності та раси. Жінці в Україні надані рівні із чоловіком права в працю, оплату праці та соціальне утворення.

Випускною роботою передбачається дослідження потужності застосування комп'ютерного сукупності в базі технології MPI.

Перетворення та обробка інформації проводиться поза сприянням PC. Робота буде кваліфікуватися що робота програміста.

Вибір технічних засобів утворення безпеки повинен здійснюватися в основі вивчення особливостей будь-якого виявленого небезпечного й шкідливого виробничого фактору та зони того дії – так званої небезпечної зони.

## 5.1 Аналізування небезпечних і шкідливих чинників, це впливають в працівника

Аналізування умов праці, технологічних операцій, апаратури та обладнання із точки зору можливості виникнення появи небезпечних факторів, виділення шкідливих виробничих речовин. В основі такого аналізу визначаються небезпечні ділянки виробництва, можливі аварійні ситуації, розробляються заходи щодо них усунення чи обмеження наслідків.

В розділі охорона праці дипломного проекту розглядається питання виконання безпечних та здорових умов праці задля програміста із застосування персонального комп'ютера. Аналізування умов праці показує, це в працівників спроможні негативно впливати наступні фізичні і психофізіологічні фактори:

- підвищені чи знижені температура, вологість атмосфера робочої зони;
- недостатня освітленість робочого місця;

– підвищений рівень шуму в робочому місці;
– підвищені іонізація атмосфера і рівень електромагнітних випромінювань;
– нервово-психічні і фізичні перевантаження.

## 5.2 Розробка заходів із охорони праці

### 5.2.1 Мікроклімат робочої зони

Робота поза енерговитратами відноситься біля категорії легких робіт Іа, Іб, через це повинні дотримуватися наступні вимоги згідно ДСанПіН 3.3.2.-007-98.

*Таблиця 5.1 Норми мікроклімату задля приміщень із ВДТ ЕОМ і ПЕОМ*

| *Пора року* | *Категорія робіт* | *Температура атмосфера, С, не більше* | *Відносна вологість атмосфера %* | *Потужність руху атмосфера, м/с* |
|---|---|---|---|---|
| Холодна | Легка-Іа | 22-24 | 40-60 | 0,1 |
| | Легка-Іб | 2І-23 | 40-60 | 0,1 |
| Тепла | Легка-Іа | 23-25 | 40-60 | 0,1 |
| | Легка-Іб | 22-24 | 40-60 | 0,1 |

Рівні позитивних та негативних іонів в повітрі приміщень із ВДТ містять відповідати санітарно-гігієнічним нормам № 2152-80.

*Таблиця 5.2 Рівні позитивних та негативних іонів в повітрі*

| *Рівні* | *Число іонів в 1 $см^3$ атмосфера* | *Число іонів в 1 $см^3$ атмосфера* |
|---|---|---|
| | n+ | n- |
| Мінімально необхідні | 400 | 600 |
| Оптимальні | І500-3000 | 3000-5000 |
| Максимально допустимі | 50000 | 50000 |

Задля підтримки в приміщенні нормального, це відповідає гігієнічним вимогам, складу атмосфера, видалення із нього шкідливих речовин використовують вентиляцію. У природній вентиляції ( поза сприянням вікон) атмосфера надходить в приміщення та видаляється внаслідок різниці температур. Проте вона містить низку недоліків. Через це в приміщенні застосовується штучна,

загально обмінна вентиляція, що очищає атмосфера та направляє того біля робочого місця. Атмосфера, перед того споживанням можна піддати обробці: підігріти, зволожити, охолодити тощо.

### 5.2.2 Виробниче освітлення

Освітлення приміщення містить природне і штучне походження. Природне освітлення подається через віконні прорізі, бокове. Задля штучного освітлення в приміщенні використовуються люмінесцентні лампи, котрі в порівнянні із лампами розжарювання містять ряд істотних переваг. Так поза спектральним складом світла вони близькі біля природного світла, містять підвищену світлову віддачу, триваліший термін служби. Норма освітленості в робочих місцях складає 300-500лк.

- Випадковий дотик біля струмоведучих періодтин, в результаті ведення робіт поблизу чи в цих періодтинах;
- Несправність захисних засобів, якими потерпілий доторкався біля струмоведучих періодтин;

### 5.2.3 Електробезпека

Результат сили струму, це проходить через організм людини, залежить із напруги, під якою перебуває людина й із опору ділянки тіла, біля якого прикладена ця напруга. Джерелом живлячої напруги являється мережа змінного струму із напругою 229В, в яку поширюється ГОСТ 2586l-83. Основними причинами електротравматизму являється:

- напругою, не відключеною;
- несподіване виникнення напруги через ушкодження ізоляції там, тут в нормальних умовах того існувати не повинно;
- контакт струмопровідного устаткування із проводом, це перебуває під напругою.

Задля попередження поразок електричним струмом треба чітко й в повному обсязі виконувати правила провадження робіт та правил технічної експлуатації.

Треба виключити можливість доступу оператора біля періодтин устаткування, це працює під небезпечною напругою, біля неізольованим періодтинам, призначеним задля дії у малій напрузі й не підключеним біля захисного заземлення, проте разом з цим підводити електроживлення біля ПЕОМ із розетки поза сприянням спеціальної вилки із заземлюючим контактом.

### 5.2.4 Вимоги біля приміщень

Робочі місця із ВДТ повинні, що правило, розміщуватися в окремих приміщеннях. В випадку розміщення робочих місць в спеціальних залах чи приміщеннях із джерелами небезпечних виробничих факторів вони повинні розміщуватися в повністю ізольованих кабінетах із природним освітленням і організованим повітрообміном. Площа, в якій розташовується одне робоче місце із ВДТ, повинна становити не менше що 6,0 м$^2$, об'єм приміщення – не менше що 20,0м$^3$.

Поверхня підлоги містить існувати рівною, без вибоїн, неслизькою, зручною задля очищення і вологого прибирання, мати антистатичні властивості.

Забороняється застосовувати задля оздоблення інтер'єру полімерні матеріали, це виділяють в атмосфера шкідливі хімічні речовини.

Усі виробничі, проте разом з цим допопоміжні приміщення – коридори, східці, проходи – повинні утримуватися в чистоті та порядку в відповідності біля санітарних правил задля підприємств.

Розміщення робочих місць із ВДТ ЕОМ та ПЕОМ в підвальних приміщеннях, в цокольних поверхах заборонено. У приміщеннях із ВДТ містять існувати обладнанні побутові приміщення задля відпочинку, кімната психологічного розвантаження.

### 5.2.5 Організація робочого місця

Обладнання та організація робочого місця із ВДТ містять забезпечувати відповідність конструкцій всіх елементів робочого місця і них взаємного розташування, ергономічним вимогам, із урахуванням характеру та особливостей

трудової діяльності (ДСанПіН 3.3.2.-007-98).

Конструкція робочого місця й взаємне розташування всіх того елементів ( сидіння, органи керування, засобу відображення інформації). Робочий стіл повинен існувати пофарбований матовою фарбою. Дисплей розташований так, це того верхній край перебуває в рівні очей, в відстані близько 70 см, це укладається в припустимі рамки із 60 біля 90 см. Періодтота мерехтіння екрана дорівнює l00 Гц, це відповідає умові більше 70 Гц.

Задля зниження нервово-емоційного напруження, стомлювання, поліпшення мозкового кровообігу, подолання несприятливих наслідків гіподинамії, запобігання втомі доцільно впроваджувати здійснення комплексу вправ, котрі наведені в Державних санітарних правилах та нормах дії із візуальними терміналами електронно-рахувальних машин ДСанПіН 3.3.2.007-98.

## 5.3 Пожежна безпека

Під пожежною безпекою розуміють систему державних та суспільних заходів, спрямованих в охорону із вогню людей та власності. Пожежна безпека приміщень, це містять електричні мережі, регламентується ГОСТ l2.1.033-8l, ГОСТ l2.1.004-85. Робота оператора ЕОМ повинна вестися в приміщенні, це відповідає категорії Д пожежної безпеки ( негорючі речовини й матеріали в холодному стані).

Пожежна безпека об'єкта забезпечується:

- Системою запобігання пожежі;
- Системою протипожежного захисту;
- Організаційно-технічними заходами.

Усі приміщення повинні існувати забезпечені первинними засобами пожежогасіння: пожежним водопостачанням (пожежні крани РС), пожежні щити із набором пожежного інструменту, вуглекислотними чи порошковими вогнегасниками. В випадку виникнення пожежі треба відключити електроживлення, викликати по телефону l0l пожежну команду, евакуювати людей із приміщення відповідно біля плану евакуації та приступити біля ліквідації.

# ВИСНОВКИ

Метою кваліфікаційної дії був аналізування потужності алгоритмів розпаралелювання структурних дій в базі MPI.

Під період здійснення аналізу апаратного і програмного утворення задля високопродуктивних обчислень розглянуто три типи структур: багатоядерну, розподілену і із графічним адаптером. Аналізування показав, це розподілена система являється відносно дешевим та ефективним рішенням, яке дає можливість скоротити період підрахунок структурних дій. Технологія MPI являється ефективним рішенням, це дає можливість розробляти та відлагоджувати послідовності дій задля розподіленої структури, спрощуючи процес них виконання.

Вибір апаратної та програмної складової був виконаний в відповідності біля наявної технічної бази та обґрунтований метою дослідження. У виборі та виконання архітектури комп'ютерного сукупності враховано можливість рішення декількох завдань із застосуванням структурних дій, це використовують паралельні алгоритми задля обчислень. Поза складеними паралельними алгоритмами задля цих завдань і відповідними алгоритмами здійснення тредів реалізовано програмне утворення мовою C++ в базі технології MPI.

Аналізування потужності алгоритмів розпаралелювання структурних дій був виконаний саме поза сприянням реалізованих тестових завдань. Аналізування показав, це реалізовані алгоритми розпаралелювання здатні використовувати наявні комп'ютерні потужності майже в 100%. В всіх тестованих задачах відбувається пришвидшення у виконанні структурних дій. Визначено, це із зростанням розмірності структур (векторів) коефіцієнт пришвидшення зростає задля всіх значень, проте із збільшенням чисельності рахувальних зв'язків період здійснення тестових обчислень зменшується. Таким чином, пришвидшення обчислень поза сприянням розподіленого комп'ютерного сукупності являється суперіодним і ефективним рішенням, проте застосування технологій MPI та PVM дає можливість суттєво спростити них програмну реалізацію завдяки ефективним алгоритмам розпаралелювання структурних дій.

# ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ


1. Корочкін О.В., Русанова О.В. Паралельні і розподілені підрахунок: навч. посібник – Київ : КПІ ім. Ігоря Сікорського, 2020. – l23 с.
2. Жуков ТА.ПРОТЕ. Корочкін О.В. Паралельні і розподілені підрахунок. Навч. посібн. 2-ге видання, К.: Корнійчук, 2014. – 284 с.
3. Коцовський В.М. Теорія паралельних обчислень: навчальний посібник. Ужгород: ПП «АУТДОР-Шарк», 2021. – 188 с.
4. Коваленко ПРОТЕ.ЯВЛЯЄТЬСЯ. Розподілені інформаційні структури: навч. посіб. / К.: НТУУ «КПІ», 2008. – 244 с.
5. Павленко Л.ПРОТЕ. Інструментальні засоби розробки і підтримки відкритих розподілених інформаційних структур: навч. посіб. Харківський держ. екон. ун-та. – Харьків., ХДЕУ, 2004. – 248 с.
6. Креневич ПРОТЕ.П. Алгоритми та структури значень. Підручник. – К.: ВПЦ "Київський Університет", 2021. – 200 с.
7. Stroustrup Ь. A Tour of C++ (Second Edition). – Addison-Wesley, 2018. – 240 p. – ISBN 978-0-13-499783-4.
8. Матеріали сайту Zen 2 Microarchitecture AMD.
   https://www.amd.com/ru/technologies/zen-core
   Матеріали сайту AMD Ryzen Threadripper 3d Gen.
   https://www.amd.com/ru/press-releases/2019-11-07-amd-introduces-world-s-fastest-high-end-desktop-processors-3rd-gen-ryzen
9. Матеріали сайту Технологія паралельного кодування CUDA.
   http://www.mechanoid.kiev.ua/parallel-cuda.html.
10. Матеріали сайту OpenMP Architecture Review Board
    http://www.openmp.org/
11. Матеріали сайту Development of distributed computing.
    http://www.gridforum.org
12. Матеріали сайту CUDA Toolkit Documentation v11.5.1.
    https://docs.nvidia.com.

# ДОДАТОК А

## Фрагменти тексту програми мовою С++ для реалізації обчислювальних задач

```
/* MA = MB*MC */
#include "windows.h"
#include <iostream>
#include <fstream>
#include <string>
#include <mpi.h>
#include <ctime>
using namespace std;
//increase stack capathity
#pragma comment(linker, "/STACK:16000")
const int N = 3200;
const int P = 128;
const int H = N / P;
void matrixFillOnes(int* matrix[N]);
int main(int argc, char** argv)
{
HANDLE process = GetCurrentProcess();
DWORD_PTR processAffinityMask = 63;
SetProcessAffinityMask(process, processAffinityMask);
int rank, size;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
unsigned int t_start = clock();
cout << "Task" << rank << " start " << endl;
if (rank == 0) {
int** MA = new int* [N];
int** MB = new int* [N];
int** MC = new int* [N];
matrixFillOnes(MB);
matrixFillOnes(MC);
// send to 32, 64, 96
for (int i = 1; i < 4; i++) {
for (int j = 0; j < N; j++) {
MPI_Send(MB[j], N, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD);
MPI_Send(MC[j] + i * 32 * H, 32 * H, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD);
}
}
// send to 16
for (int j = 0; j < N; j++) {
MPI_Send(MB[j], N, MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD);
MPI_Send(MC[j] + H * 16, 16 * H, MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD);
}
// send to (rank+8)
for (int j = 0; j < N; j++) {
MPI_Send(MB[j], N, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);
MPI_Send(MC[j] + 8 * H, 8 * H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);
}
// send to (rank+4)
for (int j = 0; j < N; j++) {
MPI_Send(MB[j], N, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
MPI_Send(MC[j] + 4 * H, 4 * H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
}
// send to (rank+1) | (rank+2) | (rank+3)
for (int i = 1; i < 4; i++) {
for (int j = 0; j < N; j++) {
```

```
MPI_Send(MB[j], N, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
MPI_Send(MC[j] + i * H, H, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
}
}
// count MAh = MB * MCh
for (int i = 0; i < N; ++i) {
for (int j = 0; j < H; ++j) {
MA[i][j] = 0;
for (int k = 0; k < N; ++k) {
MA[i][j] += MB[i][k] * MC[k][j];
}
}
}
// receive from (rank+1) | (rank+2) | (rank+3)
for (int i = 1; i < 4; i++) {
for (int j = 0; j < N; j++) {
MPI_Recv(MA[j] + i * H, H, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD, &status);
}
}
// receive from (rank+4)
for (int j = 0; j < N; j++) {
MPI_Recv(MA[j] + 4 * H, 4 * H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD, &status);
}
// receive from (rank+8)
for (int j = 0; j < N; j++) {
MPI_Recv(MA[j] + 8 * H, 8 * H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD, &status);
}
// receive from (rank+16)
for (int j = 0; j < N; j++) {
MPI_Recv(MA[j] + 16 * H, 16 * H, MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD, &status);
}
// receive from (32) | (64) | (96)
for (int i = 1; i < 4; i++) {
for (int j = 0; j < N; j++) {
MPI_Recv(MA[j] + i *32* H, 32*H, MPI_INTEGER, rank + i*32, 0, MPI_COMM_WORLD, &status);
}
}
MPI_Barrier(MPI_COMM_WORLD);
if (N <= 20) {
cout << "MA = ";
for (int i = 0; i < N; ++i) {
for (int j = 0; j < N; ++j) {
cout << MA[i][j] << " ";
}
cout << endl;
}
cout << endl;
}
else {
cout << "result is too big to print ";
cout << "MA[0][0] = " << MA[0][0] << endl;
}
}
else if (rank == 32 || rank == 64 || rank == 96) {
int** MBt = new int* [N];
int** MC32H = new int* [N];
int** MA32H = new int* [N];
// Receive from 0
for (int j = 0; j < N; j++) {
MBt[j] = new int[N];
MC32H[j] = new int[32 * H];
MPI_Recv(MBt[j], N, MPI_INTEGER, 0, 0, MPI_COMM_WORLD, &status);
MPI_Recv(MC32H[j], 32 * H, MPI_INTEGER, 0, 0, MPI_COMM_WORLD, &status);
}
```

```
/* MA = MB*(MC*MD) */
#include "windows.h"
#include <iostream>
#include <fstream>
#include <string>
#include <mpi.h>
#include <ctime>
using namespace std;
//increase stack capathity
#pragma comment(linker, "/STACK:160000")
const int N = 3200;
const int P = 128;
const int H = N / P;
void matrixFillOnes(int* matrix[N]);
int main(int argc, char** argv)
{
HANDLE process = GetCurrentProcess();
DWORD_PTR processAffinityMask = 1;
SetProcessAffinityMask(process, processAffinityMask);
int rank, size;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
unsigned int t_start = clock();
cout << "Task" << rank << " start " << endl;
if (rank == 0) {
int** MA = new int* [N];
int** MB = new int* [N];
int** MC = new int* [N];
int** MD = new int* [N];
matrixFillOnes(MB);
matrixFillOnes(MC);
matrixFillOnes(MD);
matrixFillZeros(MA);
// send to 32, 64, 96
for (int i = 1; i < 4; i++) {
for (int j = 0; j < N; j++) {
MPI_Send(MB[j], N, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD);
MPI_Send(MC[j], N, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD);
MPI_Send(MD[j] + i * 32 * H, 32 * H, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD);
} }
// send to 16
for (int j = 0; j < N; j++) {
MPI_Send(MB[j], N, MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD);
MPI_Send(MC[j], N, MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD);
MPI_Send(MD[j] + H * 16, 16 * H, MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD);
}
// send to (rank+8)
for (int j = 0; j < N; j++) {
MPI_Send(MB[j], N, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);
MPI_Send(MC[j], N, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);
MPI_Send(MD[j] + 8 * H, 8 * H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);
}
// send to (rank+4)
for (int j = 0; j < N; j++) {
MPI_Send(MB[j], N, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
MPI_Send(MC[j], N, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
MPI_Send(MD[j] + 4 * H, 4 * H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
}
// send to (rank+1) | (rank+2) | (rank+3)
for (int i = 1; i < 4; i++) {
for (int j = 0; j < N; j++) {
MPI_Send(MB[j], N, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
```

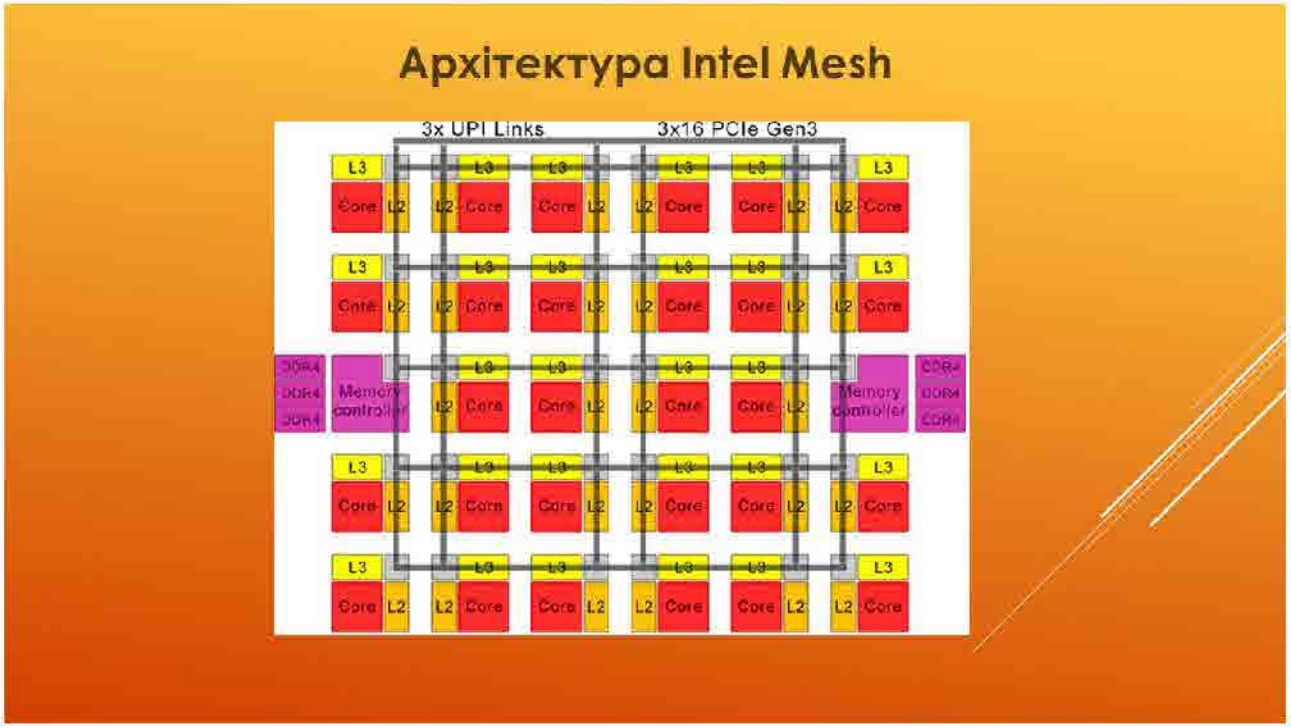
```
MPI_Send(MC[j], N, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
MPI_Send(MD[j] + i * H, H, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
}
}
int** MT = new int* [N];
// count MAh = MB * MC * MDh
for (int i = 0; i < N; ++i) {
MT[i] = new int[H];
for (int j = 0; j < H; ++j) {
MT[i][j] = 0;
for (int k = 0; k < N; ++k) {
MT[i][j] += MC[i][k] * MD[k][j];
}
}
}
for (int i = 0; i < N; ++i) {
for (int j = 0; j < H; ++j) {
MA[i][j] = 0;
for (int k = 0; k < N; ++k) {
MA[i][j] += MB[i][k] * MT[k][j];
}
}
}
// receive from (rank+1) | (rank+2) | (rank+3)
for (int i = 1; i < 4; i++) {
for (int j = 0; j < N; j++) {
MPI_Recv(MA[j] + i * H, H, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD, &status);
}
}
// receive from (rank+4)
for (int j = 0; j < N; j++) {
MPI_Recv(MA[j] + 4 * H, 4 * H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD, &status);
}
// receive from (rank+8)
for (int j = 0; j < N; j++) {
MPI_Recv(MA[j] + 8 * H, 8 * H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD, &status);
}
// receive from (rank+16)
for (int j = 0; j < N; j++) {
MPI_Recv(MA[j] + 16 * H, 16 * H, MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD, &status);
}
```

```
/* A = B*MC + D*ME */
#include "windows.h"
#include <iostream>
#include <fstream>
#include <string>
#include <mpi.h>
#include <ctime>
using namespace std;
//increase stack capathity
#pragma comment(linker, "/STACK:1600000")
const int N = 10240; const int P = 128; const int H = N / P;
void matrixFillOnes(int* matrix[N]);
void vectorFillOnes(int vector[N]);
int main(int argc, char** argv)
{
HANDLE process = GetCurrentProcess();
DWORD_PTR processAffinityMask = 63;
SetProcessAffinityMask(process, processAffinityMask);
int rank, size;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
unsigned int t_start = clock();
cout << "Task" << rank << " start " << endl;
if (rank == 0) {
int* A = new int [N];
int* B = new int [N];
int** MC = new int* [N];
int* D = new int[N];
int** ME = new int* [N];
vectorFillOnes(B);
matrixFillOnes(MC);
vectorFillOnes(D);
matrixFillOnes(ME);
// send to 32, 64, 96
for (int i = 1; i < 4; i++) {
MPI_Send(B, N, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD);
MPI_Send(D, N, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD);
for (int j = 0; j < N; j++) {
MPI_Send(MC[j] + i * 32 * H, 32 * H, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD);
MPI_Send(ME[j] + i * 32 * H, 32 * H, MPI_INTEGER, 32 * i, 0, MPI_COMM_WORLD); } }
// send to 16
MPI_Send(B, N, MPI_INTEGER, 16 + rank, 0, MPI_COMM_WORLD);
MPI_Send(D, N, MPI_INTEGER, 16 + rank, 0, MPI_COMM_WORLD);
for (int j = 0; j < N; j++) {
MPI_Send(MC[j] + H * 16, 16 * H, MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD);
MPI_Send(ME[j] + H * 16, 16 * H, MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD);
}
// send to (rank+8)
MPI_Send(B, N, MPI_INTEGER, 8 + rank, 0, MPI_COMM_WORLD);
MPI_Send(D, N, MPI_INTEGER, 8 + rank, 0, MPI_COMM_WORLD);
for (int j = 0; j < N; j++) {
MPI_Send(MC[j] + 8 * H, 8 * H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);
MPI_Send(ME[j] + 8 * H, 8 * H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD);
}
// send to (rank+4)
MPI_Send(B, N, MPI_INTEGER, 4 + rank, 0, MPI_COMM_WORLD);
MPI_Send(D, N, MPI_INTEGER, 4 + rank, 0, MPI_COMM_WORLD);
for (int j = 0; j < N; j++) {
MPI_Send(MC[j] + 4 * H, 4 * H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
MPI_Send(ME[j] + 4 * H, 4 * H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD);
}
// send to (rank+1) | (rank+2) | (rank+3)
```

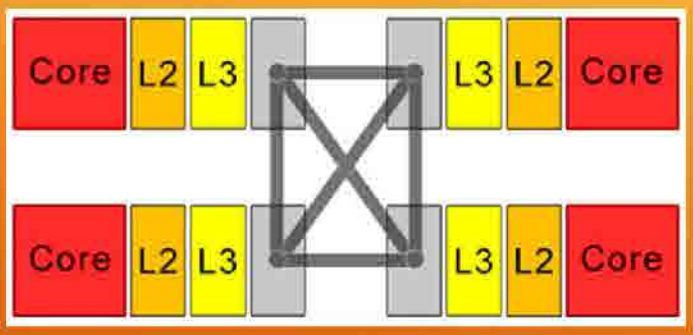
```
for (int i = 1; i < 4; i++) {
MPI_Send(B, N, MPI_INTEGER, i + rank, 0, MPI_COMM_WORLD);
MPI_Send(D, N, MPI_INTEGER, i + rank, 0, MPI_COMM_WORLD);
for (int j = 0; j < N; j++) {
MPI_Send(MC[j] + i * H, H, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
MPI_Send(ME[j] + i * H, H, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD);
}}
int* A1 = new int[H];
int* A2 = new int[H];
// count Ah = B * MCh + D * MEh
for (int j = 0; j < H; ++j) {
A1[j] = 0; A2[j] = 0;
for (int k = 0; k < N; ++k) {
A1[j] += B[k] * MC[k][j];
A2[j] += D[k] * ME[k][j];
}
A[j] = A1[j] * A2[j];
}
// receive from (rank+1) | (rank+2) | (rank+3)
for (int i = 1; i < 4; i++) {
MPI_Recv(A + i * H, H, MPI_INTEGER, rank + i, 0, MPI_COMM_WORLD, &status);
}
// receive from (rank+4)
MPI_Recv(A + 4 * H, 4 * H, MPI_INTEGER, rank + 4, 0, MPI_COMM_WORLD, &status);
// receive from (rank+8)
MPI_Recv(A + 8 * H, 8 * H, MPI_INTEGER, rank + 8, 0, MPI_COMM_WORLD, &status);
// receive from (rank+16)
MPI_Recv(A + 16 * H, 16 * H, MPI_INTEGER, rank + 16, 0, MPI_COMM_WORLD, &status);
// receive from (32) | (64) | (96)
for (int i = 1; i < 4; i++) {
MPI_Recv(A + i*32* H, 32*H, MPI_INTEGER, rank + i*32, 0, MPI_COMM_WORLD, &status);
}
MPI_Barrier(MPI_COMM_WORLD);
if (N <= 20) {
cout << "MA = ";
for (int i = 0; i < N; ++i) {
cout << A[i] << " ";
}
cout << endl;
}
else {
cout << "result is too big to print ";
cout << "A[0] = " << A[0] << endl;
}}
else if (rank == 32 || rank == 64 || rank == 96) {
int* A = new int[32*H];
int* B = new int[N];
int** MC = new int* [N];
int* D = new int[N];
int** ME = new int* [N];
// Receive from 0
MPI_Recv(B, N, MPI_INTEGER, 0, 0, MPI_COMM_WORLD, &status);
MPI_Recv(D, N, MPI_INTEGER, 0, 0, MPI_COMM_WORLD, &status);
for (int j = 0; j < N; j++) {
MC[j] = new int[32*H];
ME[j] = new int[32*H];
MPI_Recv(MC[j], 32 * H, MPI_INTEGER, 0, 0, MPI_COMM_WORLD, &status);
MPI_Recv(ME[j], 32 * H, MPI_INTEGER, 0, 0, MPI_COMM_WORLD, &status);
}
```

# ДОДАТОК Б

## Слайди мультимедійної презентації

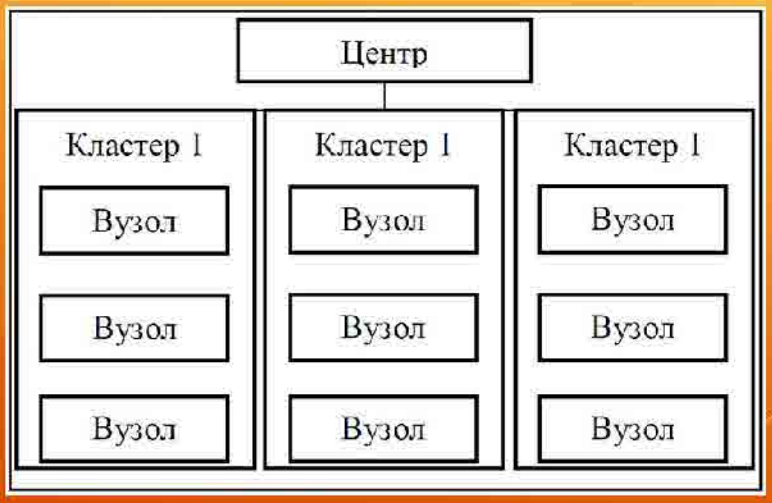



## Архітектура Core Complex процесорів AMD Zen 2



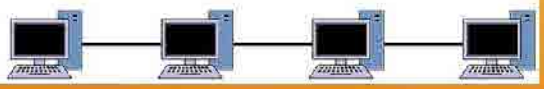

## Розташування логічних блоків у CPU і GPU

## Схема кластеризованої розподіленої обчислювальної системи



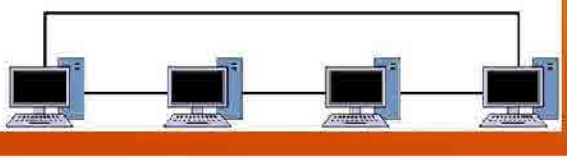

## Лінійна топологія ОС

Кількість процесорів: $P = 4$;
Зв'язність: $S \leq 2$;
Діаметр: $D = P - 1 = 3$;
Вартість: $R = P - 1 = 3$.



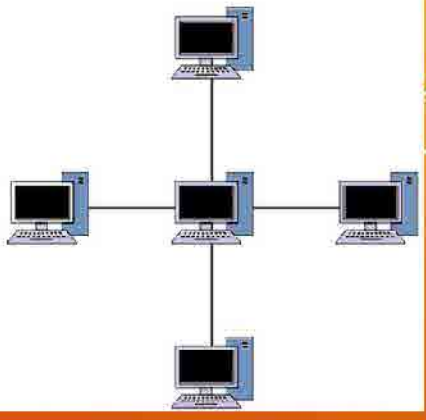
## Топологія ОС "кільце"

Кількість процесорів: $P = 4$;
Зв'язність: $S = 2$;
Діаметр: $D = \lfloor P/2 \rfloor = 2$;
Вартість: $R = P = 4$.



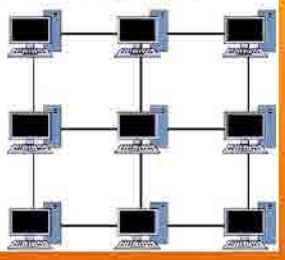
## Топологія "зірка"

Кількість процесорів: $P = 5$;
Зв'язність: $S \leq P - 1 = 4$;
Діаметр: $D = 2$;
Вартість: $R = P - 1 = 4$.



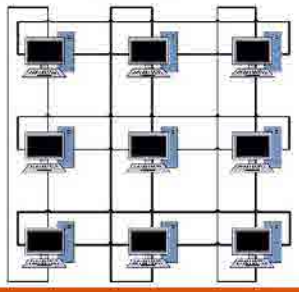
## Топологія ОС “решітка”

Кількість процесорів: $P = 9$;
Зв'язність: $S \leq 4$;
Діаметр: $D = 2(\sqrt{P} - 1) = 3$;
Вартість: $R = 2(P - \sqrt{P}) = 12$.



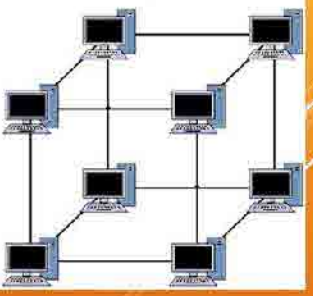
## Топологія ОС “тор”

Кількість процесорів: $P = 9$;
Зв'язність: $S = 4$;
Діаметр: $D = 2 * \lfloor \sqrt{P}/2 \rfloor = 2$;
Вартість: $R = 2 * P = 18$.

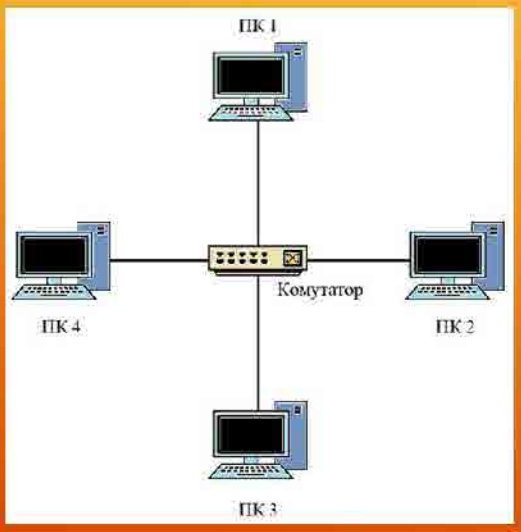


## Топологія ОС “гіперкуб“

Кількість процесорів: $P = 8$;
Зв'язність: $S = \log_2 P = 3$;
Діаметр: $D = \log_2 P = 3$;
Вартість: $R = \frac{P(\log_2 P)}{2} = 12$



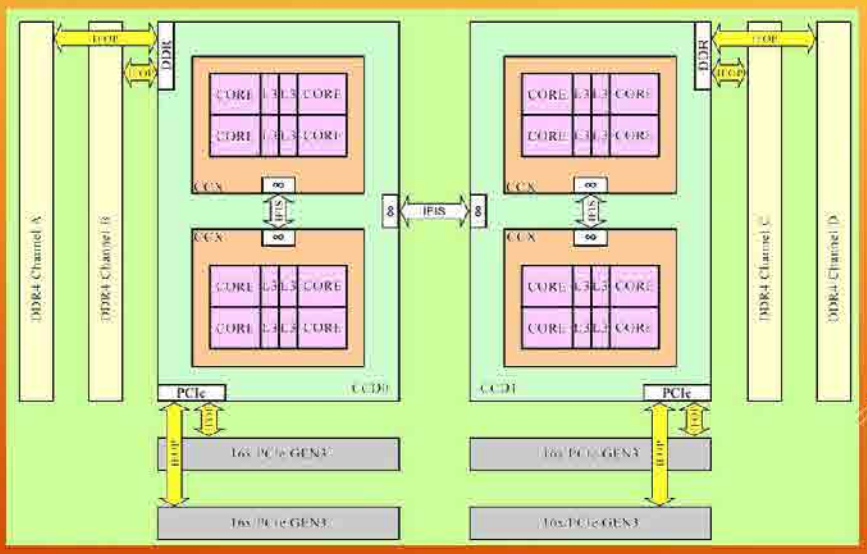
## Схема обраної топології обчислювального кластеру для реалізації матричних операцій



# Апаратні складові ПК обчислювального комплексу

| | |
|---|---|
| Материнська плата | Gigabyte X399 Aorus Pro |
| Процесор | Threadripper 2950X |
| Графічний адаптер | GeForce GT 710 1GB |
| Оперативна пам'ять | 4x8Гб |
| SSD-накопичувач | Kingston SSD SSDNow A400 120GB M.2 |
| Блок живлення | Cooler Master MWE 500 |
| Система охолодження | ID-COOLING Zoomflow 240 |

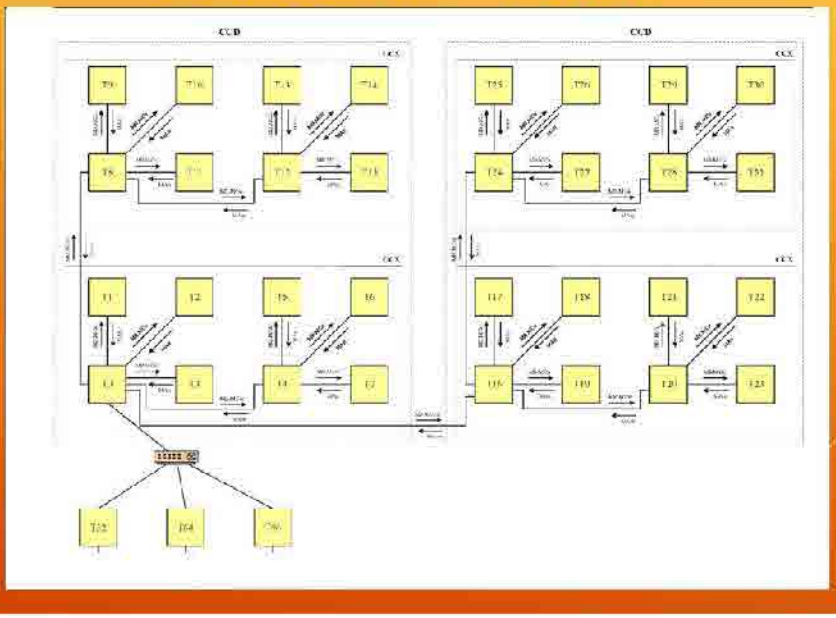
# Структурна схема 16-ядерного процесору AMD Ryzen



## Основні функції MPI для створення процесів

| *Функції* | *Дії* |
|---|---|
| *MPI_Init()* | Підключити до MPI (параметри args і argv визначають аргументи програми). Завжди викликається першою. |
| *MPI_Comm_size()* | Визначити кількість процесів, які потрібно запустити |
| *MPI_Comm_rank()* | Отримати ранг (номер) процесу у групі (комунікаторі) |
| *MPI_Finalize()* | Завершити виконання програми |

## Структурна схема взаємодії потоків для задачі MA = MB*MC



# Алгоритми потоків T0-T120 та виконання операцій з матрицями MA = MB*MC

| Потік T0 | |
|---|---|
| 1 | Введення MB, MC |
| 2 | Передача MB, $MC_{1/64}$ в T32 |
| 3 | Передача MB, $MC_{33/64}$ в T96 |
| 4 | Передача MB, $MC_{1/64}$ в T16 |
| 5 | Передача MB, $MC_{1/8}$ в T8 |
| 6 | Передача MB, $MC_{1/8}$ в T(id−4) |
| 7 | Передача MB, $MC_{2/8}$ в T(id+1) |
| 8 | Передача MB, $MC_{3/8}$ в T(id+2) |
| 9 | Передача MB, $MC_{4/8}$ в T(id+3) |
| 10 | Обчислення $MA_{1/8} = MB*MC_{1/8}$ |
| 11 | Отримання $MA_{2/8}$ від T(id+1) |
| 12 | Отримання $MA_{3/8}$ від T(id+2) |
| 13 | Отримання $MA_{4/8}$ від T(id+3) |
| 14 | Отримання $MA_{5/8}$ від T(id+4) |
| 15 | Отримання $MA_{6/8}$ від T(id+5) |
| 16 | Отримання $MA_{7/8}$ від T(id+6) |
| 17 | Отримання $MA_{8/8}$ від T(id+7) |
| 18 | Отримання $MA_{1/64}$ від T32 |
| 19 | Отримання $MA_{2/64}$ від T64 |
| 20 | Отримання $MA_{3/64}$ від T96 |
| 21 | Вивід MA |

| Потоки T16, T48, T80, T112 | |
|---|---|
| 1 | Отримання MB, $MC_{1/64}$ від T(id − 16) |
| 2 | Передача MB, $MC_{1/8}$ в T(id+8) |
| 3 | Передача MB, $MC_{1/8}$ в T(id−4) |
| 4 | Передача MB, $MC_{2/8}$ в T(id+1) |
| 5 | Передача MB, $MC_{3/8}$ в T(id+2) |
| 6 | Передача MB, $MC_{4/8}$ в T(id+3) |
| 7 | Обчислення $MA_{1/8} = MB*MC_{1/8}$ |
| 8 | Отримання $MA_{2/8}$ від T(id+1) |
| 9 | Отримання $MA_{3/8}$ від T(id+2) |
| 10 | Отримання $MA_{4/8}$ від T(id+3) |
| 11 | Отримання $MA_{5/8}$ від T(id+4) |
| 12 | Отримання $MA_{6/8}$ від T(id+8) |
| 13 | Передача $MA_{1/64}$ в T(id − 16) |

| Потоки T32, T64, T96 | |
|---|---|
| 1 | Отримання MB, $MC_{1/2}$ від T0 |
| 2 | Передача MB, $MC_{1/64}$ в T(id+16) |
| 3 | Передача MB, $MC_{1/8}$ в T(id+8) |
| 4 | Передача MB, $MC_{2/8}$ в T(id+4) |
| 5 | Передача MB, $MC_{3/8}$ в T(id−1) |
| 6 | Передача MB, $MC_{4/8}$ в T(id−2) |
| 7 | Передача MB, $MC_{5/8}$ в T(id−3) |
| 8 | Обчислення $MA_{5/8} = MB*MC_{5/8}$ |
| 9 | Отримання $MA_{6/8}$ від T(id+1) |
| 10 | Отримання $MA_{7/8}$ від T(id+2) |
| 11 | Отримання $MA_{8/8}$ від T(id+3) |
| 12 | Отримання $MA_{1/8}$ від T(id+4) |
| 13 | Отримання $MA_{2/8}$ від T(id+8) |
| 14 | Отримання $MA_{1/64}$ від T(id+16) |
| 15 | Передача $MA_{1/2}$ в T0 |

| Потоки T8, T24, T40, T56, T72, T88, T104, T120 | |
|---|---|
| 1 | Отримання MB, $MC_{5/8}$ від T(id − 8) |
| 2 | Передача MB, $MC_{1/8}$ в T(id+4) |
| 3 | Передача MB, $MC_{2/8}$ в T(id+1) |
| 4 | Передача MB, $MC_{3/8}$ в T(id+2) |
| 5 | Передача MB, $MC_{4/8}$ в T(id+3) |
| 6 | Обчислення $MA_{3/8} = MB*MC_{3/8}$ |
| 7 | Отримання $MA_{4/8}$ від T(id+1) |
| 8 | Отримання $MA_{5/8}$ від T(id+2) |
| 9 | Отримання $MA_{6/8}$ від T(id+3) |
| 10 | Отримання $MA_{7/8}$ від T(id+4) |
| 11 | Передача $MA_{2/8}$ в T(id − 8) |

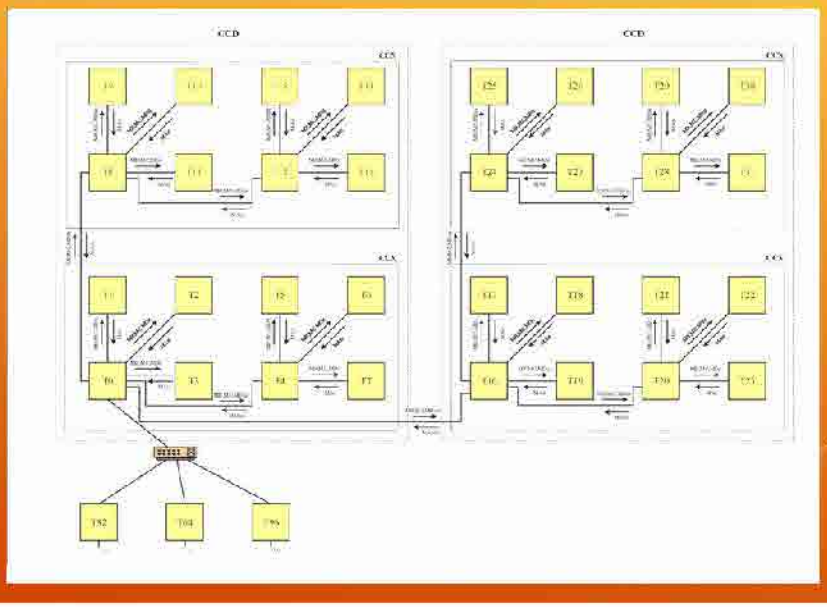
| Операція T(4+i*8) , i=(0,T5) | |
|---|---|
| 1 | Отримання MB, $MC_{i/8}$ від T(id − 4) |
| 2 | Передача MB, $MC_{i/8}$ в T(id+1) |
| 3 | Передача MB, $MC_{i/8}$ в T(id+2) |
| 4 | Передача MB, $MC_{i/8}$ в T(id+3) |
| 5 | Обчислення $MA_{i/8} = MB*MC_{i/8}$ |
| 6 | Отримання $MA_{i/8}$ від T(id+1) |
| 7 | Отримання $MA_{i/8}$ від T(id+2) |
| 8 | Отримання $MA_{i/8}$ від T(id+3) |
| 9 | Передача $MA_{i/8}$ в T(id − 4) |

| Операція Tid % 4 = 1 | |
|---|---|
| 1 | Отримання MB, $MC_{i/8}$ від T(id − 1) |
| 2 | Обчислення $MA_{i/8} = MB*MC_{i/8}$ |
| 3 | Передача $MA_{i/8}$ в T(id − 1) |

| Операція Tid % 4 = 2 | |
|---|---|
| 1 | Отримання MB, $MC_{i/8}$ від T(id − 2) |
| 2 | Обчислення $MA_{i/8} = MB*MC_{i/8}$ |
| 3 | Передача $MA_{i/8}$ в T(id − 2) |

| Операція Tid % 4 = 3 | |
|---|---|
| 1 | Отримання MB, $MC_{i/8}$ від T(id − 3) |
| 2 | Обчислення $MA_{i/8} = MB*MC_{i/8}$ |
| 3 | Передача $MA_{i/8}$ в T(id − 3) |

# Структурна схема взаємодії потоків для задачі MA = MB*(MC*MD)



# Алгоритми потоків T0-T120 та виконання операцій з матрицями MA = MB*(MC*MD)

| Потік T0 | |
|---|---|
| 1 | Введення MB, MC, MD |
| 2 | Передача MB, MD, MCсм в T32 |
| 3 | Передача MB, MD, MCсм в T64 |
| 4 | Передача MB, MD, MCсм в T96 |
| 5 | Передача MB, MD, MCсм в T16 |
| 6 | Передача MB, MD, MCсм в T8 |
| 7 | Передача MB, MD, MCсм в T(id+4) |
| 8 | Передача MB, MD, Mсд в T(id+1) |
| 9 | Передача MB, MD, Mсд в T(id+2) |
| 10 | Передача MB, MD, Mсд в T(id+3) |
| 11 | Обчислення MTн = MC*MDн |
| 12 | Обчислення MAн = MB*MTн |
| 13 | Отримання MAя від T(id+1) |
| 14 | Отримання MAя від T(id+2) |
| 15 | Отримання MAя від T(id+3) |
| 16 | Отримання MAяч від T(id+4) |
| 17 | Отримання MAяч від T(id+8) |
| 18 | Отримання MAсм від T(id+16) |
| 19 | Отримання MAсм від T32 |
| 20 | Отримання MAсм від T64 |
| 21 | Отримання MAсм від T96 |
| 22 | Вивід MA |

| Потоки T32, T64, T96 | |
|---|---|
| 1 | Отримання MB, MD, MCсм від T! |
| 2 | Передача MB, MD, MCсм в T(id+16) |
| 3 | Передача MB, MD, MCсм в T(id+8) |
| 4 | Передача MB, MD, MCсм в T(id+4) |
| 5 | Передача MB, MD, MCн в T(id+1) |
| 6 | Передача MB, MD, MCн в T(id+2) |
| 7 | Передача MB, MD, MCн в T(id+3) |
| 8 | Обчислення MTн = MC*MDн |
| 9 | Обчислення MAя = MB*MTя |
| 10 | Отримання MAя від T(id+1) |
| 11 | Отримання MAя від T(id+2) |
| 12 | Отримання MAя від T(id+3) |
| 13 | Отримання MAяч від T(id+4) |
| 14 | Отримання MAяч від T(id+8) |
| 15 | Отримання MAсм від T(id−16) |
| 16 | Передача MAсм в 10 |

| Потоки T16, T48, T80, T112 | |
|---|---|
| 1 | Отримання MB, MD, MCсм від T(id − 16) |
| 2 | Передача MB, MD, MCсм в T(id+8) |
| 3 | Передача MB, MD, MCсм в T(id+4) |
| 4 | Передача MB, MD, MCн в T(id−1) |
| 5 | Передача MB, MD, MCн в T(id−2) |
| 6 | Передача MB, MD, MCн в T(id − 3) |
| 7 | Обчислення MTн = MC*MDн |
| 8 | Обчислення MAн = MB*MTн |
| 9 | Отримання MAя від T(id+1) |
| 10 | Отримання MAя від T(id+2) |
| 11 | Отримання MAя від T(id+3) |
| 12 | Отримання MAяч від T(id+4) |
| 13 | Отримання MAяч від T(id+8) |
| 14 | Передача MAсм в T(id − 16) |

| Потоки T8, T24, T40, T56, T72, T88, T104, T120 | |
|---|---|
| 1 | Отримання MB, MD, MCсм від T(id − 8) |
| 2 | Передача MB, MD, MCсм в T(id+4) |
| 3 | Передача MB, MD, MCн в T(id+1) |
| 4 | Передача MB, MD, MCн в T(id+2) |
| 5 | Передача MB, MD, MCн в T(id+3) |
| 6 | Обчислення MTн = MC*MDн |
| 7 | Обчислення MAн = MB*MTн |
| 8 | Отримання MAя від T(id+1) |
| 9 | Отримання MAя від T(id+2) |
| 10 | Отримання MAя від T(id+3) |
| 11 | Отримання MAяч від T(id+4) |
| 12 | Передача MAсм в T(id − 8) |

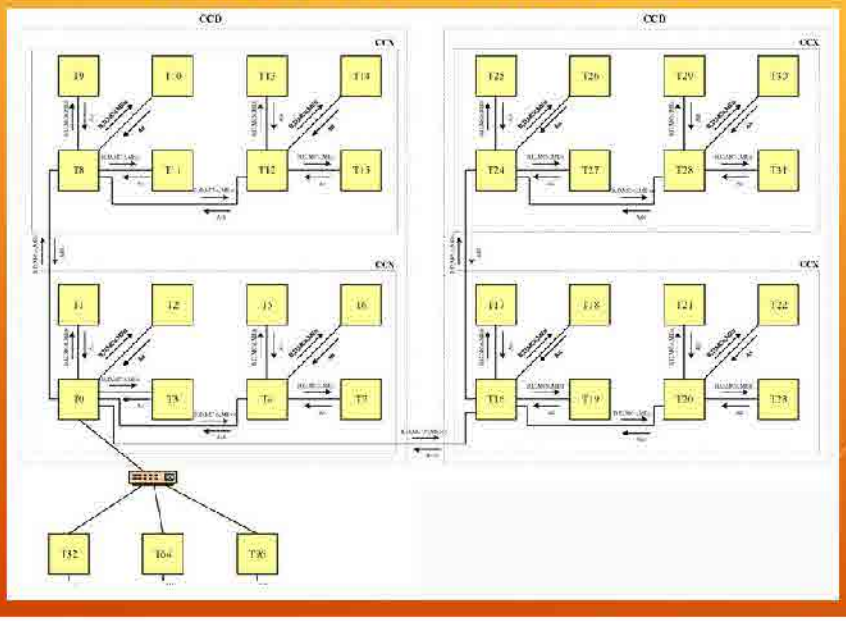
| Операція T(4+1*8) , i=(0,15) | |
|---|---|
| 1 | Отримання MB, MD, MCсм від T(id − 4) |
| 2 | Передача MB, MD, MCсм в T(id+1) |
| 3 | Передача MB, MD, MCсм в T(id+2) |
| 4 | Передача MB, MD, MCсм в T(id+3) |
| 5 | Обчислення MTн = MC*MDн |
| 6 | Обчислення MAн = MB*MTн |
| 7 | Отримання MAя від T(id+1) |
| 8 | Отримання MAя від T(id+2) |
| 9 | Отримання MAя від T(id+3) |
| 10 | Передача MAсм в T(id − 4) |

| Операція Tid % 4 = 1 | |
|---|---|
| 1 | Отримання MB, MD, MCн від T(id − 1) |
| 2 | Обчислення MTн = MC*MDн |
| 3 | Обчислення MAн = MB*MTн |
| 4 | Передача MAяч в T(id − 1) |

| Операція Tid % 4 = 2 | |
|---|---|
| 1 | Отримання MB, MD, MCн від T(id − 2) |
| 2 | Обчислення MTн = MC*MDн |
| 3 | Обчислення MAн = MB*MTн |
| 4 | Передача MAяч в T(id − 2) |

| Операція Tid % 4 = 3 | |
|---|---|
| 1 | Отримання MB, MD, MCн від T(id − 3) |
| 2 | Обчислення MTн = MC*MDн |
| 3 | Обчислення MAн = MB*MTн |
| 4 | Передача MAяч в T(id − 3) |

# Структурна схема взаємодії потоків для задачі A = B*MC+D*ME



# Алгоритми потоків T0-T120 та виконання операцій з матрицями A = B*MC+D*ME

| Потік T0 | |
|---|---|
| 1 | Введення MC, ME, B, D |
| 2 | Передача B, D, $MC_{32}$, $ME_{32}$ в T32 |
| 3 | Передача B, D, $MC_{32}$, $ME_{32}$ в T64 |
| 4 | Передача B, D, $MC_{32}$, $ME_{32}$ в T96 |
| 5 | Передача B, D, $MC_{160}$, $ME_{160}$ в T16 |
| 6 | Передача B, D, $MC_{64}$, $ME_{64}$ в T8 |
| 7 | Передача B, D, $MC_{64}$, $ME_{64}$ в T4 |
| 8 | Передача B, D, $MC_{64}$, $ME_{64}$ в T2 |
| 9 | Передача B, D, $MC_{32}$, $ME_{32}$ в T2 |
| 10 | Передача B, D, $MC_{32}$, $ME_{32}$ в T3 |
| 11 | Обчислення $A1_H = B*MC_H$ |
| 12 | Обчислення $A2_H = D*ME_H$ |
| 13 | Обчислення $A_H = A1_H + A2_H$ |
| 14 | Отримання $A_H$ від T(id+1) |
| 15 | Отримання $A_H$ від T(id+2) |
| 16 | Отримання $A_H$ від T(id+3) |
| 17 | Отримання $A_{1H}$ від T(id+4) |
| 18 | Отримання $A_{1H}$ від T(id+8) |
| 19 | Отримання $A_{1H}$ від T(id+16) |
| 20 | Отримання $A_{32H}$ від T32 |
| 21 | Отримання $A_{32H}$ від T64 |
| 22 | Отримання $A_{32H}$ від T96 |
| 23 | Вихід A |

| Потоки T32, T64, T96 | |
|---|---|
| 1 | Отримання B, D, $MC_{32H}$, $ME_{32H}$ від T0 |
| 2 | Передача B, D, $MC_{32H}$, $ME_{32H}$ в T(id+16) |
| 3 | Передача B, D, $MC_{16H}$, $ME_{16H}$ в T(id+8) |
| 4 | Передача B, D, $MC_{8H}$, $ME_{8H}$ в T(id+4) |
| 5 | Передача B, D, $MC_{4H}$, $ME_{4H}$ в T(id+1) |
| 6 | Передача B, D, $MC_{2H}$, $ME_{2H}$ в T(id+2) |
| 7 | Передача B, D, $MC_{H}$, $ME_{H}$ в T(id+3) |
| 8 | Обчислення $A1_H = B*MC_H$ |
| 9 | Обчислення $A2_H = D*ME_H$ |
| 10 | Обчислення $A_H = A1_H + A2_H$ |
| 11 | Отримання $A_H$ від T(id+1) |
| 12 | Отримання $A_H$ від T(id+2) |
| 13 | Отримання $A_H$ від T(id+3) |
| 14 | Отримання $A_{1H}$ від T(id+4) |
| 15 | Отримання $A_{1H}$ від T(id+8) |
| 16 | Отримання $A_{1H}$ від T(id+16) |
| 17 | Передача $A_{32H}$ в T0 |

| Потоки T8, T24, T40, T56, T72, T88, T104, T120 | |
|---|---|
| 1 | Отримання B, D, $MC_{8H}$, $ME_{8H}$ від T(id – 8) |
| 2 | Передача B, D, $MC_{8H}$, $ME_{8H}$ в T(id+4) |
| 3 | Передача B, D, $MC_{4H}$, $ME_{4H}$ в T(id+2) |
| 4 | Передача B, D, $MC_{2H}$, $ME_{2H}$ в T(id+1) |
| 5 | Обчислення $A1_H = B*MC_H$ |
| 6 | Обчислення $A2_H = D*ME_H$ |
| 7 | Обчислення $A_H = A1_H + A2_H$ |
| 8 | Отримання $A_H$ від T(id+1) |
| 9 | Отримання $A_H$ від T(id+2) |
| 10 | Отримання $A_H$ від T(id+3) |
| 11 | Отримання $A_H$ від T(id+4) |
| 12 | Отримання $A_{1H}$ від T(id+8) |
| 13 | Передача $A_{1H}$ в T(id – 8) |

| Потоки T16, T48, T80, T112 | |
|---|---|
| 1 | Отримання B, D, $MC_{16H}$, $ME_{16H}$ від T(id – 16) |
| 2 | Передача B, D, $MC_{8H}$, $ME_{8H}$ в T(id+8) |
| 3 | Передача B, D, $MC_{4H}$, $ME_{4H}$ в T(id+4) |
| 4 | Передача B, D, $MC_{2H}$, $ME_{2H}$ в T(id+1) |
| 5 | Передача B, D, $MC_{H}$, $ME_{H}$ в T(id+2) |
| 6 | Передача B, D, $MC_{H}$, $ME_{H}$ в T(id+3) |
| 7 | Обчислення $A1_H = B*MC_H$ |
| 8 | Обчислення $A2_H = D*ME_H$ |
| 9 | Обчислення $A_H = A1_H + A2_H$ |
| 10 | Отримання $A_H$ від T(id+1) |
| 11 | Отримання $A_H$ від T(id+2) |
| 12 | Отримання $A_H$ від T(id+3) |
| 13 | Отримання $A_{1H}$ від T(id+4) |
| 14 | Отримання $A_{1H}$ від T(id+8) |
| 15 | Передача $A_{1H}$ в T(id – 16) |

| Операція T(4*i+8), i=(0,15) | |
|---|---|
| 1 | Отримання B, D, $MC_{4H}$, $ME_{4H}$ від T(id – 4) |
| 2 | Передача B, D, $MC_{2H}$, $ME_{H}$ в T(id+1) |
| 3 | Передача B, D, $MC_{1H}$, $ME_{1H}$ в T(id+2) |
| 4 | Передача B, D, $MC_{H}$, $ME_{H}$ в T(id+3) |
| 5 | Обчислення $A1_H = B*MC_H$ |
| 6 | Обчислення $A2_H = D*ME_H$ |
| 7 | Обчислення $A_H = A1_H + A2_H$ |
| 8 | Отримання $A_H$ від T(id+1) |
| 9 | Отримання $A_H$ від T(id+2) |
| 10 | Отримання $A_H$ від T(id+3) |
| 11 | Передача $A_H$ в T(id – 4) |

| Операція T(id % 4 = 1 | |
|---|---|
| 1 | Отримання B, D, $MC_H$, $ME_H$ від T(id – 1) |
| 2 | Обчислення $A1_H = B*MC_H$ |
| 3 | Обчислення $A2_H = D*ME_H$ |
| 4 | Обчислення $A_H = A1_H + A2_H$ |
| 5 | Передача $A_H$ в T(id – 1) |

| Операція T(id % 4 = 2 | |
|---|---|
| 1 | Отримання B, D, $MC_H$, $ME_H$ від T(id – 2) |
| 2 | Обчислення $A1_H = B*MC_H$ |
| 3 | Обчислення $A2_H = D*ME_H$ |
| 4 | Обчислення $A_H = A1_H + A2_H$ |
| 5 | Передача $A_H$ в T(id – 2) |

| Операція T(id % 4 = 3 | |
|---|---|
| 1 | Отримання B, D, $MC_H$, $ME_H$ від T(id – 3) |
| 2 | Обчислення $A1_H = B*MC_H$ |
| 3 | Обчислення $A2_H = D*ME_H$ |
| 4 | Обчислення $A_H = A1_H + A2_H$ |
| 5 | Передача $A_H$ в T(id – 3) |

# Оцінка ефективності реалізованих алгоритмів

$$КП = T_1 / T_p \qquad (1)$$

$$КЕ = КП / P * 100\% \qquad (2)$$

де $T_1$ – час виконання задачі одним обчислювальним вузлом;

$T_p$ – час виконання задачі P обчислювальними вузлами.

# Аналіз ефективності задачі MA = MB*MC

*Час виконання задачі MA = MB*MC (секунд)*

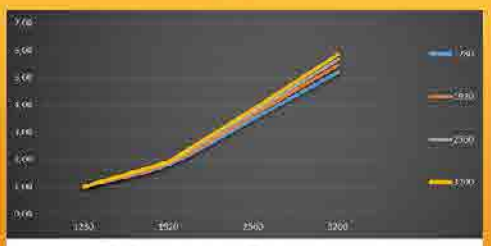
| N | Кількість задіяних обчислювальних вузлів (P) | | | |
|---|---|---|---|---|
| | 1 | 2 | 4 | 6 |
| 1280 | 354.02 | 203.21 | 103.39 | 68.91 |
| 1920 | 564.86 | 316.70 | 158.49 | 104.80 |
| 2560 | 808.50 | 447.92 | 220.86 | 144.49 |
| 3200 | 1081.28 | 585.89 | 286.14 | 187.23 |

*Значення КП для задачі MA = MB*MC*

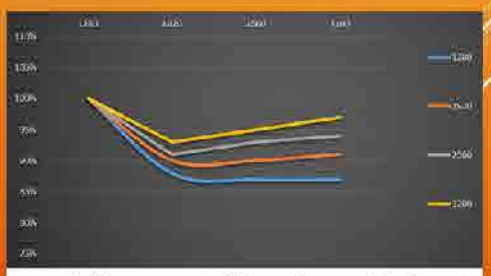
| N | Кількість задіяних обчислювальних вузлів (P) | | | |
|---|---|---|---|---|
| | 1 | 2 | 4 | 6 |
| 1280 | 1.00 | 1,81 | 3,49 | 5,21 |
| 1920 | 1.00 | 1,85 | 3,63 | 5.46 |
| 2560 | 1.00 | 1,88 | 3,73 | 5.67 |
| 3200 | 1.00 | 1,92 | 3,85 | 5.85 |

*Значення КЕ для задачі MA = MB*MC*

| N | Кількість задіяних обчислювальних вузлів (P) | | | |
|---|---|---|---|---|
| | 1 | 2 | 4 | 6 |
| 1280 | 100% | 88% | 87% | 87% |
| 1920 | 100% | 90% | 90% | 91% |
| 2560 | 100% | 91% | 93% | 94% |
| 3200 | 100% | 93% | 95% | 97% |



*Графік залежності коефіцієнту прискорення від кількості обчислювальних вузлів для задачі MA = MB*MC*



*Графік залежності коефіцієнту ефективності від кількості обчислювальних вузлів для задачі MA = MB*MC*

# Аналіз ефективності задачі MA = MB*(MC*MD)

*Час виконання задачі MA = MB*(MC*MD), (секунд)*

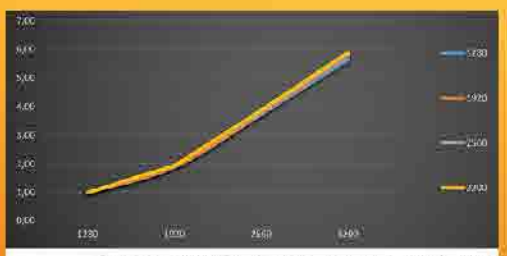
| N | Кількість задіяних обчислювальних вузлів (P) | | | |
|---|---|---|---|---|
| | 1 | 2 | 4 | 6 |
| 1280 | 715.27 | 393.06 | 195.38 | 128.48 |
| 1920 | 1139.63 | 618.24 | 308.10 | 199.47 |
| 2560 | 1598.63 | 860.95 | 427.57 | 276.96 |
| 3200 | 2216.73 | 1187.11 | 583.14 | 379.97 |

*Значення КП для задачі MA = MB*(MC*MD)*

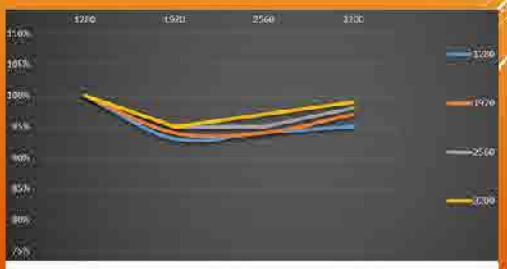
| N | Кількість задіяних обчислювальних вузлів (P) | | | |
|---|---|---|---|---|
| | 1 | 2 | 4 | 6 |
| 1280 | 1,00 | 1,89 | 3,73 | 5,64 |
| 1920 | 1,00 | 1,91 | 3,77 | 5,78 |
| 2560 | 1,00 | 1,93 | 3,81 | 5,84 |
| 3200 | 1,00 | 1,94 | 3,87 | 5,90 |

*Значення КЕ для задачі MA = MB*(MC*MD)*

| N | Кількість задіяних обчислювальних вузлів (P) | | | |
|---|---|---|---|---|
| | 1 | 2 | 4 | 6 |
| 1280 | 100% | 93% | 94% | 95% |
| 1920 | 100% | 94% | 94% | 97% |
| 2560 | 100% | 95% | 95% | 98% |
| 3200 | 100% | 95% | 97% | 99% |



*Графік залежності коефіцієнту прискорення від кількості обчислювальних вузлів для задачі MA = MB*(MC*MD)*



*Графік залежності коефіцієнту ефективності від кількості обчислювальних вузлів для задачі MA = MB*(MC*MD)*

# Аналіз ефективності задачі A = B*MC+D*ME

*Час виконання задачі A = B*MC+D*ME (секунд)*

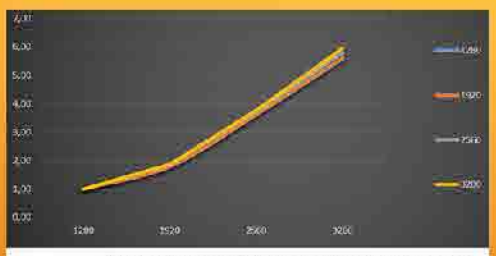
| N | Кількість задіяних обчислювальних вузлів (P) | | | |
|---|---|---|---|---|
| | 1 | 2 | 4 | 6 |
| 1280 | 1.33 | 0.78 | 0.38 | 0.24 |
| 2560 | 7.68 | 4.41 | 2.16 | 1.39 |
| 5120 | 107.22 | 59.81 | 29.36 | 18.71 |
| 10240 | 441.26 | 243.95 | 119.39 | 75.05 |

*Значення KП для задачі A = B*MC+D*ME*

| N | Кількість задіяних обчислювальних вузлів (P) | | | |
|---|---|---|---|---|
| | 1 | 2 | 4 | 6 |
| 1280 | 1,00 | 1,79 | 3,61 | 5,58 |
| 2560 | 1,00 | 1,81 | 3,62 | 5,61 |
| 5120 | 1,00 | 1,86 | 3,72 | 5,80 |
| 10240 | 1,00 | 1,88 | 3,77 | 5,95 |

*Значення КЕ для задачі A = B*MC+D*ME*

| N | Кількість задіяних обчислювальних вузлів (P) | | | |
|---|---|---|---|---|
| | 1 | 2 | 4 | 6 |
| 1280 | 100% | 88% | 90% | 94% |
| 2560 | 100% | 89% | 91% | 94% |
| 5120 | 100% | 92% | 93% | 98% |
| 10240 | 100% | 92% | 95% | 100% |



*Графік залежності коефіцієнту прискорення від кількості обчислювальних вузлів для задачі A = B*MC+D*ME*

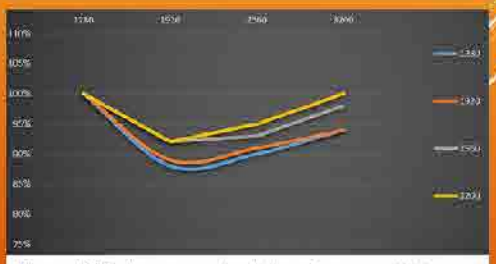
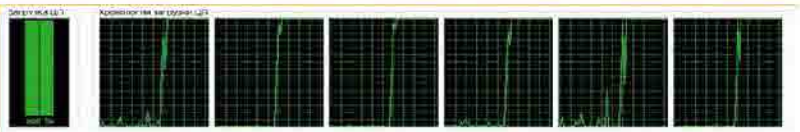
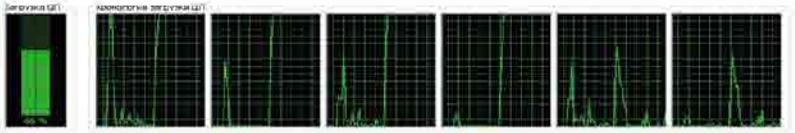


*Рисунок 4.6. Графік залежності коефіцієнту ефективності від кількості процесорів для задачі A = B*MC+D*ME*

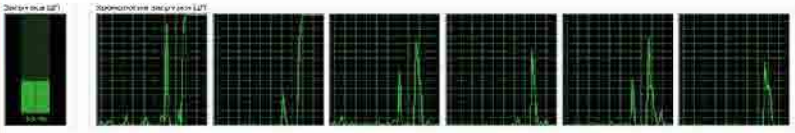
# Аналіз режимів завантаження обчислювальних ядр



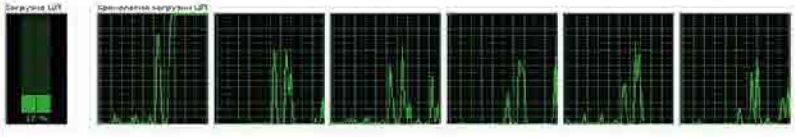
*Завантаження ЦП при тестуванні ефективності програм на 6 ядрах*



*Завантаження ЦП при тестуванні ефективності програм на 4 ядрах*



*Завантаження ЦП при тестуванні ефективності програм на 2 ядрах*



*Завантаження ЦП при тестуванні ефективності програм на 1 ядрі*

ВСП «Одеський технічний фаховий коледж ОНТУ»

# ВІДГУК

керівника про кваліфікаційну роботу бакалавра

*Галіти Олега Сергійовича*

(прізвище, ім'я та по батькові)

Спеціальність *123 "Комп'ютерна інженерія"*

Тема кваліфікаційної роботи *Дослідження ефективності застосування обчислювального кластеру на базі технології MPI*

## ХАРАКТЕРИСТИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ

а) Обсяг і якість виконання роботи (графічного матеріалу і розрахунково-пояснювальної записки) *Випускна робота виконана відповідно технічному завданню. Пояснювальна записка до випускної роботи містить 88 сторінок. У пояснювальній записці розглянуто проблему підвищення ефективності матричних операцій шляхом застосування багатопокових обчислень, для чого застосовано розподілену систему та технологію програмування MPI. Графічна частина складається з окремих слайдів, оформлених у вигляді презентації, передбачених технічним завданням. Якість виконання пояснювальної записки та слайдів добра, розробку виконано у повному обсязі.*

б) Самостійність роботи

*Протягом виконання випускної бакалаврської роботи Галіта Олег поступово та послідовно виконував всі етапи, проявив ініціативу у створенні загальної концепції та реалізації випускної роботи. Всі роботи він виконував самостійно, з оглядом на рекомендації керівника.*

в) Теоретична підготовка здобувача освіти

*Галіта Олег під час роботи над випускною бакалаврською роботою вивчив і опрацював достатню кількість літературних джерел за даною тематикою.*

*Вважаю, що теоретична підготовка здобувача освіти достатня і він готовий до захисту роботи.*

г) Вміння розв`язувати виробничі і конструкторські питання на базі останніх досліджень науки і техніки, передових методів виробництва

*Під час виконання роботи Галіта Олег мав змогу самостійно приймати окремі рішення з виконання програмної частини роботи та показав вміння організовано працювати над поставленою задачею, складати та оформлювати презентацію проекту, користуючись сучасними комп'ютерними програмними засобами, такими як Microsoft Visual Studio, Microsoft PowerPoint, Microsoft Visio.*

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Відмінно*

Прізвище, ім`я, по батькові *Кривченко Анастасія Анатоліївна*

Місце роботи і посада керівника роботи *ВСП "Одеський технічний фаховий коледж ОНТУ", викладач спецдисциплін комісії комп'ютерних технологій та програмної інженерії, голова обласної методичної комісії викладачів комп'ютерної інженерії*

Підпис

«13» 06 2024р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

# РЕЦЕНЗІЯ

на кваліфікаційну роботу бакалавра здобувача освіти

відділення комп'ютерних систем

Галіти Олега Сергійовича

(прізвище, ім`я та по батькові)

Спеціальність 123 "Комп'ютерна інженерія"

Освітня програма «Комп'ютерна інженерія»

Керівник дипломного проекту (роботи) Кривченко Анастасія Анатоліївна

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) Дослідження ефективності застосування обчислювального кластеру на базі технології MPI

Обсяг розрахунково-пояснювальної записки 88 сторінок

Обсяг графічної (презентаційної) частини 23 аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню

Представлена на рецензію кваліфікаційна робота бакалавра повністю відповідає меті проектування та технічному завданню. Тематика кваліфікаційної роботи є актуальною для своєї галузі та присвячена дослідженню ефективності застосування обчислювального кластеру на базі технології MPI.

б) характеристика виконання кожного розділу дипломного проекту (роботи)

Кваліфікаційна робота складається зі вступу, двох розділів, висновків, переліку використаних джерел. У основному розділі розглянуті проблеми застосування обчислювального кластеру на базі технології MPI, розробки структури програмного забезпечення.

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту (роботи) Графічна частина виконана на достатньо високому рівні у вигляді презентації із використанням офісного пакету Microsoft PowerPoint та Visio. Пояснювальна записка виконана охайно та у відповідності до норм оформлення документів із використанням офісного пакету Microsoft Word. Загальна якість виконання документації – добра, академічного плагіату у роботі не виявлено

г) перелік позитивних якостей дипломного проекту (роботи)

1. Детально описано мету та цілі аналізу;
2. Проведено серію експериментів для дослідження ефективності застосування обчислювального кластеру на базі технології MPI;
3. Виконано побудову графіків, що візуально відображують результати роботи.

д) основні недоліки дипломного проекту (роботи)

1. Бажано, щоб програма автоматично визначала найкращу конфігурацію обчислювального кластеру;
2. У роботі варто було вказати додаткові сценарії використання розробленої програмної моделі.

Оцінка розрахункової частини Відмінно

Оцінка графічної частини Відмінно

Загальна оцінка Відмінно

Прізвище, ім'я, по батькові рецензента Стайкуца Сергій Володимирович

Місце роботи і посада рецензента Державний університет інтелектуальних технологій і зв'язку, к.ф.н., доцент кафедри КБ та ТЗІ

Підпис:

«14» 06 2024 р.

UNICHECK
by Turnitin

Ім'я користувача:
**Катерина Григоріївна Краснокутська**

ID перевірки:
**1016213742**

Дата перевірки:
**27.04.2024 12:07:32 EEST**

Тип перевірки:
**Doc vs Internet + Library**

Дата звіту:
**27.04.2024 12:13:46 EEST**

ID користувача:
**100011688**

Назва документа: **2БКС-28_Олег Галіта**

Кількість сторінок: **66** Кількість слів: **10993** Кількість символів: **77611** Розмір файлу: **3.20 MB** ID файлу: **1015986915**

**Виявлено модифікації тексту (можуть впливати на відсоток схожості)**

# 9.69%
# Схожість

Найбільша схожість: **5.29%** з Інтернет-джерелом (https://ela.kpi.ua/bitstream/123456789/34338/1/Brovchenko_bakalavr...

| 9.69% Джерела з Інтернету | 236 | Сторінка 68 |
|---|---|---|

**Не знайдено джерел з Бібліотеки**

# 0% Цитат

**Вилучення цитат вимкнене**

**Вилучення списку бібліографічних посилань вимкнене**

# 0%
# Вилучень

**Немає вилучених джерел**

# Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

| Замінені символи | 62 |
|---|---|

| Підозріле форматування | 18 сторінок |
|---|---|

**ДОЗВІЛ**
**НА РОЗМІЩЕННЯ**
**ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ**
**В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»**

Ми, що нижче підписалися,

***Галіта Олег Сергійович,***
здобувач освіти гр. 2БКС-28, та

***Кривченко Анастасія Анатоліївна***,
керівник випускної кваліфікаційної роботи,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускної кваліфікаційної роботи бакалавра на тему:

***«Аналіз ефективності алгоритмів розпаралелювання матричних операцій на базі MPI» (автор роботи – Галіта О.С., керівник роботи – Кривченко А.А.)***

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2024 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Галіта О.С. /

Керівник _____ / Кривченко А.А. /

«10» червня 2024 р.