НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра обчислювальної техніки

**КУРСОВА РОБОТА**

з дисципліни «Паралельні та розподілені обчислення»

на тему: «Розробка програмного забезпечення для паралельних

комп’ютерних систем»

Студента 3 курсу групи ІО-01

напряму підготовки 050102

«Комп’ютерна інженерія»

Ткаченка Ігоря Миколайовича

Керівник доцент Корочкін О.В.

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали

Київ - 2013 рік

Національний технічний університет України

«Київський політехнічний інститут»

Факультет (інститут) інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 6.050102 «Комп’ютерна інженерія»

***З А В Д А Н Н Я***

НА КУРСОВУ РОБОТУ СТУДЕНТУ

Ткаченку Ігорю Миколайовичу

1. Тема роботи «Розробка програмного забезпечення для паралельних комп’ютерних систем»

керівник роботи Корочкін Олександр Володимирович к.т.н., доцент

2. Строк подання студентом роботи 13 травня 2013 р.

3. Вихідні дані до роботи

- Порівняння реалізації механізму «критичні секції» в

мовах і бібліотеках паралельного програмуванняя - математична задача A = B(MO\*MX)+α\*Z

- структури ПКС СП та ПКС ЛП

- мови програмування: OpenMP, Ada Рандеву

- засоби організації взаємодії процесів: бар’єри та критичні секції, замки бібліотеки OpenMP, механізм рандеву мови Ada

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно

розробити)

- Порівняння реалізації механізму «критичні секції» в

мовах і бібліотеках паралельного програмуванняя

- розробка і тестування програми ПРГ1 для ПКС СП



Рис. 1.1. Схема ПКС СП

- розробка і тестування програми ПРГ2 для ПКС ЛП



Рис. 1.2. Схема ПКС ЛП

5. Перелік графічного матеріалу

- структурна схема ПКС СП

- структурна схема ПКС ЛП

- схеми алгоритмів процесів і головної програми для ПРГ1

- схеми алгоритмів процесів і головної програми для ПРГ2.

6. Дата видачі завдання \_\_\_\_\_\_\_\_\_\_\_\_\_

*КАЛЕНДАРНИЙ ПЛАН*

|  |  |  |
| --- | --- | --- |
| №  з/п | Назва етапів виконання КР | Строк виконання етапів КР |
| 1 | Виконання огляду для розділу 1 | 20.03.2013 |
| 2 | Розробка паралельного алгоритму  рішення задачі | 1.04.2013 |
| 3 | Розробка алгоритмів процесів | 6.04.2014 |
| 4 | Розробка схем взаємодії процесів | 13.04.2013 |
| 5 | Розробка програм | 20.04.2013 |
| 6 | Тестування програм | 30.04.2013 |
| 7 | Оформлення КР | 10.05.2013 |
| 8 | Захист КР | 18.05.2013 |

**Студент** \_\_\_\_\_\_\_\_\_ Ткаченко І.М.

( підпис )

**Керівник роботи** \_\_\_\_\_\_\_\_\_ Корочкін О.В

. ( підпис )

Зміст

[Розділ 1 . Огляд механізму критичні секції 7](#_Toc356181034)

[1. 1 Загальні теоретичні відомості 7](#_Toc356181035)

[1. 2 Завдання взаємного виключення. 7](#_Toc356181036)

[1.3 Завдання синхронізації процесів. 9](#_Toc356181037)

[1. 4 Поняття критична секція 9](#_Toc356181038)

[1. 5 Win32 10](#_Toc356181039)

[1. 6 C++ 12](#_Toc356181040)

[1. 7 Delphi 12](#_Toc356181041)

[1. 8 Java 14](#_Toc356181042)

[1. 9 OpenMp 16](#_Toc356181043)

[1. 10 OpenMp в С++ 17](#_Toc356181044)

[1. 11 OpenMР в Fortran. 17](#_Toc356181045)

[1. 12 C# 18](#_Toc356181046)

[1. 13 Python 20](#_Toc356181047)

[Висновки до розділу 1 23](#_Toc356181048)

[РОЗДІЛ 2. Розробка програми ПРГ1 для ПКС СП 24](#_Toc356181049)

[2.1 Аналіз задачі на внутрішній паралелізм з використанням концепції необмеженого паралелізму. 24](#_Toc356181050)

[2.2 Розробка паралельного математичного алгоритму 26](#_Toc356181051)

[2.3 Розробка алгоритмів процесів 26](#_Toc356181052)

[2.4 Розробка схеми взаємодії процесів 29](#_Toc356181053)

[2.5 Розробка програми ПРГ1 29](#_Toc356181054)

[2.6 Тестування програми ПРГ1 30](#_Toc356181055)

[Висновки до розділу 2 36](#_Toc356181056)

[РОЗДІЛ 3. Розробка програми ПРГ2 для ПКС ЛП 37](#_Toc356181057)

[3.1 Розробка паралельного математичного алгоритму 37](#_Toc356181058)

[3.2 Розробка алгоритмів процесів 37](#_Toc356181059)

[3.3 Розробка схеми взаємодії процесів 40](#_Toc356181060)

[3.4 Розробка програми ПРГ2 40](#_Toc356181061)

[3.5 Тестування програми ПРГ2 40](#_Toc356181062)

[Аналогічно на основі таблиці 3.3 проаналізуємо зміну значень коефіцієнту ефективності Ке залежно від розмірності матриць N та кількості процесорів Р. Графіки зображені на рисунках 3.7 – 3.10. 43](#_Toc356181063)

[Висновки до розділу 3 46](#_Toc356181064)

[Список використаної літератури 47](#_Toc356181065)

[Додаток А 49](#_Toc356181066)

[Додаток Б 51](#_Toc356181067)

[Додаток В 53](#_Toc356181068)

[Додаток Г 59](#_Toc356181069)

[Додаток Д 67](#_Toc356181070)

[Розділ 2 68](#_Toc356181071)

[Розділ3 69](#_Toc356181072)

# Розділ 1 . Огляд механізму критичні секції

## 1. 1 Загальні теоретичні відомості

Паралельне і незалежне виконання процесів здійснюється до тих пір, поки не виникає завдання їх взаємодії, тобто процеси стають такими, що взаємодіють[1]. Існують різні форми взаємодії процесів, визначувані типами зв'язків між ними, - функціональними, просторово-часовими, такими, що управляють, інформаційними та ін.

Процес може породжувати інші процеси (відношення головний – дочірній), обмінюватися даними з іншими процесами, конкурувати по доступу до загального ресурсу, синхронізовуватися по певних подіях в інших процесах.

Існує два завдання синхронізації в паралельных обчисленнях - задача взаємного виключення і завдання синхронізації процесів по подіях[2].

## 1. 2 Завдання взаємного виключення.

Необхідно погоджувати роботу двох і більше процесів, що використовують деякий загальний ресурс, так, щоб у будь-який момент часу цей ресурс міг використовуватися тільки одним процесом і ресурс ставав доступним іншому процесу тільки при звільненні ресурсу. При цьому в випадку запиту процесом зайнятого ресурсу, цей процес (процес, що запрошує ресурс) має бути припинений (блокований) до тих пір, поки процес, що працює в даний момент із загальним ресурсом (процес, що володіє ресурсом), не звільнить загальний ресурс[3]. Існує два підходи до рішення задачі взаємного виключення, що базуються на контролі процесу або на контролі загального ресурсу.

Перший підхід грунтується на контролі процесів. Для цього в процесі виділяється критична ділянка (КД) - частина процесу, де виконується звернення до загального ресурсу. Знаходження процесу в критичній ділянці означає, що він працює із загальним ресурсом. Отже, контролювати роботу процесу із загальним ресурсом можна через безпосередній контролю КД :

- процес запрошує загальний ресурс, коли він намагається увійти в КД

- процес працює із загальним ресурсом, коли він знаходиться в КД

- процес звільняє загальний ресурс, коли він виходить з КД.

За допомогою спеціальних конструкцій (примітивів) ВХІДКД (Вхід в Критичну Ділянку) і ВИХІДКД (Вихід з Критичої Ділянки) здійснюється контроль за входом і виходом процесу у свою критичну ділянку. При цьому конструкції ВХІДКД і ВИХІДКД оточують критичну ділянку виконуючи для процесів роль своєрідної "огорожі" навколо критичної ділянки, який захищає загальний ресурс від одночасного використання. [4]

Функції конструкції ВХІДКД:

* *перевірка знаходження іншого процесу у своїй КД;*
* *введення заборони на вхід в КД для усіх процесів у разі, якщо КД вільна і процес входить в нього;*
* *блокування процесу, якщо КД вже зайнята.*

Функції конструкції ВИХІДКД:

* *зняття заборони на вхід в КД для інших процесів і вихід з КД.*

Реальними реалізаціями механізмів, заснованих на першому підході являються *семафори, мютекси, критичні секції та ін*.

Другий підхід пов'язаний з контролем безпосередньо загального ресурсу, для чого вводяться спеціальні засоби доступу до загального ресурсу, наприклад, ресурс разом із засобами доступу до нього інкапсулюють в спеціальний програмний модуль. Тут огорожа зводиться вже безпосередньо навколо загального ресурсу. Подібна концепція лежить в основі механізму моніторів.

## 1.3 Завдання синхронізації процесів.

Необхідно забезпечити узгодження (синхронізацію) поведінки двох і більше процесів по деякій події, яка матиме місце в певному процесі, наприклад, введення даних або завершення обчислень. При цьому процес, очікуючий подію, досягнувши певної точки (точки синхронізації), має бути припиненим до тих пір, поки в іншому процесі не станеться задана подія і цей процес (сигналізуючий процес) не пошле сигнал процесу, що чекає. Вихід в точку синхронізації для випадку, коли подія вже сталася, не повинен призводити до припинення процесу.

Для вирішення завдань взаємного виключення і синхронізації процесів потрібне використання спеціальних програмних засобів, таких як семафори, мютекси, критичні секції монітори та ін. Подібні механізми ґрунтуються на звертанні до команд операційної системи, оскільки зв’язані з такими операціями як блокувати і розблоковувати процес, виконуваних тільки операційною системою. Одним з найпростіших варіантів реалізації завдання взаємного виключення і синхронізації є критична секція.

## 1. 4 Поняття критична секція

Важливим поняттям синхронізації процесів є поняття "критична секція" програми. Критична секція - це частина програми, в якій здійснюється доступ до даних, що розділяються. Щоб виключити ефект гонок по відношенню до деякого ресурсу, необхідно забезпечити, щоб у кожен момент в критичній секції, пов'язаної з цим ресурсом, знаходився максимум один процес. Цей прийом називають взаємним виключенням.

Найпростіший спосіб забезпечити взаємне виключення – дозволити процесу, що знаходиться в критичній секції, забороняти всі переривання. Однак цей спосіб непридатний, оскільки небезпечно довіряти керування системою користувальницькому процесу; він може надовго зайняти процесор, а при краху процесу в критичній області крах потерпить вся система, тому що переривання ніколи не будуть дозволені.

Іншим способом є використання блокуючих змінних. З кожним розділяються ресурсом зв'язується двійкова змінна, яка приймає значення 1, якщо ресурс вільний (тобто жоден процес не знаходиться в даний момент в критичній секції, пов'язаної з даним процесом), і значення 0, якщо ресурс зайнятий.

Якщо всі процеси написані з використанням вищеописаних угод, то взаємне виключення гарантується. Слід зауважити, що операція перевірки і установки блокуючої змінної повинна бути неподільною. Пояснимо це. Нехай в результаті перевірки змінної процес визначив, що ресурс вільний, але відразу після цього, не встигнувши встановити змінну в 0, був перерваний. За час його призупинення інший процес зайняв ресурс, увійшов у свою критичну секцію, але також був перерваний, не завершивши роботи з ресурсом. Коли управління було повернуто першого процесу, він, вважаючи ресурс вільним, встановив ознака зайнятості і почав виконувати свою критичну секцію. Таким чином було порушено принцип взаємного виключення, що потенційно може призвести до небажаних наслідків. Щоб уникнути таких ситуацій в системі команд машини бажано мати єдину команду "перевірка-установка", або ж реалізовувати системними засобами відповідні програмні примітиви, які б забороняли переривання протягом всієї операції перевірки і установки.

Розглянемо механізм критичних секцій реалізованих різних мовах програмування.[5]

## 1. 5 Win32

В бібліотеці Win32 існує реалізація механізму критична секція. Вона виконана у вигляді двох функцій EnterCriticalSection() LeaveCriticalSection(). Дана реалізація не належить одному із операторів типу region, що робить її схожою на семафори і мютекси і не вирішує повністю проблему контроля примітивів для входу і виходу з критичної ділянки.

Робота з критичною секцією в Win32 зв’язана з виконанням наступних дій:

* Ініціалізація змінної типа Critical
* Ініціалізація критичної секції за допомогою функції InitializeCriticalSection();
* Вхід в критичну секцію через функцію EntercriticalSection()
* Вихід з критичної секції через функцію LeaveCriticalSection().
* Знищення критичної секції за допомогою функції DeleteCriticalSection.

**Приклад 1**

**Procedure** Prymer **is**

Crit\_sec: Critical Section;

**task** A **is**

……..

**EnterCriticalsection**(Crit\_sec);

-- Критична секція

**LeaveCriticalSection**(Crit\_sec);

**end** A;

**task** B **is**

……..

**EnterCriticalsection**(Crit\_sec);

-- Критична секція

**LeaveCriticalSection**(Crit\_sec);

**end** B;

**begin**

null;

**end** Start;

**begin**

**InitializeCriticalSection**(Crit\_sec);

Start;

**end** Prymer.

[4]

## 1. 6 C++

Реалізація критичних секцій в С++ реалізується завдяки класу CCriticalSection [6] та його методів і конструктора. Функціональні можливості класу CCriticalSection представлена об’єктом CRITICAL\_SECTION у Win32.

Методи CCriticalSection: lock() викликається з метою отримання доступу до критичної секції. Якщо інший потік вже знаходиться в критичній ділянці коду то він блоку потік доти поки не звільниться критична ділянка. Фактично реалізує операцію ВХІДКД

Unlock() звільняє критичну секцію для використання іншим потоком. Реалізує операцію ВИХІДКД. Ініціалізація критичної секції реалізується завдяки конструктору CCriticalSection.

## 1. 7 Delphi

Найбільш простий в розумінні способом синхронізації в Delphi є критичні секції. Механізм реалізації критичної секції являється таким же як і Win32, C++. Критична секція реалізується класом TCriticalSection в бібліотеці VCL. Розглянемо методи цього класу.

**procedure Acquire; override -** прив’язує критичну секцію до потоку, який викликає;

**constructor Create -**  створює об’єкт критичної секції;

**destructor Destroy; override -** знищує об’єкт критичної секції;

**procedure Enter -** реалізує операцію ВХІДКД, блокує інші потоки, коли потік, що визиває, заходить в критичну секцію;

**procedure Leave -** реалізує операцію ВИХІДКД, дозволяє іншим потокам використовувати критичну секцію;

**procedure Release; override -** звільняє критичну секцію.[7]

Тут потібно уточнити, що методи Enter і Leave включають себе виклики методів Acquire і Release відповідно. Тому можна сказати , що ці відповідні пари методів (Enter і Acquire, Leave і Release) виконують одну і ту дію.

**Приклад 2**

**procedure** TFooThread.Execute;

**var**

i: **Integer**;

**begin**

OnTerminate := Form1.DispGlblArr;

**EnterCriticalSection**(CS);

**for** i := 1 to MaxSize **do**

**begin**

GlobalArray[i] := GetNextNumber;

**Sleep**(5);

**end**;

**LeaveCriticalSection**(CS);

**end.**[8]

## 

## 1. 8 Java

Синхронізовані блоки в мові Java визначають у програмі фрагмент коду потоку, вхід в який здійснюється в режимі взаємного виключення. Це означає, що при спробі входу в синхронізований блок виконується перевірка, чи не зайнятий синхронізований блок іншим потоком. Якщо зайнятий, то потік, що викликав, блокується до тих пір, поки другий потік не покине свій синхронізований блок. Для ініціалізації синхронізованих блоків вони повинні зв'язуватися за допомогою спільних об'єктів: [4]

**synchronized** (sync\_object) {

/ / Операції з спільним ресурсами

} ,

де Object - це посилання на об’єкт, доступ до якого треба синхронізувати. Синхронізований блок гарантує, що виклик з потоків методу, що є членом

об’єкта Object, буде виконуватися в режимі взаємного виключення.

Існують наступні обмеження стосовно synchronized блоків у мові програмування Java:

1. Не iснує способу вийти з черги на отримання об’єкту критичної секцiї. Тобто спроба захопити об’єкт синхронiзацiї, який вже був захоплений iншим потоком, призведе до неминучого блокування процесу i помiщення його в чергу до тих пiр, поки йому не нададуть право ввiйти в критичну дiлянку.
2. Не iснує можливостi змiнити поведiнку блокування, тобто семантику оператора synchronized.
3. Немає нiякого контролю доступу до синхронiзацiї. Будь-який метод може використати оператор synchronized(obj) для будь-якого досту пного йому об’єкту. Це, звичайно, може призвести до зайвих затримок, зменшення швидкостi роботи програми у звязку з захопленням необхiдного iншим потокам об’єкту.
4. Наявнiсть оператора та блочного видiлення критичної дiлянки призводить до чiтких обмежень використання оператора synchronized в межах одного блоку. Наприклад, неможливо захопити об’єкт синхронiзацiї в одному методi, а зняти блокування (вiдпустити об’єкт) в iншому.

Всi цi обмеження можливо подолати, використовуючи допомiжнi класи, якi б контролювали та стежили за блокуванням та синхронiзацiєю. Наступною особливiстю синхронiзованих блокiв в мовi Java є те, що об’ єкт синхронiзацiї звiльнюється у будь-якому випадку, навiть якщо при виконаннi критичної дiлянки виникла виключна ситуацiя, яку не було оброблено. Таку поведiнку було вибрано з мiркувань зменшення кiлькостi ситуацiй, в яких можлива поява тупикувих ситуацiй та, як наслiдок, збiльшення надійностi програм. Однiєю з найчастiших причин застосування synchronized блокiв, є необхiднiсть створення синхронiзованих методiв, не маючи при цьому самої реалiзацiї цих методiв. У такому випадку створюють новий, синхронізований варiант такої функції, в тiлi якого будь-який необхiдний виклик базового методу розмiщується в синхронiзованному блоцi. Хоча synchronized блоки в мовi програмування Java э бiльш гнучкими, нiж synchronized методи, останнi виконуються значно швидше, беручи в увагу особливостi реалiзацiї цих механiзмiв в Вiртуальнiй Машинi Java (JVM).

Synchronized методи транслюються в значно меншу кiлькiсть операторiв байт коду, нiж виклик оператора синхронiзованного блоку.

**Приклад 3**

**Class** Work{

**double** Schet(**double** e){

}

}

**Class** Access **extends** Thread{

**Work** block1;

**Access**( block o){

Block1 = o;

}

**Public** **void** **run**(){

**double** x=5;

**double** y;

**Synchronized**(block1){

y = block1. Schet(x);

}

}

}

**Class** Main{

**public** **static** **void** **main**(**String** [] args){

Work wk = new Work();

Access ac1 = new Access(wk);

Access ac2 = new Access(wk);

Ac1.start;

Ac2.start;

}

}[4]

## 1. 9 OpenMp

**OpenMP [9]** (Open Multi-Processing) — це набір директив компілятора, бібліотечних процедур та змінних середовища, які призначені для програмування багатопоточних програм на багатопроцесорних системах із спільною пам'яттю на мовах C, C++ та Fortran.

Ключовим елементом для вирішення завдання взаємного виключення є механізм критичної секції. Принцип її роботи у всіх доступних для OpenMp мов є однаковою, але синтаксис буде різним.

Розглянемо реалізацію бібліотеки OpenMp в С++.

## 1. 10 OpenMp в С++

Механізм критичної секції в OpenMp в С++ ідентичний механізму в Win32 тому що обоє використовують Win32 API . Функції для реалізації методів ВХІДКД і ВИХІДКД такі як і в Win32 EnterCriticalSection() та LeaveCriticalSection(). Критична секція ініціалізується за допомогою директиви #pragma omp critical де ми вказуємо ім’я нашої критичної секції. У паралельних регіонах часто зустрічаються блоки коду, доступ до яких бажано надавати тільки одному потоку, - наприклад, блоки коду, що відповідають за запис даних у файл. У багатьох таких ситуаціях не має значення, який потік виконає код, важливо лише, щоб цей потік був єдиним. Для цього в OpenMP служить директива # pragma omp single.[10]

#**pragma** **omp** critical

{

memmove(C2, C, sizeof(C2));

memmove(MO2, MO, sizeof(MO2));

}

[11]

## 1. 11 OpenMР в Fortran.

У програмах, написаних на мові Fortran, завдання синхронізації типу critical здійснюється за допомогою наступного пропозиції OpenMP:

c $ omp critical [name]

<Структурний блок програми>

c $ omp end critical [name]

Тут name - ім'я критичної секції (critical section). Різні критичні секції незалежні, якщо вони мають різні імена. Не зазначені критичні секції відносяться до однієї і тієї ж секції.

**Приклад 4**

integer :: cnt1, cnt2

с$omp parallel private (i)

с$omp& shared (cnt1, cnt2)

с$omp do

do i = 1, n

... do work ...

if (condition1) then

с$omp critical (name1)

cnt1 = cnt1 + 1

с$omp end critical (name1)

else

с$omp critical (name1)

cnt1 = cnt1 - 1

с$omp end critical (name1)

endif

if (condition2) then

с$omp critical (name2)

cnt2 = cnt2+1

с$omp end critical (name2)

endif

enddo

с$omp end parallel

приклад використання механізму критичної секції в Fortran. [12]

## 1. 12 C#

Механізм який вирішує завдання доступу до спільного ресурсу є влаштованим в мову С#, тому доступ до всіх об’єктів може бути синхронізованим. Синхронізація підтримується ключовим словом lock. Ключове слово lock викликає Enter на початку блоку і Exit в кінці блоку. Оскільки механізм закладений вже в мову, тому використовувати його дуже легко. Формат використання інструкції lock такий. [13]

lock(object) {

// Операції з спільним ресурсами, синхронізація.

} Якщо потрібно синхронізувати тільки один елемент, фігурні дужки можна опустити.

Інструкція lock гарантує, що вказаний блок коду, захищений блокуванням для даного об'єкта, може бути використаний тільки потоком, який блокує. Всі інші потоки залишаються заблокованими до тих пір, поки блокування не буде знята. А знята вона буде лише при виході з цього блоку.

**Приклад 5**

**class** Account {

**long** val = 0;

**object** thisLock = new **object**();

**public** **void** Deposit(**const** **long** x) {

**lock** (thisLock) {

val += x;

}

}

**public** **void** Withdraw(**const** **long** x) {

**lock** (thisLock) {

val -= x;

}

}

}

[14]

У наведеному вище прикладі область блокування обмежена цією функцією, оскільки не існує посилань на об'єкт lockThis поза функції. Якби таке посилання існувала, область блокування включала б цей об'єкт. Строго кажучи, об'єкт, що надається для lock, використовується тільки для того, щоб унікальним чином визначити ресурс, до якого надається доступ для різних потоків, тому це може бути довільний екземпляр класу.

## 1. 13 Python

Аналогом критичної секції як і в С# є критична секція в Python. Блокування в мові Python найбільш фундаментальний механізм синхронізації, який надається модулем threading. У кожен момент часу блокування може належати не більш ніж одному потоку. Якщо потік намагається отримати блокування, вже приналежну іншому потоку, виконання першого потоку припиняється до тих пір, поки блокування не буде звільнена. Зазвичай блокування використовуються для доступу до ресурсів. Для кожного ресурсу, що розділяється створюйте об'єкт типу Lock. Коли потрібен доступ до ресурсу, слід викликати метод acquire щоб захопити блокування (виклик у разі потреби почекає звільнення блокування), і викликати метод release щоб звільнити блокування. [15]

lock = Lock ()

lock.acquire () # заблокує виконання, якщо блокування кимось захоплена

... доступ до ресурсу

lock.release ()

Для коректного виконання важливо звільняти блокування навіть у випадку виникнення помилок. Для цього можна використовувати try-finally

lock.acquire ()

try:

    ... доступ до ресурсу

finally:

    lock.release () # звільнити блокування, що б не відбулося

В Python 2.5 та пізніших версіях можна використовувати оператор with. Якщо оператор with працює з блокуванням, то він автоматично захоплює блокування перед входом в блок, і звільняє її після виходу з блоку:

from \_\_ future\_\_ import with\_statement # тільки у версії 2.5

with lock:

    ... доступ до ресурсу

Метод acquire приймає необов'язковий прапорець очікування, який можна використовувати для уникнення блокування, якщо блокування вже кимось захоплена. Якщо встановити прапорець в False, метод не блокується, але поверне False, якщо блокування кимось захоплена:[16]

**Приклад 6**

class FetchUrls(threading.Thread):

def \_\_init\_\_(self, urls, output, lock):

...

self.lock = lock

def run(self):

...

while self.urls:

...

self.lock.acquire()

print 'lock acquired by %s' % self.name

self.output.write(d.read())

print 'write done by %s' % self.name

print 'lock released by %s' % self.name

self.lock.release()

...

def main():

...

lock = threading.Lock()

...

t1 = FetchUrls(urls1, f, lock)

t2 = FetchUrls(urls2, f, lock)

...[17]

# Висновки до розділу 1

Існує два завдання синхронізації в паралельних обчисленнях - задача взаємного виключення і завдання синхронізації процесів по подіях. Основним завданням завдання взаємного виключення за допомогою критичних секцій є реалізація примітивів ВХІДКД і ВИХІДКД і слідкувати за входом і виходом у критичну ділянку, в той час як завдання задачі синхронізації є узгодження (синхронізацію) поведінки двох і більше процесів по деякій події, яка матиме місце в певному процесі.

Проаналізувавши механізм роботи критичних секцій в різних мовах і бібліотеках паралельного програмування, можна зробити висновок що сама організація роботи механізму дуже схожа. В більшості випадках відрізняються лише синтаксисом функцій для реалізації примітивів ВХІДКД і ВИХІДКД а також ініціалізації. Даний механізм являється одним із найпростіших у використанні. Аналiз реалiзацiї критичної секцiї в Java показав, що synchronized блоки є достатньо гнучким iнструментом для вирiшення задач взаємного виключення в паралельних програмах. Водночас, було показано, що даний механiзм в Java є досить повiльним, порiвняно з iншими.

Що до інших мов з використанням механізму критичної секції, то можна зробити висновок що це один з найшвидших механізмів реалізації задачі взаємного виключення.

# 

# РОЗДІЛ 2. Розробка програми ПРГ1 для ПКС СП

Відповідно до технічного завдання необхідно розробити програмне забезпечення для розв'язання задачі в паралельній комп'ютерній системі зі спільною пам'яттю. Програмне забезпечення необхідно реалізувати на мові Java, використовуючи монітори. Комп'ютерна система зі спільною пам'яттю містить 6 процесорів (P) і три пристрої введення-виведення (ПВВ). Введення матриці MO виконується в процесорі 4, матриці MХ у процесорі 6, а векторів B і Z – у процесорі 1. Константа α вводиться в процесорі 4

Виведення результату (вектор А) виконується в процесорі 1.

Схема комп'ютерної системи зі спільною пам’яттю представлена в технічному завданні на рисунку 1

## 2.1 Аналіз задачі на внутрішній паралелізм з використанням концепції необмеженого паралелізму.

Основні принципи концепції необмеженого паралелізму:

1. Кількість процесорів – необмежена ();
2. Час виконання операції однаковий і дорівнює 1.
3. Не враховується час передачі даних.

Виходячи із даної концепції побудуємо ярусно-паралельну форму алгоритму обчислення заданого математичного виразу.

Замінивши на еквіваленті вирази матричні операції, перепишемо задане завдання A= (B \* C) \* Z + R \* (MO \* MT) наступним чином:

ki = (b1,1\* c1,1,b1,2\* c1,2,…,bN, N-1\* cN, N-1, bN,N\* cN,N)

ai = (k1,1\* z1,1,k,2\* z1,2,…,kN, N-1\* zN, N-, kN,N\* zN,N) + (r1\*(o1,1\*t1,i+ … + o1,N\*tN,i)+… rN\* (oN,1\*t1,i+ … + oN,N\*tN,i))

де i=1..N

З виразу видно, що елементи вектора ai шукаються в два етапи.

Ярусно-паралельна форма ai зображена на рис. 2.1.

ЯПФ знаходження зображена на рис. 2.1.

Знайдемо параметри цієї ЯПФ:

1. Висота ЯПФ:

.

1. Ширина ЯПФ:
2. Загальна кількість вузлів (операцій):

.

Оскільки для знаходження результату необхідно виконати розглянутий вище алгоритм N раз, то як кінцевий результат отримуємо:

1. Необхідна кількість процесорів .
2. Час обчислення завдання на процесорах .
3. Час обчислення завдання на 1 процесорі

.

1. Коефіцієнт прискорення .
2. Коефіцієнт ефективності .



Рис. 2.1. Ярусно-паралельна форма алгоритму обчислення .

## 2.2 Розробка паралельного математичного алгоритму

Розрахунок даного матричного рівняння можливо провести в один етап:

AH = B\*(MO \* MXH) + *a*\*ZH

де H = N / P, N – розмірність матриць, P – кількість процесорів.

Спільні ресурси: B, MO, *a*.

## 2.3 Розробка алгоритмів процесів

Цей етап включає розроблення алгоритму роботи кожного процесу. Алгоритм має включати всі дії процесу, включаючи розв’язання завдань взаємного виключення та синхронізації.

Задачі взаємного виключення пов’язані з копіюванням спільних змінних , а задачі синхронізації – із синхронізацією введення даних та завершенням розрахунків і виведенням результату.

Таблиця 2.1. Алгоритми потоків для системи зі спільною пам’яттю

|  |  |  |
| --- | --- | --- |
| *№* | *Алгоритм процесів №1* | *ТС/КУ* |
| 1. | Ждати введення даних | W2-1, W4-1, W6-1 |
| 2. | Копіювання *a* 1 := *a*, B1 := B, MO1 := MO | КУ |
| 3. | AH = B1\*(MO1 \* MXH) + *a1*\*ZH |  |
| 4. | Сигнал про закінчення обрахування AH | S1-2 |

|  |  |  |
| --- | --- | --- |
| *№* | *Алгоритм процесу №2* | *ТС/КУ* |
| 1. | Введення B,Z |  |
| 2. | Сигнал всім задачам про введення | S1-1, S3-1, S4-1, S5-1, S6-1 |
| 3. | Очікування вводу в T4 і T6 | W4-1,W6-1 |
| 4. | Копіювання *a* 2 := *a*, B2 := B, MO2 := MO | КУ |
| 5. | AH = B2\*(MO2 \* MXH) + *a2*\*ZH |  |
|  | Очікування обрахування AH | W1-2, W3-2, W4-2, W5-2, W6-2 |
| 6. | Вивід А |  |

|  |  |  |
| --- | --- | --- |
| *№* | *Алгоритм процесів №3* | *ТС/КУ* |
| 1. | Ждати введення даних | W2-1, W4-1, W6-1 |
| 2. | Копіювання *a* 3 := *a*, B3 := B, MO3 := MO | КУ |
| 3. | AH = B3\*(MO3 \* MXH) + *a3*\*ZH |  |
| 4. | Сигнал про закінчення обрахування AH | S1-2 |

|  |  |  |
| --- | --- | --- |
| *№* | *Алгоритм процесу №4* | *ТС/КУ* |
| 1. | Введення *a*, MO |  |
| 2. | Сигнал всім задачам про введення | S1-1, S2-1, S3-1, S4-1, S5-1 |
| 3. | Очікування вводу в T2 і T6 | W2-1,W6-1 |
| 4. | Копіювання a4 := a, B4 := B, MO4 := MO | КУ |
| 5. | AH = B4\*(MO4 \* MXH) + *a*\*ZH |  |
| 6. | Сигнал про закінчення обрахування AH | S2-2 |

|  |  |  |
| --- | --- | --- |
| *№* | *Алгоритм процесів №5* | *ТС/КУ* |
| 1. | Ждати введення даних | W2-1, W4-1, W6-1 |
| 2. | Копіювання *a* 5 := *a*, B5 := B, MO5 := MO | КУ |
| 3. | AH = B5\*(MO5 \* MXH) + *a5*\*ZH |  |
| 4. | Сигнал про закінчення обрахування AH | S1-2 |

|  |  |  |
| --- | --- | --- |
| *№* | *Алгоритм процесу №6* | *ТС/КУ* |
| 1. | Введення *MX* |  |
| 2. | Сигнал всім задачам про введення | S1-1, S2-1, S3-1, S4-1, S5-1 |
| 3. | Очікування вводу в T2 і T4 | W2-1,W4-1 |
| 4. | Копіювання a5 := a, B6 := B, MO6 := MO | КУ |
| 5. | AH = B6\*(MO6 \* MXH) + *a*\*ZH |  |
| 6. | Сигнал про закінчення обрахування AH | S2-2 |

## 2.4 Розробка схеми взаємодії процесів

При розробці схеми взаємодії процесів, повинно враховуватися особливості самої структури програми. Оскільки враховувати особливості бібліотеки OpenMP то можна поділити схему на декілька етапів: введення даних процесорами, синхронізація по вводу реалізована за допомогою бар’єру, копіювання спільних ресурсів за допомогою механізму критичної секції, обчислення формули, синхронізація по збору результатів і вивід результуючого вектора, реалізовано за допомогою бар’єра. Схема взаємодії процесорів наведена в додатку А .

## 2.5 Розробка програми ПРГ1

Для вирішення проблеми синхронізації при вводі даних використовується засіб паралельного програмування бібліотеки OpenMP – бар’єри (#pragma omp barier). Для копіювання спільного ресурсу використовується директива #pragma omp critical. Тільки один потік може мати доступ до об’єкту. Синхронізація по виводу реалізована також за допомогою бар’єрів. Лістинг програми наведений в додатку В

## 

## 2.6 Тестування програми ПРГ1

Тестування програми ПРГ1 здійснювалося на комп’ютерній системі з такими технічними характеристиками:

* Процесор AMD Phenom (tm) I I X6 1055T з внутрішньою тактовою частотою 2,80 ГГц;
* 4 Гб оперативної пам’яті;
* Операційна система – Windows 7, 32-розрядна.

Охарактеризуємо методику тестування: в ПКС послідовно використовуються один, два, три, чотири, п’ять та шість процесорів – ядра (шляхом відключення ядер в Task Manager), для яких визначається час виконання програми ПРГ1 (значення Т1, Т2, Т3, Т4, Т5, Т6). При цьому встановлюється декілька значень розмірності матриць N = 900, 1300, 1900.Для дослідження ефективності ПЗ задачі були визначені значення коефіцієнтів прискорення та ефективності.

Для замірів часу використовувались функції зі стандартної бібліотеки мови - **ctime**

Таблиця 2.1. Час виконання програм обчислення на системі зі СП

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N / P | 1 | 2 | 3 | 4 | 5 | 6 |
| 900 | 7,172 | 3,938 | 3,109 | 2,625 | 2,36 | 2,032 |
| 1300 | 30,574 | 16,531 | 11,359 | 9,641 | 8,954 | 6,563 |
| 1900 | 102,906 | 51,969 | 35,641 | 30,375 | 28,469 | 22,609 |

Коефіцієнт прискорення обраховується за формулою [4]:



Таблиця 2.2. Значення коефіцієнтів прискорення для системи з СП

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N / P | 2 | 3 | 4 | 5 | 6 |
| 900 | 1,821229 | 2,306851 | 2,73219 | 3,038983 | 3,529528 |
| 1300 | 1,849495 | 2,69161 | 3,171248 | 3,414563 | 4,65854 |
| 1900 | 1,980142 | 2,887293 | 3,387852 | 3,614669 | 4,55155 |

Коефіцієнт ефективності обраховується за формулою [4]:



Таблиця 2.3. Значення коефіцієнтів ефективності для системи з СП

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N / P | 2 | 3 | 4 | 5 | 6 |
| 900 | 91,06145 | 76,89504 | 68,30476 | 60,77966 | 58,82546 |
| 1300 | 92,47474 | 89,72034 | 79,28119 | 68,29127 | 77,64234 |
| 1900 | 99,0071 | 96,24309 | 84,6963 | 72,29337 | 75,85917 |

Оцінимо час виконання обчислень програмою ПРГ1 залежно від розмірності матриць N та кількості процесорів Р за допомогою графіків, побудованих на основі таблиці 2.1. Графіки зображені на рисунку 2.2.

Рис. 2.2. Графік зміни часу виконання програми для різних N в залежності від кількості ядер.

Проаналізуємо зміну значень коефіцієнту прискорення КП залежно від розмірності матриць N та кількості процесорів Р за допомогою графіків, побудованих на основі таблиці 2.2. Графіки зображені на рисунках 2.3 – 2.6.

Рис. 2.3. Графік зміни КП для різних N в залежності від кількості ядер.

Рис. 2.4. Графік зміни КП в залежності від кількості ядер. N = 900

Рис. 2.5. Графік зміни КП в залежності від кількості ядер. N = 1300

Рис. 2.6. Графік зміни КП в залежності від кількості ядер. N =1900

Аналогічно на основі таблиці 2.3 проаналізуємо зміну значень коефіцієнту ефективності Ке залежно від розмірності матриць N та кількості процесорів Р. Графіки зображені на рисунках 2.7 – 2.10.

Рис. 2.7. Графік зміни коефіцієнту ефективності Ке для різних N в залежності від кількості ядер.

Рис. 2.8. Графік зміни Ке в залежності від кількості ядер. N = 900

Рис. 2.9. Графік зміни Ке в залежності від кількості ядер. N = 1300

Рис. 2.10. Графік зміни Ке в залежності від кількості ядер. N = 1900

## Висновки до розділу 2

У цьому розділі курсової роботи було оцінено задану векторно-матричну операцію на паралелізм з використанням концепції необмеженого паралелізму та теореми Мунро-Петерсона, розроблено паралельний математичний алгоритм розв’язання задачі, алгоритм основної програми та алгоритми процесів, схему взаємодії процесів та виконано розробку програми ПРГ1 для ПКС СП з використанням бібліотеки OpenMP.

Тестування програми показало наступне:

* Використання багатоядерної ПКС та програми ПРГ1 забезпечує скорочення часу обчислення заданої математичної задачі A = B\*(MO\*MX)+α\*Z.
* Максимальний час роботи програми становить 102,906с.

Значення коефіцієнтів прискорення КП лежать в межах від 1,821229 до 4,65854

в залежності від розмірності матриць N та кількості ядер Р, що використовуються.

* Максимальне значення КП забезпечує ПКС з Р = 6 та N = 1300.
* Мінімальне значення КП забезпечує ПКС з Р = 2 та N = 900.
* Значення коефіцієнтів ефективності Ке лежать в межах від 58,8% до 99% в залежності від розмірності матриць N та кількості ядер Р, що використовуються.
* Максимальне значення Ке забезпечує ПКС з Р = 2 та N = 1300.
* Мінімальне значення Ке забезпечує ПКС з Р = 6 та N = 900.

Отже, розроблена програма ПРГ1 ефективніше працює для більших значень розмірності матриць N, але це потребує більше часу і, відповідно, більше оперативної пам’яті.

# РОЗДІЛ 3. Розробка програми ПРГ2 для ПКС ЛП

Відповідно до технічного завдання необхідно розробити програмне забезпечення для розв'язання задачі в паралельній комп'ютерній системі зі спільною пам'яттю. Програмне забезпечення необхідно реалізувати на мові Java, використовуючи монітори. Комп'ютерна система зі локальною пам'яттю містить 6 процесорів (P) і три пристрої введення-виведення (ПВВ). Введення матриці MO виконується в процесорі 3, матриці MХ у процесорі 6, а векторів B і Z – у процесорі 1. Константа α вводиться в процесорі 3

Виведення результату (вектор А) виконується в процесорі 1.

Схема комп'ютерної системи зі локальною пам’яттю представлена в технічному завданні на рисунку 2.

## 3.1 Розробка паралельного математичного алгоритму

Розрахунок даного матричного рівняння можливо провести в один етап:

AH = B\*(MO \* MXH) + *a*\*ZH

де H = N / P, N – розмірність матриць, P – кількість процесорів.

## 3.2 Розробка алгоритмів процесів

Цей етап включає розроблення алгоритму роботи кожного процесу. Алгоритм має включати всі дії процесу, включаючи розв’язання завдань взаємного виключення та синхронізації.

Таблиця 3.1. Алгоритми потоків для системи зі локальною пам’яттю

|  |  |
| --- | --- |
| *№* | *Алгоритм процесів №1* |
| 1. | Введення B,Z |
| 2. | Передати B,Z5H  задачі Т2 |
| 3. | Прийняти від Т2 МО, *a, МХ* H |
| 4. | AH = B\*(MO \* MXH) + *a*\*ZH |
| 5. | Прийняти A5H від задачі Т2 |
| 6. | Вивід А |

|  |  |
| --- | --- |
| *№* | *Алгоритм процесу №2* |
| 1. | Прийняти від Т3 МО, *a, МХ* 2H |
| 2. | Передати Т1 МО, *a, МХ* H |
| 3. | Прийняти від Т1 B,Z5H |
| 4. | Передати Т3 B,Z4H |
| 5. | AH = B\*(MO \* MXH) + *a*\*ZH |
| 6. | Прийняти A4H від задачі Т3 |
| 7. | Передати A5H задачі Т1 |

|  |  |
| --- | --- |
| *№* | *Алгоритм процесів №3* |
| 1. | Введення МО , *a* |
| 2. | Прийняти від Т4 *МХ* 3H |
| 3. | Передати Т2 МО, *a, МХ* 2H |
| 4. | Прийняти від Т2 B,Z4H |
| 5. | Передати Т4 МО, *a, В*, Z3H |
| 6. | AH = B\*(MO \* MXH) + *a*\*ZH |
| 7. | Прийняти A3H від задачі Т4 |
| 8. | Передати A5H задачі Т2 |

|  |  |
| --- | --- |
| *№* | *Алгоритм процесу №4* |
| 1. | Прийняти від Т5 *МХ* 4H |
| 2. | Передати Т3 *МХ* 3H |
| 3. | Прийняти від Т3 МО, *a, В* , Z3H |
| 4. | Передати Т5 МО, *a, В* , Z2H |
| 5. | AH = B\*(MO \* MXH) + *a*\*ZH |
| 6. | Прийняти A2H від задачі Т5 |
| 7. | Передати A3H задачі Т3 |

|  |  |
| --- | --- |
| *№* | *Алгоритм процесів №5* |
| 1. | Прийняти від Т6 *МХ* 5H |
| 2. | Передати Т4 *МХ* 4H |
| 3. | Прийняти від Т4 МО, *a, В* , Z2H |
| 4. | Передати Т6 МО, *a, В* , ZH |
| 5 | AH = B\*(MO \* MXH) + *a*\*ZH |
| 6 | Прийняти AH від задачі Т6 |
| 7 | Передати A2H задачі Т4 |

|  |  |
| --- | --- |
| *№* | *Алгоритм процесу №6* |
| 1. | Введення *MX* |
| 2. | Передати Т5 *МХ* 5H |
| 3. | Прийняти від Т5 МО, *a, В* , ZH |
| 4. | AH = B\*(MO \* MXH) + *a*\*ZH |
| 5. | Передати AH задачі Т5 |

## 

## 3.3 Розробка схеми взаємодії процесів

Для розробки схеми взаємодії процесів у Ада Рандеву потрібно прив’язати передачу даних з однієї задачу в іншу з входом описаним для кожної з задач. Також потрібно чітко вказати елементи які передаються, або їх частини.

## 3.4 Розробка програми ПРГ2

Для вирішення проблеми синхронізації та взаємного виключення потрібно врахувати особливості механізму Ада Рандеву. В програмі використовуються лише входи для передачі даних з однієї задачі в іншу. В кожній задачі описані декілька входів необхідних для отримання інформації. Ці ж входи будуть виконувати роль задачі синхронізації процесів, оскільки поки не спрацює оператор прийому accept задача блокується поки не виконає прийом даних. Лістинг програми наведений в додатку Г

## 3.5 Тестування програми ПРГ2

Тестування проводилося на системі з характеристиками:

Центральний процесор:

Оперативна пам’ять:

Операційна система:

Версії програмного забезпечення:

Для замірів часу використовувались функції зі стандартної бібліотеки мови -  **Ada.Calendar**

Таблиця 3.1. Час виконання програм обчислення на системі зі СП

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N / P | 1 | 2 | 3 | 4 | 5 | 6 |
| 600 | 17,3171 | 8,8978 | 5,8742 | 4,7293 | 4,2537 | 3,974 |
| 1200 | 141,3447 | 70,677 | 49,1009 | 38,7431 | 35,674 | 28,1273 |
| 1800 | 453,9751 | 230,6442 | 157,5652 | 126,5164 | 108,2614 | 91,2791 |

Коефіцієнт прискорення обраховується за формулою [4]:



Таблиця 3.2. Значення коефіцієнтів прискорення для системи з СП

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N / P | 2 | 3 | 4 | 5 | 6 |
| 900 | 1,946223 | 2,947993 | 3,661662 | 4,071068 | 4,357599 |
| 1300 | 1,999868 | 2,878658 | 3,648255 | 3,962121 | 5,025178 |
| 1900 | 1,968292 | 2,881189 | 3,588271 | 4,193324 | 4,973484 |

Коефіцієнт ефективності обраховується за формулою [4]:



Таблиця 3.3. Значення коефіцієнтів ефективності для системи з ЛП

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N / P | 2 | 3 | 4 | 5 | 6 |
| 900 | 97,31113 | 98,26643 | 91,54156 | 81,42135 | 72,62666 |
| 1300 | 99,99342 | 95,95527 | 91,20637 | 79,24242 | 83,75297 |
| 1900 | 98,41459 | 96,03963 | 89,70677 | 83,86648 | 82,89139 |

Оцінимо час виконання обчислень програмою ПРГ2 залежно від розмірності матриць N та кількості процесорів Р за допомогою графіків, побудованих на основі таблиці 3.1. Графіки зображені на рисунку 3.2.

Рис. 3.2. Графік зміни часу виконання програми для різних N в залежності від кількості ядер.

Проаналізуємо зміну значень коефіцієнту прискорення КП залежно від розмірності матриць N та кількості процесорів Р за допомогою графіків, побудованих на основі таблиці 3.2. Графіки зображені на рисунках 3.3 –3.6.

Рис. 3.3. Графік зміни КП для різних N в залежності від кількості ядер.

Рис. 3.4. Графік зміни КП в залежності від кількості ядер. N = 900

Рис. 3.5. Графік зміни КП в залежності від кількості ядер. N = 1300

## Аналогічно на основі таблиці 3.3 проаналізуємо зміну значень коефіцієнту ефективності Ке залежно від розмірності матриць N та кількості процесорів Р. Графіки зображені на рисунках 3.7 – 3.10.

Рис. 3.7. Графік зміни коефіцієнту ефективності Ке для різних N в залежності від кількості ядер.

Рис. 3.8. Графік зміни Ке в залежності від кількості ядер. N = 900

Рис. 3.9. Графік зміни Ке в залежності від кількості ядер. N = 1300

Рис. 3.10. Графік зміни Ке в залежності від кількості ядер. N = 1900

## Висновки до розділу 3

У цьому розділі курсової роботи було оцінено ПРГ2 для ПКС ЛП з використанням механізму Ада Рандеву.

Тестування програми показало наступне:

* Використання багатоядерної ПКС та програми ПРГ2 забезпечує скорочення часу обчислення заданої математичної задачі A = B\*(MO\*MX)+α\*Z.
* Максимальний час роботи програми становить 453,9751с.

Значення коефіцієнтів прискорення КП лежать в межах від 1,946223

до 4,973484 в залежності від розмірності матриць N та кількості ядер Р, що використовуються.

* Максимальне значення КП забезпечує ПКС з Р = 6 та N = 1300.
* Мінімальне значення КП забезпечує ПКС з Р = 2 та N = 900.

Значення коефіцієнтів ефективності Ке лежать в межах від 72,62666% до 99,99342% в залежності від розмірності матриць N та кількості ядер Р, що використовуються.

* Максимальне значення Ке забезпечує ПКС з Р = 2 та N = 1300.
* Мінімальне значення Ке забезпечує ПКС з Р = 6 та N = 900.

Отже, розроблена програма ПРГ2 ефективніше працює для менших значень розмірності матриць N, але це потребує більше часу і, відповідно, більше оперативної пам’яті.

# Список використаної літератури

1. Паралелізм [Електронний ресурс] – Режим доступу:

<http://uk.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%BB%D0%B5%D0%BB%D1%96%D0%B7%D0%BC_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)>

1. Синхронізація процесів [Електронний ресурс] – Режим доступу: <http://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%96%D0%B2>
2. Задача взаємного виключення [Електронний ресурс] – Режим доступу:

<http://alice.pnzgu.ru/~dvn/prolog/articls/8.htm>

1. Жуков І.А., Корочкін О.В. Паралельні та розподілені обчислення. Навч. Посіб. – К.: Корнейчук, 2005. – 226 с.
2. Критична секція [Електронний ресурс] – Режим доступу:<http://uk.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D1%82%D0%B8%D1%87%D0%BD%D0%B0_%D1%81%D0%B5%D0%BA%D1%86%D1%96%D1%8F>
3. Класс CCriticalSection [Електронний ресурс] – Режим доступу: <http://msdn.microsoft.com/ru-ru/subscriptions/h5zew56b>
4. Организация потоков в Delphi-приложениях. Синхронизация потоков [Електронний ресурс] – Режим доступу: <http://it.kgsu.ru/DelThread/thr0005.html>
5. Основы функционирования операционных систем. Практикум [Електронний ресурс] – Режим доступу: <http://chekalov.sumdu.edu.ua/os.09/topic_02/crit_sec.htm>
6. OpenMP [Електронний ресурс] – Режим доступу: <http://uk.wikipedia.org/wiki/OpenMP>
7. OpenMP и С++ [Електронний ресурс] – Режим доступу: <http://msdn.microsoft.com/ru-ru/library/dd335940.aspx>
8. Директивы OpenMP [Електронний ресурс] – Режим доступу:<http://edu.chpc.ru/parallel/mainse5.html>
9. Загрузка и синхронизация в OpenMP [Електронний ресурс] – Режим доступу: <http://www.intuit.ru/department/se/openmp/3/>
10. Критические секции [Електронний ресурс] – Режим доступу: <http://www.rsdn.ru/article/baseserv/critsec.xml>
11. Д. Указание Реализация- расширений функциональности в OpenMP C/C# [Електронний ресурс] – Режим доступу: <http://msdn.microsoft.com/ru-ru/library/cz27w838.aspx>
12. Что нового в Python 2.6 [Електронний ресурс] – Режим доступу: <http://ru.wikibooks.org/wiki/%D0%A7%D1%82%D0%BE_%D0%BD%D0%BE%D0%B2%D0%BE%D0%B3%D0%BE_%D0%B2_Python_2.6>
13. threading — Higher-level threading interface [Електронний ресурс] – Режим доступу: <http://docs.python.org/2/library/threading.html>
14. Python threads synchronization: Locks, RLocks, Semaphores, Conditions, Events and Queues [Електронний ресурс] – Режим доступу: <http://www.laurentluce.com/posts/python-threads-synchronization-locks-rlocks-semaphores-conditions-events-and-queues/>

# Додаток А



# Додаток Б



# Додаток В

1. // course work

2. // OpenMP

3. // Tkachenko Igor IO-01

4. // ---------------------------------------------------

5.

6. #include "stdafx.h"

7. #include <iostream>

8. #include <omp.h>

9.

10. #include <ctime>

11. #include <cstdio>

12.

13. using std::cout;

14. using std::endl;

15.

16. const int N = 900;

17. int P = 6;

18. int H = N / P;

19. int A[N], Z[N], B[N];

20. int MO[N][N], MX[N][N];

21. int alfa;

22. int X[N];

23. int MAA[N][N];

24. int Q[N];

25.

26.

27. // \* \* \* \* \* Ввод вектора

28. void VectorInput(int A[N], int filler) {

29. for (int i = 0; i < N; i++) {

30. A[i] = filler;

31. }

32. }

33. // \* \* \* \* \* Ввод матрицы

34. void MatrixInput(int MM[N][N], int filler) {

35. for (int i = 0; i < N; i++) {

36. for (int j = 0; j < N; j++) {

37. MM[i][j] = filler;

38. }

39. }

40. }

41. // \* \* \* \* \* Вывод вектора

42. void VectorOutput(int A[N]) {

43. for (int i = 0; i < N; i++) {

44. cout<<A[i]<<" ";

45. }

46. cout<<endl;

47. }

48. // \* \* \* \* \* Умножение матриц

49. void MatricesMul(int border1, int border2, int MA[N][N], int MB[N][N], int MAA[N][N]) {

50. for (int i = border1; i < border2; i++) {

51. for (int j = 0; j < N; j++) {

52. MAA[i][j] = 0;

53. for (int k = 0; k < N; k++) {

54. MAA[i][j] += MA[k][j] \* MB[i][k];

55. }

56. }

57. }

58. }

59. // \* \* \* \* \* Умножение вектора на матрицу

60. void MatrixOnVectorMul(int border1, int border2, int A[N], int MA[N][N], int X[N]) {

61. for (int i = border1; i < border2; i++) {

62. X[i] = 0;

63. for (int j = 0; j < N; j++) {

64. X[i] += A[j] \* MA[i][j];

65. }

66. }

67. }

68.

69. void VectorOnConstantMul(int border1, int border2, int A[N], int B, int X[N]){

70. for (int i = border1; i < border2; i++) {

71. X[i] = A[i] \* B;

72. }

73.

74. }

75. // \* \* \* \* \* Сложение векторов

76. void VectorsAdd(int border1, int border2, int A[N], int B[N], int X[N]) {

77. for (int i = border1; i < border2; i++) {

78. X[i] = A[i] + B[i];

79. }

80. }

81.

82.

83.

84. // \* \* \* \* \* \* \* \* \* \* TASK T1 \* \* \* \* \* \* \* \* \* \*

85. void T1\_B1() {

86. cout<<"Task 1 has started"<<endl;

87. }

88.

89. void T1\_B2() {

90. int B1[N];

91. int MO1[N][N];

92. int alfa1;

93. #pragma omp critical

94. {

95. memmove(B1, B, sizeof(B1));

96. memmove(MO1, MO, sizeof(MO1));

97. alfa1=alfa;

98. }

99.

100. MatricesMul(0, H, MO1, MX, MAA);

101. MatrixOnVectorMul(0, H, B1, MAA, X);

102. VectorOnConstantMul(0, H, Z, alfa1, Q);

103. VectorsAdd(0, H, Q, X, A);

104.

105. }

106.

107. void T1\_B3() {

108.

109.

110. if (N < 13) {

111. cout<<"Result: A = ";

112. VectorOutput(A);

113. }

114. cout<<"Task 1 has finished."<<endl;

115. }

116.

117.

118. // \* \* \* \* \* \* \* \* \* \* TASK T2 \* \* \* \* \* \* \* \* \* \*

119. void T2\_B1() {

120. cout<<"Task 2 has started"<<endl;

121. VectorInput(Z, 1);

122. VectorInput(B, 1);

123. }

124.

125. void T2\_B2() {

126. int B2[N];

127. int MO2[N][N];

128. int alfa2;

129. #pragma omp critical

130. {

131. memmove(B2, B, sizeof(B2));

132. memmove(MO2, MO, sizeof(MO2));

133. alfa2= alfa;

134. }

135.

136. MatricesMul(H, 2\*H, MO2, MX, MAA);

137. MatrixOnVectorMul(H, 2\*H, B2, MAA, X);

138. VectorOnConstantMul(H, 2\*H, Z, alfa2, Q);

139. VectorsAdd(H, 2\*H, Q, X, A);

140.

141.

142. cout<<"Task 2 has finished."<<endl;

143. }

144.

145.

146. // \* \* \* \* \* \* \* \* \* \* TASK T3 \* \* \* \* \* \* \* \* \* \*

147. void T3\_B1() {

148. cout<<"Task 3 has started"<<endl;

149. }

150.

151. void T3\_B2() {

152. int B3[N];

153. int MO3[N][N];

154. int alfa3;

155. #pragma omp critical

156. {

157. memmove(B3, B, sizeof(B3));

158. memmove(MO3, MO, sizeof(MO3));

159. alfa3=alfa;

160. }

161.

162. MatricesMul(2\*H, 3\*H, MO3, MX, MAA);

163. MatrixOnVectorMul(2\*H, 3\*H, B3, MAA, X);

164. VectorOnConstantMul(2\*H, 3\*H, Z, alfa3, Q);

165. VectorsAdd(2\*H, 3\*H, Q, X, A);

166. cout<<"Task 3 has finished."<<endl;

167. }

168.

169.

170.

171. // \* \* \* \* \* \* \* \* \* \* TASK T4 \* \* \* \* \* \* \* \* \* \*

172. void T4\_B1() {

173. cout<<"Task 4 has started"<<endl;

174. alfa=1;

175. MatrixInput(MO, 1);

176.

177. }

178.

179. void T4\_B2() {

180. int B4[N];

181. int MO4[N][N];

182. #pragma omp critical

183. {

184. memmove(B4, B, sizeof(B4));

185. memmove(MO4, MO, sizeof(MO4));

186. }

187.

188. MatricesMul(3\*H, 4\*H, MO4, MX, MAA);

189. MatrixOnVectorMul(3\*H, 4\*H, B4, MAA, X);

190. VectorOnConstantMul(3\*H, 4\*H, Z, alfa, Q);

191. VectorsAdd(3\*H, 4\*H, Q, X, A);

192. cout<<"Task 4 has finished."<<endl;

193. }

194.

195.

196.

197. void T5\_B1() {

198. cout<<"Task 5 has started"<<endl;

199. }

200. void T5\_B2() {

201. int B5[N];

202. int MO5[N][N];

203. int alfa5;

204. #pragma omp critical

205. {

206. memmove(B5, B, sizeof(B5));

207. memmove(MO5, MO, sizeof(MO5));

208. alfa5 = alfa;

209. }

210.

211. MatricesMul(4\*H, 5\*H, MO5, MX, MAA);

212. MatrixOnVectorMul(4\*H, 5\*H, B5, MAA, X);

213. VectorOnConstantMul(4\*H, 5\*H, Z, alfa5, Q);

214. VectorsAdd(4\*H, 5\*H, Q, X, A);

215. cout<<"Task 5 has finished."<<endl;

216. }

217.

218. void T6\_B1() {

219. cout<<"Task 6 has started"<<endl;

220. MatrixInput(MX, 1);

221. }

222. void T6\_B2() {

223. int B6[N];

224. int MO6[N][N];

225. int alfa6;

226. #pragma omp critical

227. {

228. memmove(B6, B, sizeof(B6));

229. memmove(MO6, MO, sizeof(MO6));

230. alfa6 = alfa;

231. }

232.

233. MatricesMul(5\*H, N, MO6, MX, MAA);

234. MatrixOnVectorMul(5\*H, N, B6, MAA, X);

235. VectorOnConstantMul(5\*H, N, Z, alfa6, Q);

236. VectorsAdd(5\*H, N, Q, X, A);

237. cout<<"Task 6 has finished."<<endl;

238. }

239.

240.

241.

242.

243.

244.

245. // \* \* \* \* \* \* \* \* \* \* MAIN \* \* \* \* \* \* \* \* \* \*

246. int \_tmain(int argc, \_TCHAR\* argv[])

247. {

248.

249. cout<<"Lab4 has started"<<endl;

250. time\_t startTime;

251.

252. double start = clock();

253. omp\_set\_num\_threads(P);

254. #pragma comment(linker, "/STACK:200000000")

255. #pragma omp parallel

256. {

257. switch (omp\_get\_thread\_num()) {

258. case 0: T1\_B1();

259. break;

260. case 1: T2\_B1();

261. break;

262. case 2: T3\_B1();

263. break;

264. case 3: T4\_B1();

265. break;

266. case 4: T5\_B1();

267. break;

268. case 5: T6\_B1();

269. break;

270. }

271. #pragma omp barrier

272. switch (omp\_get\_thread\_num()) {

273. case 0: T1\_B2();

274. break;

275. case 1: T2\_B2();

276. break;

277. case 2: T3\_B2();

278. break;

279. case 3: T4\_B2();

280. break;

281. case 4: T5\_B2();

282. break;

283. case 5: T6\_B2();

284. break;

285.

286.

287. }

288.

289. #pragma omp barrier

290. if (omp\_get\_thread\_num() == 3)

291. T1\_B3();

292. }

293. printf("%.4lf\n", (clock() - start) / CLOCKS\_PER\_SEC);

294.

295. cout<<"Lab4 has finished."<<endl;

296. getchar();

297. }

# Додаток Г

1. ------------------------------------------------------------------

2. -- Course work

3. -- Ada

4. -- Tkachenko Igor IO-01

5. with Ada.Text\_IO; use Ada.Text\_IO;

6. with Ada.Integer\_Text\_IO; use Ada.Integer\_Text\_IO;

7. with Ada.Calendar; use Ada.Calendar;

8. with ada.float\_text\_io; use ada.float\_text\_io;

9.

10.

11.

12.

13.

14. procedure main is

15. N: integer :=6;

16. P: integer := 6;

17. H: Integer := N / P;

18.

19.

20. type vector is array(integer range <>) of integer;

21. Subtype Vector\_N is Vector(1..N);

22. Subtype Vector\_5H is Vector(1..5\*H);

23. Subtype Vector\_4H is Vector(1..4\*H);

24. Subtype Vector\_3H is Vector(1..3\*H);

25. Subtype Vector\_2H is Vector(1..2\*H);

26. Subtype Vector\_H is Vector(1..H);

27.

28. type Matrix is array(integer range <>) of Vector\_N;

29. Subtype Matrix\_N is Matrix(1..N);

30. Subtype Matrix\_5H is Matrix(1..5\*H);

31. Subtype Matrix\_4H is Matrix(1..4\*H);

32. Subtype Matrix\_3H is Matrix(1..3\*H);

33. Subtype Matrix\_2H is Matrix(1..2\*H);

34. Subtype Matrix\_H is Matrix(1..H);

35.

36. Time\_all: duration;

37. time1,time2: time;

38.

39. task T1 is

40.

41. entry allother(MOc: in Matrix\_N; alfax: in Integer; MXx: in matrix\_h);

42. entry REZ(AA: in Vector\_5h);

43. end T1;

44.

45. task T2 is

46. entry BZ5H(Bx: in Vector\_N; z5h: in Vector\_5h);

47. entry MX2h(MXx: in MATRIX\_2h; MOc:in Matrix\_N; alfax:Integer);

48. entry REZ(AA: in Vector\_4H);

49.

50. end T2;

51.

52. task T3 is

53. entry BZ4H( Zx: in Vector\_4h; Bx: in Vector\_N);

54. entry MX3H(MXx: in Matrix\_3h);

55. entry REZ(AA: in Vector\_3H);

56.

57. end T3;

58.

59. task T4 is

60. entry MOalfaBZ3h(MOx: in matrix\_N;alfax: in integer; Bx: in Vector\_N; Zx : in Vector\_3h);

61. entry MX4H(MXx: in Matrix\_4h);

62.

63. entry REZ(AA: in Vector\_2H);

64.

65. end T4;

66.

67. task T5 is

68. entry MOalfaBZ2h(MOx: in matrix\_N;alfax: in integer; Bx: in Vector\_N; Zx : in Vector\_2h);

69. entry MX5H(MZx: in Matrix\_5h);

70. entry REZ(AA: in Vector\_H);

71. end T5;

72.

73. task T6 is

74. entry MOalfaBZh(MOx: in matrix\_N;alfax: in integer; Bx: in Vector\_N; Zx : in Vector\_h);

75.

76. end T6;

77.

78. task body T1 is

79. Sum1: Integer := 0;

80. MO: Matrix\_N;

81. MX: Matrix\_h;

82. Z: Vector\_N;

83. B: Vector\_N;

84. alfa: integer;

85.

86. A: Vector\_N;

87. MTx: Matrix\_H;

88. begin

89. put\_Line("T1 started!");

90.

91. for i in 1 .. N loop

92. B(i):=1;

93. Z(i):=1;

94. end loop;

95.

96. accept allother(MOc: in Matrix\_N; alfax: in Integer; MXx: in matrix\_h) do

97.

98. MO := MOc;

99. alfa:= alfax;

100. MX:=MXx;

101. end allother;

102.

103. T2.BZ5H(B,Z(h+1 .. N));

104.

105. for i in 1.. N loop

106. for j in 1 .. H loop

107. sum1 := 0;

108. for z in 1 .. N loop

109. Sum1 := sum1 + MX(J)(Z) \* MO(Z)(I);

110. end loop;

111. MTx(J)(I) := Sum1;

112. end loop;

113. end loop;

114. for i in 1.. H loop

115. Sum1 := 0;

116. for j in 1 .. N loop

117. Sum1 := sum1 + B(J) \* MTx(I)(J);

118. end loop;

119. A(I) := Sum1 +Z(i);

120. end loop;

121.

122. accept REZ(AA: in Vector\_5h) do

123. A(h+1 .. N) := AA;

124. end REZ;

125. if N < 36 then

126.

127. for i in 1 .. N loop

128. Put(A(i), 4);

129. end loop;

130.

131. end if;

132. time2:=clock;

133. time\_all:=time2-time1;

134. put\_line("TIME");

135. put (Float(Time\_all), 4, 3, 0);

136.

137. put\_Line("T1 end!");

138. end T1;

139.

140. task body T2 is

141. Sum1: Integer := 0;

142. MO: Matrix\_N;

143. MX: Matrix\_2H;

144. Z: Vector\_5h;

145. B: Vector\_n;

146. alfa: integer;

147. A: Vector\_5H;

148. MTx: Matrix\_H;

149. begin

150. put\_Line("T2 started!");

151.

152.

153. accept MX2h(MXx: in MATRIX\_2h; MOc:in Matrix\_N; alfax:Integer) do

154. MX := MXx;

155. MO:=MOc;

156. alfa:= alfax;

157. end MX2h;

158.

159. T1.allother(MO, Alfa, MX(1 .. h));

160.

161. accept BZ5H(Bx: in Vector\_N; z5h: in Vector\_5h) do

162. B := Bx;

163. Z:= Z5h;

164.

165. end BZ5h;

166.

167. T3.BZ4H( Z(h+1 .. 5\*h), B);

168.

169. for i in 1.. N loop

170. for j in 1 .. H loop

171. sum1 := 0;

172. for z in 1 .. N loop

173. Sum1 := sum1 + MX(J)(Z) \* MO(Z)(I);

174. end loop;

175. MTx(J)(I) := Sum1;

176. end loop;

177. end loop;

178. for i in 1.. H loop

179. Sum1 := 0;

180. for j in 1 .. N loop

181. Sum1 := sum1 + B(J) \* MTx(I)(J);

182. end loop;

183. A(I) := Sum1 + alfa\*Z(I);

184. end loop;

185.

186. accept REZ(AA: in Vector\_4H) do

187. A(h+1 .. 5\*H) := AA;

188. end REZ;

189.

190. T1.REZ(A);

191.

192. Put\_Line("T2 ended");

193. end T2;

194.

195.

196.

197. task body T3 is

198. Sum1: Integer := 0;

199. MO: Matrix\_N;

200. MX: Matrix\_3H;

201. B: Vector\_N;

202. Z: Vector\_4H;

203. alfa: integer;

204. A: Vector\_4H;

205. MTx: Matrix\_H;

206. begin

207. put\_Line("T3 started!");

208.

209. --1. take inf from T2

210. for i in 1 .. N loop

211. for j in 1 .. N loop

212. MO(i)(j) := 1;

213. end loop;

214. end loop;

215. alfa:=1;

216.

217.

218. accept MX3H(MXx: in Matrix\_3h) do

219. MX:=MXx;

220. end MX3H;

221. t2.MX2h(MXx => MX(1..2\*h),

222. MOc => MO,

223. alfax => alfa);

224.

225. accept BZ4H(Zx: in Vector\_4h; Bx: in Vector\_N) do

226. Z := Zx;

227. B:=Bx;

228. end BZ4H;

229. T4.MOalfaBZ3h(MO,alfa, B, Z(h+1 .. 4\*h));

230.

231. for i in 1.. N loop

232. for j in 1 .. H loop

233. sum1 := 0;

234. for z in 1 .. N loop

235. Sum1 := sum1 + Mx(J)(Z) \* MO(Z)(I);

236. end loop;

237. MTx(J)(I) := Sum1;

238. end loop;

239. end loop;

240. for i in 1.. H loop

241. Sum1 := 0;

242. for j in 1 .. N loop

243. Sum1 := sum1 + b(J) \* MTx(I)(J);

244. end loop;

245. A(I ) := Sum1 + Alfa\*Z(I);

246. end loop;

247.

248. accept REZ(AA: in Vector\_3H) do

249. A(h+1 .. 4\*H) := AA;

250. end REZ;

251.

252. T2.REZ(A);

253.

254. Put\_Line("T3 ended");

255. end T3;

256.

257. task body T4 is

258. Sum1: Integer := 0;

259. MO: Matrix\_N;

260. MX: Matrix\_4H;

261. B: Vector\_N;

262. Z: Vector\_3H;

263. alfa: Integer;

264. A: Vector\_3H;

265. MTx: Matrix\_H;

266. begin

267. put\_Line("T4 started!");

268.

269. accept MX4H(MXx: in Matrix\_4h) do

270. MX := MXx;

271.

272. end MX4H;

273.

274. T3.MX3h(MX(1 .. 3\*h));

275.

276. accept MOalfaBZ3h(MOx: in matrix\_N;alfax: in integer; Bx: in Vector\_N; Zx : in Vector\_3h) do

277. MO := MOx;

278. alfa:= alfax;

279. b:= bx;

280. Z := Zx;

281. end MOalfaBZ3h;

282.

283. T5.MOalfaBZ2h(MO,alfa,B,Z(h+1 .. 3\*h));

284.

285. for i in 1.. N loop

286. for j in 1 .. H loop

287. sum1 := 0;

288. for z in 1 .. N loop

289. Sum1 := sum1 + MX(J)(Z) \* MO(Z)(I);

290. end loop;

291. MTx(J)(I) := Sum1;

292. end loop;

293. end loop;

294. for i in 1.. H loop

295. Sum1 := 0;

296. for j in 1 .. N loop

297. Sum1 := sum1 + B(J) \* MTx(I)(J);

298.

299. end loop;

300.

301. A(I) := Sum1 + alfa\*Z(i );

302.

303. end loop;

304.

305. accept REZ(AA: in Vector\_2H) do

306. A(h+1 .. 3\*H) := AA;

307. end REZ;

308.

309. T3.REZ(A);

310.

311. Put\_Line("T4 ended");

312. end T4;

313.

314. task body T5 is

315. Sum1: Integer := 0;

316. MO: Matrix\_N;

317. MX: Matrix\_5H;

318. B: Vector\_N;

319. z: Vector\_2H;

320. alfa: integer;

321. A: Vector\_2H;

322. MTx: Matrix\_H;

323. begin

324. put\_Line("T5 started!");

325.

326. accept MX5H(MZx: in Matrix\_5h) do

327. MX := MZx;

328.

329. end MX5H;

330.

331. T4.MX4H(MX(1 .. 4\*h));

332. accept MOalfaBZ2h(MOx: in matrix\_N;alfax: in integer; Bx: in Vector\_N; Zx : in Vector\_2h) do

333. B := Bx;

334. Z := Zx;

335. MO:= MOx;

336. alfa:= alfax;

337. end MOalfaBZ2h;

338.

339. T6.MOalfaBZh(MO,alfa, B, Z(h+1 .. 2\*h));

340.

341. for i in 1.. N loop

342. for j in 1 .. H loop

343. sum1 := 0;

344. for z in 1 .. N loop

345. Sum1 := sum1 + MX(J)(Z) \* MO(Z)(I);

346. end loop;

347. MTx(J)(I) := Sum1;

348. end loop;

349. end loop;

350. for i in 1.. H loop

351. Sum1 := 0;

352. for j in 1 .. N loop

353. Sum1 := sum1 + B(J) \* MTx(I)(J);

354. end loop;

355. A(I) := Sum1 + alfa\* z(2);

356.

357. end loop;

358.

359. accept REZ(AA: in Vector\_H) do

360. A(h+1 ..2\* H) := AA;

361. end REZ;

362.

363. T4.REZ(A);

364.

365. Put\_Line("T5 ended");

366. end T5;

367.

368. task body T6 is

369. Sum1: Integer := 0;

370. MO: Matrix\_N;

371. MX: Matrix\_N;

372. Z: Vector\_h;

373. B: Vector\_N;

374. alfa: integer;

375. A: Vector\_h;

376. MTx: Matrix\_H;

377. begin

378. put\_Line("T6 started!");

379.

380. for i in 1 .. N loop

381. for j in 1..N loop

382. MX(i)(j):=1;

383. end loop;

384. end loop;

385.

386. T5.MX5H(MX(1 .. 5\*h));

387. alfa:=0;

388. accept MOalfaBZh(MOx: in matrix\_N;alfax: in integer; Bx: in Vector\_N; Zx : in Vector\_h) do

389. MO := MOx;

390. Z := Zx;

391. alfa:= alfa;

392. B:=Bx;

393.

394. end MOalfaBZh;

395.

396. for i in 1.. N loop

397. for j in 1 .. H loop

398. sum1 := 0;

399. for z in 1 .. N loop

400. Sum1 := sum1 + MX(J)(Z) \* MO(Z)(I);

401. end loop;

402. MTx(J)(I) := Sum1;

403. end loop;

404. end loop;

405. for i in 1.. H loop

406. Sum1 := 0;

407. for j in 1 .. N loop

408. Sum1 := sum1 + B(J) \* MTx(I)(J);

409. end loop;

410. A(I) := Sum1 +Z(I);

411. end loop;

412.

413.

414. t5.REZ(A);

415.

416. Put\_Line("T6 ended");

417. end T6;

418.

419. begin

420.

421. time1:=clock;

422.

423. null;

424. end main;

425.

# Додаток Д

## 

## Розділ 2

## Розділ3