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 ІО-23 групи

напряму підготовки 050102 «Комп’ютерна інженерія»

Вітриченко А.А

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- 2015 рік

Національний технічний університет України

“Київський політехнічний інститут”

Факультет 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

Вітриченко Андрій Андрійович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Тема роботи «Розробка програмного забезпечення для паралельних

комп’ютерних систем»

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

( прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом роботи 18 травня 2014 р.

3. Вхідні дані до роботи

- засоби роботи з процесами в Win32

- математична задача МА = МВ\*МС

- структури ПКС ОП та ПКС ЛП

- мови і бібліотеки програмування: Ада ,C#

- засоби організації взаємодії процесів: захищений модуль, механізм

рандеву мови Ада

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

- огляд засобів роботи з процесами в Win32

- розробка і тестування програми ПРГ1 для ПКС ОП

- розробка і тестування програми ПРГ2 для ПКС ЛП

5. Перелік графічного матеріалу

- структурна схема ПКС ОП

- структурна схема ПКС ЛП

- схеми алгоритмів процесів і головної програми для ПРГ1

- схеми алгоритмів процесів і головної програми для ПРГ2.

7. Дата видачі завдання 2.02. 2015

#### КАЛЕНДАРНИЙ ПЛАН

|  |  |  |
| --- | --- | --- |
| №  з/п | Назва етапів виконання КР | Строк виконання етапів КР |
| 1 | Виконання Розділу 1 | 23.02.2015 |
| 2 | Виконання Розділу 2 | 23.03.2015 |
| 3 | Виконання Розділу 3 | 23.04.2015 |
| 4 | Тестування програм ПРГ1 та ПРГ2 | 10.05.2015 |
| 7 | Оформлення КР | 17.05.2015 |
| 8 | Захист КР | 18.05.2015 |

**Студент \_\_\_\_\_\_\_\_\_** \_\_\_\_\_\_Вітриченко А.А.\_\_\_\_\_

( підпис ) (прізвище та ініціали)

**Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_** \_\_\_\_\_\_\_Корочкін А.В.\_\_\_\_\_\_

( підпис ) (прізвище та ініціали)

ЗМІСТ

З А В Д А Н Н Я 2

ВСТУП 5

РОЗДІЛ 1. Засоби роботи з процессами в Win32 6

Реалізація семафорів у мові Ада 7

Реалізація семафорів у мові Java 10

Реалізація семафорів в мові C# 15

Реалізація семафорів у бібліотеці Win32 21

Висновки до розділу 1: 23

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС ЗП 25

2.1. Розробка паралельного математичного алгоритму 25

2.2. Розробка алгоритмів процесів 26

2.3. Розробка схеми взаємодії процесів 28

2.4. Розробка програми ПРГ1 29

Тестування програми ПРГ1 30

Висновки до розділу 2 32

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС ЛП 33

Розробка паралельного математичного алгоритму 33

Розробка алгоритмів процесів 34

Розробка схеми взаємодії процесів 36

Рис. 3.2. Схема для синхронізації та доступу до ЗР 37

Розробка програми ПРГ2 38

Тестування програми ПРГ2 38

Висновки до розділу 3 40

ДОДАТКИ 42

Додаток А. 42

Лістинг до ПРГ1 42

Додаток B. 51

Лістинг до ПРГ2 51

# ВСТУП

В даній курсовій роботі в першому розділі розглянуто різноманітні реалізації семафорів – спеціальних механізмів для вирішення задач взаємного виключення та синхронізації. Описані реалізації в мовах програмування Ада, Java, C# і бібліотеці Win32 та проведено їх порівняння.

В другому розділі розроблено програму для обчислення математичної задачі у паралельній комп’ютерній системі з загальною пам’яттю. Описані етапи розробки програми та наведено сирцевий код. Проведено тестування отриманого програмного продукту та зроблено відповідні висновки.

В третьому розділі розроблено програму для обчислення математичної задачі у паралельній комп’ютерній системі з локальною пам’яттю. Описані етапи розробки програми та наведено сирцевий код. Проведено тестування отриманого програмного продукту та зроблені відповідні висновки.

Після цих розділів зроблено загальні висновки до курсової роботи. Наведено список використаних інформаційних джерел, алгоритми та лістинг програмного коду до програм з розділів 2 і 3.

# РОЗДІЛ 1. Засоби роботи з процессами в Win32

Семафор — це універсальний механізм для організації взаємодії процесів [7]. Він використовується для розв'язання задачі взаємного виключення та синхронізації потоків. Даний механізм є одним з найстаріших засобів розподілення доступу процесів, що працюють паралельно, до критичних ресурсів. Семафори використовуються для контролю доступу до спільного ресурсу, або для синхронізації процесів (потоків).

Семафор — це об'єкт ядра ОС, який можна розглядати як лічильник, що містить ціле число в діапазоні від 0 до заданого при його створенні максимального значення. При досягненні семафором значення 0 він переходить у несигнальний стан, при будь-яких інших значеннях лічильника – його стан сигнальний.

Традиційне позначення семафора : S. Операції, які можна виконати над семафором:

Ініціалізація - встановлення початкового значення семафору.

Операція P(S): Вона перевіряє стан семафору. Якщо семафор не рівний нулю, то виконується операція S:=S-1. Інакше, процес блокується, поки S=0.

Операція V(S): Ця операція збільшує значення семафору на 1. Тобто виконується операція S:=S+1.

Типи семафорів

В залежності від значень, які може приймати семафор він поділяється на:

* Двійковий : здатний приймати значення 0 та 1.
* Множинний : здатний приймати значення від 0 до n (n>1).

Семафори реалізовано у мовах Ада, Java, C# та у бібліотеці Win32.

## Реалізація семафорів у мові Ада

У стандарті мови Ада механізм семафорів реалізовано у вигляді пакету Ada.Synchronous\_Task\_Control, який вміщує в себе необхідні функції для роботи з семафорами. Пакет реалізує механізм семафорів наступним чином: семафорний тип забезпечується приватним типом Suspension\_object, який є лімітованим захищеним типом.

Специфікація пакету:

package Ada.Synchronous\_Task\_Control is

type Suspension\_Object is limited private;

procedure Set\_True(S : in out Suspension\_Object);

procedure Set\_False(S : in out Suspension\_Object);

function Current\_State(S : Suspension\_Object) return Boolean;

procedure Suspend\_Until\_True(S : in out Suspension\_Object);

private

... *-- not specified by the language*

end Ada.Synchronous\_Task\_Control;

- procedure Suspend\_until\_true (S: in out Suspension\_object), що є реалізацією операції P(S);

- procedure Set\_true (S: in out Suspension\_object), що є реалізацією операції V(S);

- procedure Set\_false (S: in out Suspension\_object), що є додатковою операцією та встановлює семафору значення false;

- function Current\_state (S: Suspension\_object) return Boolean, що є додатковою операцією та перевіряє значення семафору.

Семафор у мові Ада є бінарним і може приймати два значення – true та false. При створенні семафору йому за замовчанням присвоюється значення false. Вирішення задачі взаємного виключення з використанням семафорів мови Ада ілюструється наступним прикладом, в якому за допомогою семафору обмежується доступ до загального ресурсу buf:

with Ada.Synchronous\_Task\_Control;

use Ada.Synchronous\_Task\_Control;

with Ada.TEXT\_IO; use Ada.TEXT\_IO;

procedure ZVV is

buf : integer; --загальний ресурс

--семафор

S : Suspension\_Object; --false

--задачі

procedure Start is

task T1;

task body T1 is

begin

put ("T1 started");

-- Задачі, які даний потік виконує паралельно

Suspend\_Until\_True(S);

buf := buf+100; --критична ділянка, доступ до ЗР

Set\_True(S);

--інші паралельні дії

put ("T1 finished");

end T1;

task T2;

task body T2 is

begin

put("T2 started);

-- Задачі, які даний потік виконує паралельно

Suspend\_Until\_True(S);

buf := buf\*78-126; --критична ділянка, доступ до ЗР

Set\_True(S);

--інші паралельні дії

put ("T2 finished");

end T2;

begin

null;

end Start;

begin

-- задаємо семафору значення «вільно»

-- оскільки спочатку ресурс вільний

Set\_True(S);

Start;

end ZVV;

Вирішення задачі синхронізації з використанням семафорів мови Ада ілюструється наступним прикладом, в якому перший потік посилає сигнал другому, перемикаючи семафор в стан true:

with Ada.Synchronous\_Task\_Control;

use Ada.Synchronous\_Task\_Control;

with Ada.TEXT\_IO; use Ada.TEXT\_IO;

procedure ZS is

--семафор

S : Suspension\_Object; --false

--задачі

procedure Start is

task T1;

task body T1 is

begin

put ("T1 started");

-- Задачі, які даний потік виконує паралельно

Set\_True(S); --сигнал від задачі Т1 до Т2

--інші паралельні дії

put ("T1 finished");

end T1;

task T2;

task body T2 is

begin

put ("T2 started");

-- Задачі, які даний потік виконує паралельно

Suspend\_Until\_True(S); --очікування сигналу з Т1

--інші паралельні дії

put ("T2 finished");

end T2;

begin

null;

end Start;

begin --main

Start;

end ZS;

## Реалізація семафорів у мові Java

В мові Java семафори реалізовано класом Semaphore пакету java.util.concurent [2]. Цей клас має два конструктори – Semaphore (int X), де X – початкове значення семафору та Semaphore (IntX, Boolean Y), де X – максимальне значення семафору, а Y – поведінка P(S) та V(S). Якщо Y = true, то розблокування процесів буде відбуватися по черзі, якщо false, то процеси будуть розблоковуватись випадковим чином. В класі Semaphore визначено такі операції для роботи з семафорами:

* acquire() – метод, що реалізує операцію P(S), підчас якої виконується перевірка значення дозволів семафору, якщо значення семафору більше або дорівнює одиниці, то від значення семафору віднімається одиниця і потік продовжує своє виконання, якщо значення менше, то потік блокується до моменту коли умова виконується. Метод acquire() має модифікацію в якій задається число, на скільки зменшити семафор. Також в Java реалізований схожий метод acquireUninterruptibly, цей метод отримує дозвіл від семафору, блокуючи потік, доки дозвіл не буде доступний. Якщо є доступний дозвіл, то отримує його и миттєво повертається, зменшуючи кількість доступних дозволів на одиницю. Якщо немає доступних дозволів, то даний потік стає недіючим для розкладу потоків і очікує, доки інший потік не викличе метод release() для цього семафору і даний потік стає наступним, кому буде надано дозвіл. Якщо даний потік переривається, очікуючи на дозвіл, то він продовжить очікувати, але час, за який потік отримає дозвіл, буде змінено порівняно з часом, за який він отримав би дозвіл без переривання. Коли потік повертається з цього методу, його статус переривання буде встановлено. Тобто цей метод схожий до методу acquire(), але переривання не розблокує потік, а лише змінить час його очікування на дозвіл від семафору. У даного методу є модифікація, за якої задається кількість дозволів.
* availablePermits() – метод, що повертає кількість наявних в цьому семафорі дозволів.
* release() – метод, що реалізує операцію V(S), підчас якої виконується збільшення кількості дозволів (повернення) в семафор на один. Якщо всі потоки намагаються отримати дозвіл, то вибирається потік і йому надається дозвіл. Цей потік включається в список планування потоків. Даний метод може приймати, в якості параметру, на скільки дозволів збільшити семафор.
* drainPermits() – метод, що отримує та повертає всі дозволи, котрі зразу стають доступними.
* getQueuedThreads() - повертає колекцію потоків, які можуть очікувати дозвіл. Оскільки фактичний набір потоків може змінитися під час виконання методу, результат може бути не достовірним. Елементи колекції не мають порядку. Цей метод призначений для поліпшення побудови дочірніх класів, котрі мають більш широкі можливості контролю.
* getQueueLength() - повертає оцінку числа потоків, які очікують дозволу. Значення тільки оцінка, тому що число потоків може мінятися динамічно під час виконання методу. Цей метод призначений для використання в сфері моніторингу стану системи, а не для синхронізації.
* hasQueuedThreads() - надсилає запит, чи чекають якісь потоки методу acquire().
* isFair() - повертає істину, якщо справедливість семафору встановлена в істину.
* reducePermits() - зменшує кількість дозволів. Цей метод може бути корисним в підкласах, котрі використовують семафори для моніторингу ресурсів. Основна відмінність від методу acquire() в тому, що він не блокує потік доки дозвіл не стане доступним. Даний метод може приймати, в якості параметру, на скільки зменшити кількість дозволів, якщо без параметрів то на 1.
* tryAcquire() - отримує вказану кількість дозволів від цього семафора, тільки якщо всі вони доступні під час виклику методу і відразу ж повертається зі значенням доступних дозволів скороченим на задану величину. Якщо дозволів немає, то цей метод повертає негайно зі значенням помилки і число доступних дозволів залишається незмінним. Даний метод може приймати в якості параметрів: кількості дозволів, час очікування виконання методу і кількість дозволів та час очікування виконання методу.

В мові Java на відміну від мови Ада лічильники семафорів є чисельним (а не булевого типу). Це означає, що в Java можливо створювати множинні семафори, що значно спрощує вирішення задачі синхронізації для більше ніж двох процесів.

Використання семафорів в мові програмування Java для вирішення задачі взаємного виключення (доступу до загального ресурсу - buf) наведенов наступному прикладі

import java.util.concurrent.\*;

public class ZVV {

public static int *buf*;

public static Semaphore *S* = new Semaphore(1);

public static void main(String[] args) {

Thread1 T1 = new Thread1();

Thread2 T2 = new Thread2();

T1.start();

T2.start();

}

}

public class Thread1 extends Thread{

public void run(){

System.*out*.println("T1 started");

//паралельні дії потоку

try {

ZVV.*S*.acquire();

} catch (InterruptedException e) {

e.printStackTrace();

}

ZVV.*buf* = Cursova1.*buf*+100; // критична ділянка

ZVV.*S*.release();

//ще якісь паралельні дії

System.*out*.println("T1 finished");

}

}

public class Thread2 extends Thread{

public void run(){

System.*out*.println("T2 started");

//паралельні дії потоку

try {

ZVV.*S*.acquire();

} catch (InterruptedException e) {

e.printStackTrace();

}

ZVV.*buf* = Cursova1.*buf*\*74+26; // критична ділянка

ZVV.*S*.release();

//ще якісь паралельні дії

System.*out*.println("T1 finished");

}

}

Використання семафорів в мові програмування Java для вирішення задачі синхронізації (другий потік чекає сигналу від першого, для продовження роботи) наведено в наступному прикладі

import java.util.concurrent.\*;

public class ZS {

public static Semaphore *S* = new Semaphore(0);

public static void main(String[] args) {

Thread1 T1 = new Thread1();

Thread2 T2 = new Thread2();

T1.start();

T2.start();

}

}

public class Thread1 extends Thread{

public void run(){

System.*out*.println("T1 started");

//паралельні дії потоку

ZS.*S*.release(); //сигнал

// ще якісь паралельні дії

System.*out*.println("T1 finished");

}

}

public class Thread2 extends Thread{

public void run(){

System.*out*.println("T2 started");

//паралельні дії потоку

try {

ZS.*S*.acquire(); //очікування сигналу

} catch (InterruptedException e) {

e.printStackTrace();

}

//ще якісь паралельні дії

System.*out*.println("T1 finished");

}

}

## Реалізація семафорів в мові C#

Семафори в C# реалізовані в двох класах: Semaphore, SemaphoreSlim[6]. Клас Semaphore дозволяє доступ до ресурсу певній кількості потоків. Додаткові потоки, що запитують ресурс, блокуються до звільнення потоком семафора. Він є похідним від класу WaitHandle та може бути локальним або глобальним.

Клас Semaphore не підтримує схожість потоків. Це означає, що він може використовуватися в сценаріях, в яких один потік отримує семафор, а інший потік звільняє його.

Потоки входять в семафор за допомогою виклику методу WaitOne, який успадковується з класу WaitHandle. При поверненні виклику лічильник на семафорі зменшується. Якщо при запиті потоком запису лічильник дорівнює нулю - потік блокується. Потоки звільняють семафор за допомогою виклику методу Semaphore.Release, після чого відкривається доступ для заблокованих потоків. Для входу блокованих потоків в семафор не існує гарантованого порядку, такого як "першим прийшов, першим вийшов" (FIFO) або "останнім прийшов, першим вийшов" (LIFO).

Потік може виконувати вхід в семафор кілька разів, викликаючи багаторазово метод WaitOne. Щоб звільнити семафор, потік може викликати перевантажену версію методу Release() ту ж кількість разів або викликати перевантажену версію методу Release( Int32 ) і вказати кількість записів, яку необхідно звільнити.

Операційна система Windows дозволяє привласнювати семафорам імена. Іменований семафор відноситься до всієї системи. Тобто після створення іменованого семафора він стає видимим у всіх потоках і у всіх процесах системи. Тому іменований семафор може використовуватися для синхронізації дій процесів і потоків. Можна створити об'єкт Semaphore, який представляє іменований системний семафор за допомогою одного з конструкторів, що вказують ім'я. Так як іменовані семафори відносяться до всієї системи, інший процес, що використовує те ж ім'я, може несподівано увійти в семафор. Зловмисний код, що виконується на тому ж комп'ютері, може використовувати це як основу для атак по типу "відмова в обслуговуванні". Так як іменовані семафори відносяться до всієї системи, можна мати декілька об'єктів Semaphore, які представлятимуть один і той же іменований семафор. Кожен виклик конструктора іменованого семафору або методу Semaphore.OpenExisting створює новий об'єкт Semaphore. Вказання ​​того ж імені повторно створює кілька об'єктів, які представляють той же іменований семафор.

В класі Semaphore визначені наступні методи:

* WaitOne() - блокує даний потік до отримання сигналу. Цей метод має декілька модифікацій з різними параметрами за яких до поведінки методу додаються такі дії: спостерігання за CancellationToken; використовуючи 32-розрядне число, яке визначає тайм-аут; використовуючи TimeSpan щоб вказати тайм-аут; використовуючи 32-розрядне число, яке визначає очікування, і вказівки, чи потрібно виходити з домену синхронізації до закінчення очікування; використовуючи TimeSpan, який визначає час очікування, і вказівки, чи потрібно виходити з домену синхронізації до закінчення очікування;

- Dispose() - звільняє всі ресурси, використовувані поточним екземпляром класу WaitHandle.

* Release()- виходить з семафора і повертає попереднє значення лічильника. Є модифікація для виходу з семафору задану кількість разів.
* TryOpenExisting() - відкриває вказаний іменований семафор, якщо він вже існує, і повертає значення, яке вказує, чи є операція успішної. Даний метод має модифікацію в якій: метод відкриває вказаний іменований семафор, якщо він вже існує, з бажаною безпекою доступу і повертає значення, яке вказує, чи є операція успішною.

- System.Threading.SemaphoreSlim - це спрощений, швидкий семафор, який можна використовувати для очікування усередині одного процесу, коли передбачається, що часи очікування будуть дуже короткими. Клас SemaphoreSlim в максимально можливій мірі використовує примітиви синхронізації, надані середовищем CLR. Але він також надає неактивно ініціалізовані дескриптори очікування на основі ядра в якості необхідних засобів підтримки очікування для декількох семафорів. Клас SemaphoreSlim також підтримує використання токенів скасування, але не підтримує іменовані семафори і використання дескриптора очікування для синхронізації. Даний семафор має два конструктори: конструктор ініціалізації нового екземпляра класу SemaphoreSlim, з зазначенням початкового кількість запитів, які можуть бути надані одночасно; та конструктор ініціалізації нового екземпляра класу SemaphoreSlim, з зазначенням початкового та максимального числа запитів, які можуть бути надані одночасно.

SemaphoreSlim має методи [6]:

* Release() - виходить з семафору один раз. Даний метод може приймати в якості параметру кількість виходів з семафору.
* Wait()- блокує поточний потік, поки він не може увійти в SemaphoreSlim. Цей метод має декілька модифікацій з різними параметрами за яких до поведінки методу додаються такі дії: спостерігання за CancellationToken; використовуючи 32-розрядне число, яке визначає тайм-аут; використовуючи TimeSpan щоб вказати тайм-аут; використовуючи 32-розрядне число, яке визначає очікування, спостерігаючи за CancellationToken; використовуючи TimeSpan, який визначає час очікування, спостерігаючи CancellationToken;
* WaitAsync() - асинхронне чекання, щоб увійти в SemaphoreSlim. Цей метод має декілька модифікацій з різними параметрами за яких до поведінки методу додаються такі дії: спостерігання за CancellationToken; використовуючи 32-розрядне число, яке визначає тайм-аут; використовуючи TimeSpan щоб вказати тайм-аут; використовуючи 32-розрядне число, яке визначає очікування, спостерігаючи за CancellationToken; використовуючи TimeSpan, який визначає час очікування, спостерігаючи CancellationToken[4].

Обидва семафори в C# можуть викликати виключення SemaphoreFullException - виключення, яке виникає при Semaphore.Release, коли лічильник вже має максимальне значення.

Лічильники семафорів в C#, як і в Java, є числами, тому семафори можуть бути множинними.

Приклад використання семафорів в мові C# для вирішення задачі взаємного виключення (доступ до спільного ресурсу - buf):

using System;

using System.Threading;

using System.Text;

namespace ConsoleApplication1

{

class ZVV

{

static Semaphore S1 = new Semaphore(0, 1);

static int buf;

static void T1() {

Console.WriteLine("T1 started");

//паралельні дії потоку

S1.WaitOne();

buf = buf + 123; //критична ділянка

S1.Release();

// інші паралельні дії потоку

Console.WriteLine("T1 finished");

}

static void T2()

{

Console.WriteLine("T2 started");

// паралельні дії потоку

S1.WaitOne();

buf = buf\*10; //критична ділянка

S1.Release();

//інші паралельні дії потоку

Console.WriteLine("T2 finished");

}

static void Main(string[] args)

{

Thread Thread1 = new Thread(T1);

Thread Thread2 = new Thread(T2);

Thread1.Start();

Thread2.Start();

}

}

}

Приклад використання семафорів в мові C# для вирішення задачі синхронізації (доступ до спільного ресурсу - buf):

using System;

using System.Threading;

using System.Text;

namespace ConsoleApplication2

{

class ZS

{

static Semaphore S1 = new Semaphore(0, 1);

static void T1() {

Console.WriteLine("T1 started");

//паралельні дії потоку

S1.Release(); //сигнал

//інші паралельні дії потоку

Console.WriteLine("T1 finished");

}

static void T2()

{

Console.WriteLine("T2 started");

//паралельні дії потоку

S1.Wait(); // очікування на сигнал

//інші паралельні дії потоку

Console.WriteLine("T2 finished");

}

static void Main(string[] args)

{

Thread Thread1 = new Thread(T1);

Thread Thread2 = new Thread(T2);

Thread1.Start();

Thread2.Start();

}

}

}

## Реалізація семафорів у бібліотеці Win32

В бібліотеці Win32 семафори є змінними спеціального типу HANDLE, за якими слідкує сама система. В цій бібліотеці лічильники семафорів є множинними, тобто може набувати значень від 0 до визначеного значення. Якщо семафор дорівнює нулю, то він заборонений, і в разі виклику процесом функції очікування цей процес буде блокований доти, доки інший процес не змінить значення семафора (інкрементує лічильник). Для роботи з семафорами в Win32 є наступний набір функцій:

* **HANDLE CreateSemaphore (**

**LPSECURITY\_ATTRIBUTES lpSemaphoreAttributes,**

**LONG lInitialCount,**

**LONG lMaximumCount,**

**LPCTSTR lpName**

**)** – функція для створення семафору.

lpSemaphoreAttributes – параметри захисту, зазвичай встановлено значення NULL.

lInitialCount – початкове значення, воно має бути більше за 0 або рівним 0, та меншим за lMaximumCount або рівним йому. Стан семафору сигнальний, коли це число більше за 0 та не сигнальний, коли рівне 0. Значення семафору зменшується на 1 коли функція очікування розблоковує потік, який чекав на цей семафор. Значення семафору збільшується на визначене значення викликанням функції ReleaseSemaphore.

lMaximumCount – максимальне значення, має бути більше за 0.

lpName – ім’я семафору, яке має бути не більше за MAX\_PATH та може містити в собі будь-які символи, окрім зворотного слешу(\). Порівняння імен регістрозалежне. Якщо lpName співпадає з вже існуючим семафором, то lInitialCount та lMaximumCount ігноруються, тому що вони вже були встановлені при створенні. Якщо lpName - NULL, то семафор створюється без імені.

Якщо функція успішна, то вона повертає семафор типу HANDLE. Якщо іменований об’єкт семафору існував до виклику функції, то функція GetLastError повертає ERROR\_ ALREADY\_EXISTS. В інших випадках GetLastError повертає нуль. Якщо CreateSemaphore є невдалою, то повертає NULL. Для розширених відомостей про помилку потрібно викликати функцію GetLastError.

* **HANDLE WaitForSingleObject (**

**HANDLE hHandle,**

**DWORD dwMilliseconds**

**)** – реалізація операції P(S),

hHandle – семафор,

dwMilliseconds – визначає час тайм-ауту в мілісекундах. Функція повертає значення коли інтервал часу скінчується, навіть якщо стан семафору несигнальний. Якщо dwMilliseconds дорівнює нулю, то функція визначає стан семафору та повертається миттєво. Якщо dwMilliseconds дорівнює INFINITE, тайм-аут функції ніколи не наступає. Якщо виконання функції успішне, то значення, що вона повертає, визначає яка подія змусила її повернутися. Є два такі значення: WAIT\_OBJECT\_0, яке повертається коли стан семафору стає сигнальним та WAIT\_TIMEOUT, яке повертається коли трапляється тайм-аут та стан семафору є несигнальним. WAIT\_FAILED вказує на не успішне виконання функції. Очікування на некоректний семафор змушує функцію повернути WAIT\_FAILED. Для поширених відомостей про помилку потрібно викликати функцію GetLastError.

* **BOOL ReleaseSemaphore (**

**HANDLE hSemaphore,**

**LONG lReleaseCount,**

**LPLONG lpPreviousCount**

**)** – реалізація операції V(S),

hSemaphore – семафор, який повертається функцією CreateSemaphore.

lReleaseCount – число, на яке його потрібно збільшити, це число повинне бути більше за нуль. Якщо додавання цього числа до значення, яке вже є, зробить його більше за максимальне значення семафору, то додавання не відбудеться і функція поверне FALSE.

lpPreviousCount – попереднє значення семафору. Може бути NULL, якщо попереднє значення не потребується. Якщо функція успішна, то значення, яке вона повертає, є ненульовим. Якщо ні, то повертає нуль. Для поширених відомостей про помилку потрібно викликати функцію GetLastError.

## Висновки до розділу 1:

1. Розглянуто реалізацію семафорів у різних мовах. Показано, що у будь-яких мовах семафори мають операції створення або конструктори, реалізацію операції P(S) та реалізацію операції V(S), що є основними операціями для використання семафорів.
2. Розглянуто сутність семафорів у різних мовах. Було виявлено, що в мові Ада семафори можуть бути тільки двійковими, а в мовах Java, C# та бібліотеці Win32 вони можуть бути множинними, що спрощує задачу синхронізації, оскільки не потрібно створювати багато двійкових семафорів для однотипних сигналів.
3. Виявлено, що в реалізації P(S) у мовах Ада та C# та у бібліотеці Win32 може виконуватись тільки одна дія після виконання умови S==1: S=S-1, а у мові Java може виконуватись також S=S-k, що використовується для багатозначних семафорів та зменшує код порівняно з іншими реалізаціями семафору.
4. Виявлено, що в реалізація V(S) у мові Ада значення семафору може збільшитися лише на 1, а у мовах Java, C# та у бібліотеці Win32 на будь-яке число в межах максимального значення семафору.
5. В результаті вивчення документації по реалізаціям семафорів в різних мовах визначено, що в мові Java можна обрати як будуть розблоковуватися процеси – по черзі або випадково. В інших мовах такої можливості немає.

# РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС ЗП

В роздіді розглянуто розробку та дослідження програми ПРГ1 для ПКС з СП.

Вихідна математична задача: A = a(B+C\*MO)(MK\*MT)-bE

Мова програмування C#

Структуру ПКС СП представлено на рис. 2.1

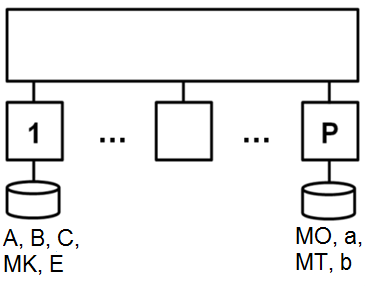


Рис. 2.1. Структура ПКС

## 2.1. Розробка паралельного математичного алгоритму

Відповідно до концепції необмеженого паралелізму:

* Кожна операція виконується за одиницю часу
* Час передачі даних між процесорами не враховується
* Застосовується будь-яка потрібна (необмежена) кількість процесорів

Вихідна математична задача вміщує в собі окремі дії, які не можуть бути виконані паралельно, тобто задача не є повністю паралельною.

При розробці паралельного алгоритму передбачається, що розмірність задачі (N) більше або дорівнює числу процесорів (P) (N> = P), причому N кратне P, що дозволяє розбити задачу на окремі частини Н, кожен з яких обробляється своїм завданням. Якщо N не кратне P, то останній процес буде обробляти частину даних, більшу ніж H.

H=N/P

Таким чином, отримано алгоритм виконання математичної задачі:

1. = +C\*



1. AH = a\*K(\* M)-b\*



Загальний ресурс: C, a, K, MK, b

## 2.2. Розробка алгоритмів процесів

При розробці алгоритмів позначимо перший процес – T1, останній процес – TP, процес між першим і останнім під номером і (1<i<P) – Ti. Тоді алгоритми кожного процесу матимуть наступний вигляд

|  |  |  |
| --- | --- | --- |
| **T1** | | **Точки**  **синхронізації** |
| 1 | Введення B, C, MK, E |  |
| 2 | Сигнал про введення даних для T2 - TP | S1 |
| 3 | Чекати сигнал про завершення введення даних в TP | W1 |
| 4 | Копіювати: C1 = C | **КС** |
| 5 | Обрахувати: = +C1\* |  |
| 6 | Сигнал про готовність для T2 - TP | S2 |
| 7 | Чекати сигнал про готовність від T2 - TP | W2 |
| 8 | Копіювати: a1=a; K1=K; MK1=MK; b1=b | КС |
| 9 | Обрахувати: AH = a1\*K1(\* M)-b1\* |  |
| 10 | Чекати завершення обчислень в T2 - TP | W3 |
| 11 | Вивести А |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Ti** | | **Точки**  **синхронізації** | |
| 1 | Чекати сигнал про завершення введення даних в T1, TP | | W1 | |
| 2 | Копіювати: Ci = C: | КС | |
| 3 | Обрахувати: = +Ci\* |  | |
| 4 | Сигнал про готовність для T1 - TP | S2 | |
| 5 | Чекати сигнал про готовність від T1 - TP | W2 | |
| 6 | Копіювати: ai=a; Ki=K; MKi=MK; bi=b | КС | |
| 7 | Обрахувати: AH = ai\*Ki(\* M)-bi\* |  | |
| 8 | Сигнал про завершення обчислень для Т1 | S3 | |

|  |  |  |
| --- | --- | --- |
| **TP** | | **Точки**  **синхронізації** |
| 1 | Введення MO, a, MT, b |  |
| 2 | Сигнал про введення даних для T1 – TP-1 | S1 |
| 3 | Чекати сигнал про завершення введення даних в T1 | W1 |
| 4 | Копіювати: Cp = C: | КС |
| 5 | Обрахувати: = +Cp\* |  |
| 6 | Сигнал про готовність для T1 - TP-1 | S2 |
| 7 | Чекати сигнал про готовність від T1 - TP-1 | W2 |
| 8 | Копіювати: ap=a; Kp=K; MKp=MK; bp=b | КС |
| 9 | Обрахувати: AH = ap\*Kp(\* M)-bp\* |  |
| 10 | Сигнал про завершення обчислень для Т1 | S3 |

## 2.3. Розробка схеми взаємодії процесів

На рисунку 2.2 зображено розроблену схему взаємодії процесів

T1

TI

InputT1

S1

W1

W1

signal\_K

signal\_K

S2

S2

W2

W2

S3

signal\_A

W3

lock\_a

InputTP

a

TP

lock\_K

S1

W1

K

lock\_MK

signal\_K

MK

S2

W2

lock\_C

C

lock\_b

b

S3

signal\_A

Рис. 2.2. Синхронізації та доступу до загальних ресурсів

a, K, C, MK, b – загальні ресурси

Та наступні функції:

**Copy\_a**–функція повертає копію змінної а

**Copy\_C** – функція повертає копію змінної C

**Copy\_MK** - функція повертає копію змінної MK

**Copy\_K** - функція повертає копію змінної K

**Copy\_b** - функція повертає копію змінної b

## 2.4. Розробка програми ПРГ1

На основі розроблених алгоритмів задач та схеми взаємодії задач розроблено паралельну програму для вирішення заданої математичної задачі за допомогою механізму захищених модулів. Програму розроблено на мові програмування C#. Для окремих видів потоків розроблено функції, які виконують дії в потоці, кожний з потоків зберігає номер, який задається при задані потоків, на основі якого визначається послідовність дій та параметри обчислень.

Програма складається з 2 класів:

**Functions**– клас вміщує в собі необхідні для обчислень функції

**Program**– головний клас програми, містить в собі основну функцію main(), з якої починається робота програми, та функції потоків і функції роботи з спільними ресурсами.

Лістинг до ПРГ1 наведено в додатку А.

## Тестування програми ПРГ1

Тестування було проведено на реальній 6-ядерній системі.

Результати проведених досліджень ефективності розробленої програми:

Таблиця 2.1 Час виконання обчислень програмою (в мілісекундах)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **N** | **T1** | **T2** | **T3** | **T4** | **T5** | **T6** |
| **900** | 22578 | 13796 | 9140 | 7079 | 5688 | 4485 |
| **1800** | 324094 | 168156 | 113453 | 83844 | 64250 | 53140 |
| **2400** | 789234 | 410140 | 277047 | 208422 | 152266 | 121703 |

Таблиця 2.2. Значення коефіцієнтів прискорення

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **N** | **Кількість процесорів (Р)** | | | | |
| **2** | **3** | **4** | **5** | **6** |
| **900** | 1,6 | 2,4 | 3,1 | 3,9 | 5,0 |
| **1800** | 1,9 | 2,8 | 3,8 | 5,0 | 6,0 |
| **2400** | 1,9 | 2,8 | 3,7 | 5,1 | 6,4 |

Таблиця 2.3. Значення коефіцієнтів ефективності

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **N** | **Кількість процесорів (Р)** | | | | |
| **2** | **3** | **4** | **5** | **6** |
| **900** | 80 | 80 | 77,5 | 78 | 83,3 |
| **1800** | 95 | 93,3 | 95 | 100 | 100 |
| **2400** | 95 | 93,3 | 92,5 | 102 | 106,6 |

На основі проведених досліджень побудовано наступні графіки залежностей коефіцієнтів прискорення та ефективності від кількості процесорів (рис. 2.3 та рис. 2.4):

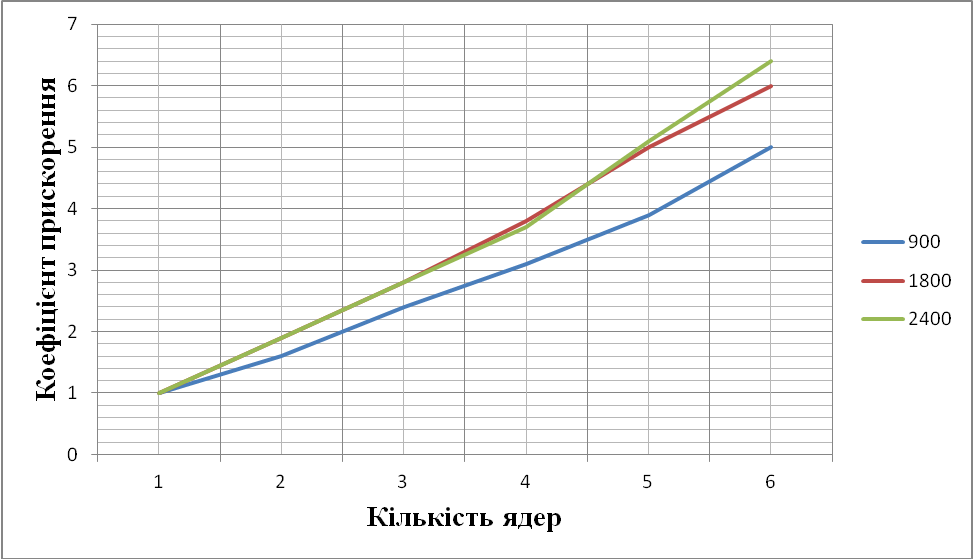


Рис. 2.3 Програма ПРГ1. Графік зміни коефіцієнту прискорення в

залежності від кількості ядер

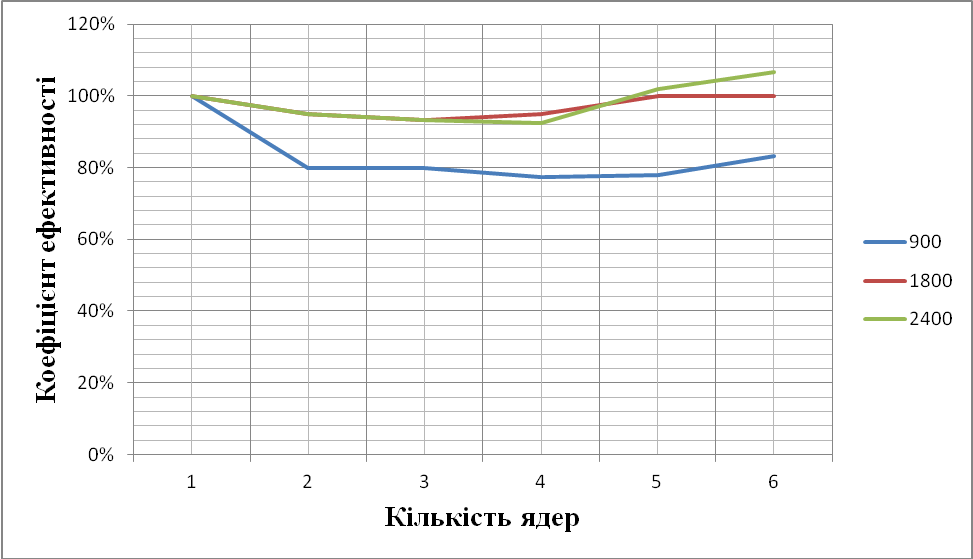


Рис. 2.4 Програма ПРГ1. Графік зміни коефіцієнту ефективності в

залежності від кількості ядер

## Висновки до розділу 2

1. Виконано розробку програми ПРГ1 для ПКС ЗП з використанням мови C#. Тестування програми показало, що використання багатоядерної ПКС та програми ПРГ1 забезпечує скорочення часу обчислення заданої математичної задачі. Значення Кп лежать в межах від 1,6 до 6,4. При цьому максимальне значення Кп забезпечує ПКС з Р= 6 та N =2400, а мінімальне мінімальне значення Кп при Р= 2 та N =900.
2. Значення коефіцієнтів прискорення та, відповідно, коефіцієнтів ефективності пов’язано з розміріністю данних, для яких відбувається обрахунок, оскільки при невеликих значеннях більше часу займає синхронізація потоків між собою та доступ до загальних ресурсів, ніж самі обрахунки.

Близьке до лінійного зростання кофефіціента прискорення вказує на те що є розкид для різних розмірностей елементів математичного виразу.

# РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС ЛП

## Розробка паралельного математичного алгоритму

**. . .**

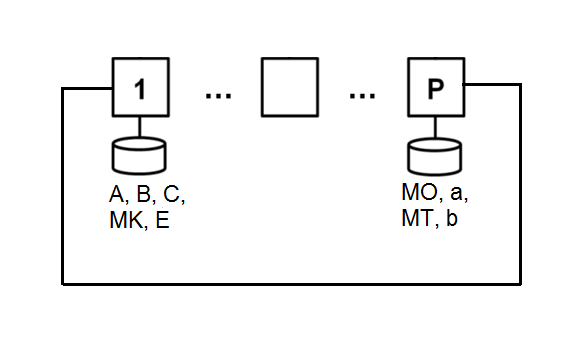


Рис. 3.1. Структура ПКС ЛП

Вихідна математична задача: A = a(B+C\*MO)(MK\*MT)-bE

Відповідно до концепції необмеженого паралелізму:

* Кожна операція виконується за одиницю часу
* Час передачі даних між процесорами не враховується
* Застосовується будь-яка потрібна (необмежена) кількість процесорів

Вихідна математична задача вміщує в собі окремі дії, які не можуть бути виконані паралельно, тобто задача не є повністю паралельною.

При розробці паралельного алгоритму передбачається, що розмірність задачі (N) більше або дорівнює числу процесорів (P) (N> = P), причому N кратне P, що дозволяє розбити задачу на окремі частини Н, кожен з яких обробляється своїм завданням. Якщо Nне кратне P, то останній процес буде обробляти частину даних, більшу ніж H.

H=N/P

Таким чином, отримано алгоритм виконання математичної задачі:

1. = +C\*



1. AH = a\*K(\* M)-b\*



## Розробка алгоритмів процесів

При розробці алгоритмів позначимо

T1 – перший процес

TP – останній процес

Tk – процеси після першого і до середнього включно

Tj – процеси після середнього і до останнього

i – номер даного процесу

**T1**

1. Введення B, C, MK, E
2. Передати в TP B(N-H\*P/2), C, MK, E(N-H\*P/2)
3. Отримати від ТP a, b, MT(H\*P/2), MO(H\*P/2)
4. Передати в T2 B(H\*(P/2-1)),C, MK, E(H\*(P/2-1)), a, b, MT(H\*(P/2-1)), MO(H\*(P/2-1))
5. Обрахувати = +C\*



1. Отримати від T2



1. Передати в TP



1. Отримати від TP



1. Передати в T2 K
2. Обрахувати: AH = a\*K(\* M)-b\*



1. Отримати від ТP: A(N-H\*P/2)
2. Отримати від Т2: A(H\*(P/2-1))
3. Вивести А

**Tk 1<k<=C**

1. Отримати від Т(і-1):B(H\*(P/2-i-1)),C,MK,E(H\*(P/2-i-1)),a,b,

MT(H\*(P/2-i-1)),MO(H\*(P/2-i-1))

1. Передати в Т(і+1): B(H\*(P/2-i)),C,MK,E(H\*(P/2-i)),a,b,

MT(H\*(P/2-i)),MO(H\*(P/2-i))

1. Обрахувати = +C\*



1. Отримати від Т(і+1)



1. Передати в Т(і-1):



1. Отримати від Т(і-1) K
2. Передати в Т(і+1): K
3. Обрахувати: AH = a\*K(\* M)-b\*



1. Отримати від Т(і+1)



1. Передати в Т(і-1)



**Tj C<j<P**

1. Отримати від Т(і+1):B(N-H\*(P/2-(P-i))),C,MK,E(N-H\*(P/2-(P-i))),a,b,

MT(N-H\*(P/2-(P-i))),MO(N-H\*(P/2-(P-i)))

1. Передати в Т(і-1): B(N-H\*(P/2-(P-i+1))),C,MK,E(N-H\*(P/2-(P-i+1))),a,b,

MT(N-H\*(P/2-(P-i+1))),MO(N-H\*(P/2-(P-i+1)))

1. Обрахувати = +C\*



1. Отримати від Т(і-1)



1. Передати в Т(і+1):



1. Отримати від Т(і+1): K
2. Передати в Т(і-1): K
3. Обрахувати: AH = a\*K(\* M)-b\*



1. Отримати від Т(і-1): A(N-H\*(P/2-(P-i+1)))
2. Передати в Т(і+1): A(N-H\*(P/2-(P-i)))

**TP**

1. Введення MO,a,MT,b
2. Передати в T1 MO(H\*P/2), a, b, MT(H\*P/2)
3. Отримати від T1 B(N-H\*P/2), C, MK, E(N-H\*P/2)
4. Обрахувати = +C\*



1. Отримати від Т(і-1):



1. Передати в T1



1. Отримати від T1



1. Передати в Т(і-1): K
2. Обрахувати: AH = a\*K(\* M)-b\*



1. Отримати від Т(і-1):



1. Передати в T1:



## Розробка схеми взаємодії процесів

На рисунку 3.2 зображено розроблену схему взаємодії процесів

Всього передбачено три типи процесів:

T1 – тип процесу для першої задачі, в ньому відбувається введення даних (B, C, MK, E) та виведення результату (A)

TP - тип процесу для останньої задачі, в ньому відбувається введення даних (MO,a,MT,b) та виведення результату (A)

T – тип процесу, який знаходиться між першим та останнім, реалізує необхідний функціонал по обчисленню та передачі даних

Входи:

**RecvET1** – вхід для передачі даних від процесу T1– B(N-H\*P/2), C, MK, E(N-H\*P/2)

**RightETP** – вхід для передачі даних від процесу TP– MO(H\*P/2), a, b, MT(H\*P/2)

**RightRecv** – вхід для передачі даних від «правого» процесу–

B(N-H\*(P/2-(P-i))),C,MK,E(N-H\*(P/2-(P-i))),a,b, MT(N-H\*(P/2-(P-i))),MO(N-H\*(P/2-(P-i)))

**LeftRecv** – вхід для передачі даних від «лівого» процесу –

B(H\*(P/2-i-1)),C,MK,E(H\*(P/2-i-1)),a,b,MT(H\*(P/2-i-1)),MO(H\*(P/2-i-1))

**RightRecvKH** – вхід для передачі даних від «правого» процесу-



**LeftRecvKH** – вхід для передачі даних від «лівого» процесу –



**RightRecvKP** – вхід для передачі даних від 1 процесу-



**LeftRecvK1** – вхід для передачі даних від P процесу –



**RightRecvK** – вхід для передачі даних від «правого» процесу– K

**LeftRecvK** – вхід для передачі даних від «лівого» процесу – K

**RightRecvAH** – вхід для передачі даних від «правого» процесу–



**LeftRecvAH** – вхід для передачі даних від «лівого» процесу – A(N-H\*(P/2-(P-i+1)))

**RightRecvATP** – вхід для передачі даних від P процесу – A(N-H\*P/2)

?

A(N-H\*P/2)

A(N-H\*(P/2-(P-i+1)))



LeftRRecvK1

RightRecvKP



B(H\*(P/2-i-1)),C,

MK,E(H\*(P/2-i-1)),a,b,

MT(H\*(P/2-i-1)),

MO(H\*(P/2-i-1))

B(H\*(P/2-i-1)),C,

MK,E(H\*(P/2-i-1))

,a,b,MT(H\*(P/2-i-1)),

MO(H\*(P/2-i-1))

B(N-H\*P/2), C, MK, E(N-H\*P/2)

MO(H\*P/2), a, b, MT(H\*P/2)

RightRecvATP

LeftRecvAH

RightRecvAH

LeftRecvK

RightRecvK

LeftRecvKH

RightRecvKH

RightRecv

LeftRecv

RecvET1

RecvETP

Tj

Ti

MO,a,MT,b

A,B,C,MK,E

TP

T1

# Рис. 3.2. Схема для синхронізації та доступу до ЗР

## Розробка програми ПРГ2

На основі розроблених алгоритмів задач та схеми взаємодії задач розроблено паралельну програму для вирішення заданої математичної задачі за допомогою механізму рандеву. Програму розроблено на мові програмування Ада. Для пересилання даних використовуються захищені входи. Є три типи потоків – Т1, Т та ТP. В залежності від номеру потоку для нього визначені входи, які пересилають або отримують дані. Всі потоки створюються динамічно, для чого були створені покажчики на потокові типи. Всі потоки, крім першого та останнього, записуються до масиву tasks, кожний потік отримує, як параметр, свій номер. Програма описана в модулі coursework. Також до складу проекту входить пакет Data, який містить в собі набір необхідних функцій для обчислення математичної задачі.

Лістинг до ПРГ2 наведено в додатку Б.

## Тестування програми ПРГ2

Тестування було проведено на реальній 6-ядерній системі.

Результати проведених досліджень ефективності розробленої програми:

Таблиця 3.1 Час виконання обчислень програмою (в мілісекундах)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **N** | **T1** | **T2** | **T3** | **T4** | **T5** | **T6** |
| **900** | 15,284 | 6,636 | 4,306 | 3,775 | 3,042 | 2,528 |
| **1800** | 166,535 | 77,002 | 46,582 | 37,533 | 28,720 | 23,103 |
| **2400** | 397,144 | 205,251 | 126,860 | 96,896 | 73,539 | 59,872 |

Таблиця 3.2. Значення коефіцієнтів прискорення

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **N** | **Кількість процесорів (Р)** | | | | |
| **2** | **3** | **4** | **5** | **6** |
| **900** | 2,3 | 3,5 | 4,0 | 5,0 | 6,0 |
| **1800** | 2,1 | 3,5 | 4,4 | 5,7 | 7,2 |
| **2400** | 1,9 | 3,1 | 4,0 | 5,4 | 6,6 |

Таблиця 3.3. Значення коефіцієнтів ефективності

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **N** | **Кількість процесорів (Р)** | | | | |
| **2** | **3** | **4** | **5** | **6** |
| **900** | 115 | 166,6 | 100 | 100 | 100 |
| **1800** | 105 | 116,6 | 110 | 114 | 120 |
| **2400** | 95 | 103,3 | 100 | 108 | 110 |

На основі проведених досліджень побудовано наступні графіки залежностей коефіцієнтів прискорення та ефективності від кількості процесорів (рис. 3.3 та рис. 3.4):

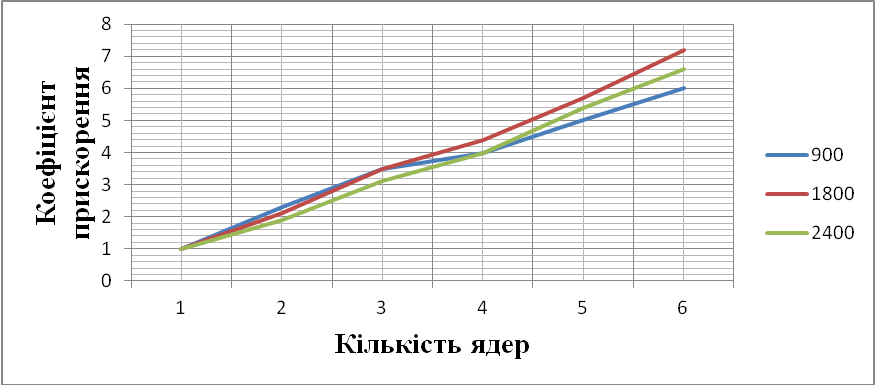


Рис. 3.3 Програма ПРГ2. Графік зміни коефіцієнту прискорення в

залежності від кількості ядер

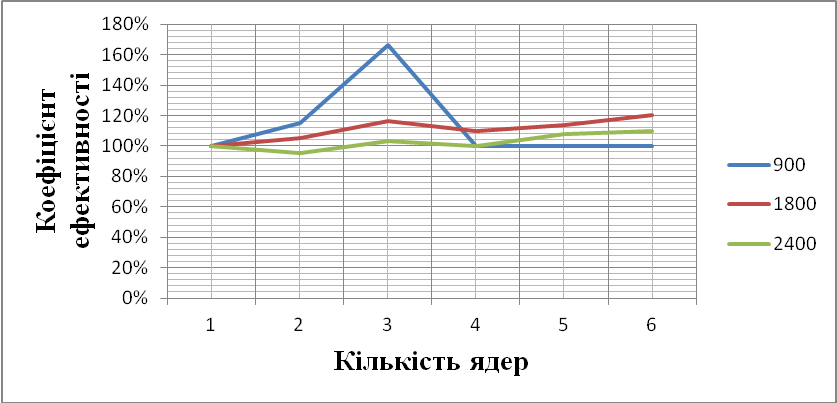


Рис. 3.4 Програма ПРГ2. Графік зміни коефіцієнту ефективності в

залежності від кількості ядер

## Висновки до розділу 3

1. Виконано розробку програми ПРГ2 для ПКС ЛП з використанням засобів синхронізації за методологією Рандеву в мові Ада. При тестуванні програми бачимо наступне: тестування програми показало, що використання багатоядерної ПКС та програми ПРГ1 забезпечує швидшення обчислення заданої математичної задачі. Значення Кп лежать в межах від 1,9 до 7.2. При цьому максимальне значення Кп забезпечує ПКС з Р= 6 та N =1800, а мінімальне значення Кп при Р= 2 та N =2400.
2. Дослідження показало, що коефіцієнт прискорення зростає практично лінійно, але для N=900, з підвищенням кількості процесорів ріст коефіцієнта прискорення різко збільшуеться на 3 ядрах та спадає, це пов’язується з більшою кількістю передач повідомлень між процесорами.
3. Коефіцієнт ефективності для N=900 досягає максимуму при P=3. Це пов’язано з тим, що для N=900, при збільшенні кількості процесорів час на передачу повідомлень збільшується більше, ніж зменшується час на обрахунок, а для N=1800 і N=2400 збільшується, зі збільшенням кількості процесорів.**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**
4. Ada.synchronous\_task\_control [Електронний ресурс] http://www.infeig.unige.ch/support/ada/gnatlb/a-sytaco.html
5. Semaphore (Java 2 PlatformSE 5.0) [Електронний ресурс] http://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/Semaphore.html
6. Semaphore and SemaphoreSlim [Електронний ресурс] http://msdn.microsoft.com/ru-ru/library/z6zx288a(v=vs.110).aspx
7. Semaphore Class (System.Threading) [Електронний ресурс] http://msdn.microsoft.com/enus/library/system.threading.semaphore(v=vs.110).aspx
8. SemaphoreFullException Class (System.Threading) [Електронний ресурс] http://msdn.microsoft.com/ru-ru/library/system.threading.semaphorefullexception(v=vs.110).aspx
9. SemaphoreSlim Class (System.Threading) [Електронний ресурс] http://msdn.microsoft.com/en-us/library/system.threading.semaphoreslim(v=vs.110).aspx
10. Адское программирование. [Електронний ресурс] http://www.ada-ru.org/V-0.4w/toc\_ru.html
11. І. А. Жуков, О. В. Корочкін Паралельні та розподілені обчислення – Київ: «Корнійчук» 2014. – 284 с.
12. Корочкин А.В. Ада 95: Введение в программирование. - Киев; Свит, 1998. - 260 с.

# ДОДАТКИ

## Додаток А.

## Лістинг до ПРГ1

Program

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

//Andriy Vitrichenko

namespace Lab\_kur\_s\_1

{

class Program

{

static DateTime start\_time;

static DateTime end\_time;

static int P = 8;

static int N = 8;

static int H = N / P;

static int a = 0;

static int b = 0;

static int[,] MO = null;

static int[,] MT = null;

static int[,] MK = null;

static int[] E;

static int[] A;

static int[] B;

static int[] K;

static int[] C;

static int[] temp1 = null;

static int[,] temp2 = null;

static int[] temp3 = null;

static int[] temp4 = null;

static int[] temp5 = null;

static object lock\_a = new object();

static object lock\_b = new object();

static object lock\_C = new object();

static object lock\_K = new object();

static object lock\_MK = new object();

static object mon\_signal\_count = new object();

static EventWaitHandle InputT1 = new EventWaitHandle(false, EventResetMode.ManualReset);

static EventWaitHandle InputTP = new EventWaitHandle(false, EventResetMode.ManualReset);

static EventWaitHandle[] signal\_K = null;

static EventWaitHandle[] signal\_A = null;

static Thread[] threads = null;

static void Main(string[] args)

{

bool success\_input = false;

while (!success\_input)

{

Console.Write("Input the number of threads: ");

string str\_count\_P = Console.ReadLine();

Console.Write("Input N: ");

string str\_count\_N = Console.ReadLine();

try

{

P = Convert.ToInt32(str\_count\_P);

N = Convert.ToInt32(str\_count\_N);

if (N < P && P>2)

{

throw new Exception("Count threads can`t be less for 'N'");

}

else

{

success\_input = true;

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

success\_input = false;

}

}

start\_time = DateTime.Now;

H = N / P;

signal\_K = new EventWaitHandle[P];

signal\_A = new EventWaitHandle[P];

threads = new Thread[P];

for (int i = 0; i < P;i++ )

{

threads[i] = new Thread(FuncI);

}

Functions funcs = new Functions(0,N,N);

temp1 = funcs.NewNullVector();

temp2 = funcs.NewNullMatrix();

temp3 = funcs.NewNullVector();

temp4 = funcs.NewNullVector();

temp5 = funcs.NewNullVector();

A = funcs.NewNullVector();

K = funcs.NewNullVector();

for (int i = 0; i < P;i++ )

{

signal\_K[i] = new EventWaitHandle(false, EventResetMode.ManualReset);

signal\_A[i] = new EventWaitHandle(false, EventResetMode.ManualReset);

threads[i].Name = i.ToString();

threads[i].Start();

}

Console.ReadLine();

}

static void FuncI()

{

//number of thread

int num\_thread = Convert.ToInt32(Thread.CurrentThread.Name);

//count last elements

int count\_last\_lm = H \* (num\_thread + 1);

if(num\_thread == P-1)

{

count\_last\_lm = N;

}

Functions funcs = new Functions(H \* num\_thread, count\_last\_lm, N);

Console.WriteLine("Start T{0}", num\_thread + 1);

if(num\_thread==0)

{

B = funcs.NewOneVector();

C = funcs.NewOneVector();

E = funcs.NewOneVector();

MK = funcs.NewOneMatrix();

InputT1.Set(); //S1

}

if (num\_thread==P-1)

{

a = 1;

b = 1;

MT = funcs.NewOneMatrix();

MO = funcs.NewOneMatrix();

InputTP.Set(); //S1

}

if(P>1)

{

WaitHandle[] handlesInput = null;

if(num\_thread != 0 && num\_thread != P-1)

{

handlesInput = new WaitHandle[2];

handlesInput[0] = InputT1;

handlesInput[1] = InputTP;

}

else

{

handlesInput = new WaitHandle[1];

if (num\_thread == P - 1)

{

handlesInput[0] = InputT1;

}

else

{

handlesInput[0] = InputTP;

}

}

WaitHandle.WaitAll(handlesInput); //W1

}

int[] Ci = Copy\_C(); //CS

funcs.Matrix\_Vector\_Mult(MO, Ci, ref temp1);

funcs.Vector\_Sum(temp1, B, ref K);

if(P>1)

{

signal\_K[num\_thread].Set(); //S2

WaitHandle.WaitAll(signal\_K); //W2

}

int[,] MKi = Copy\_MK();

funcs.Matrix\_Matrix\_Mult(MKi, MT, ref temp2);

int[] Ki = Copy\_K();

funcs.Matrix\_Vector\_Mult(temp2, Ki, ref temp3);

int ai = Copy\_a();

funcs.Vector\_Int\_Mult(temp3, ai, ref temp4);

int bi = Copy\_b();

funcs.Vector\_Int\_Mult(E, bi, ref temp5);

funcs.Vector\_Rizn(temp4, temp5, ref A);

if(P>1)

{

signal\_A[num\_thread].Set(); //S3

if(num\_thread==0)

{

WaitHandle.WaitAll(signal\_A); //W3

}

}

if(num\_thread==0)

{

funcs.Vector\_Output(A);

end\_time = DateTime.Now;

long count\_ms = (end\_time.Minute - start\_time.Minute) \* 60000 + (end\_time.Second - start\_time.Second) \* 1000 + end\_time.Millisecond -start\_time.Millisecond;

Console.WriteLine("count ms:{0}", count\_ms);

}

Console.WriteLine("Finish T{0}", num\_thread + 1);

Thread.CurrentThread.Abort();

}

#region Functions

public static int Copy\_a()

{

lock(lock\_a)

{

return a;

}

}

public static int Copy\_b()

{

lock (lock\_b)

{

return b;

}

}

public static int[,] Copy\_MK()

{

lock (lock\_MK)

{

return MK.Clone() as int[,];

}

}

public static int[] Copy\_K()

{

lock (lock\_K)

{

return K.Clone() as int[];

}

}

public static int[] Copy\_C()

{

lock (lock\_C)

{

return C.Clone() as int[];

}

}

#endregion

}

}

Functions

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Lab\_kur\_s\_1

{

public class Functions

{

int p\_From = 0;

int p\_To = 0;

int p\_N = 0;

public Functions(int m\_From, int m\_To,int m\_N)

{

p\_From = m\_From;

p\_N = m\_N;

p\_To = m\_To;

}

#region StartValues

public int[,] NewNullMatrix()

{

int[,] res = new int[p\_N,p\_N];

return res;

}

public int[,] NewOneMatrix()

{

int[,] res = new int[p\_N, p\_N];

for (int i = 0; i < p\_N;i++)

{

for (int j = 0; j < p\_N; j++)

{

res[i, j] = 1;

}

}

return res;

}

public int[] NewNullVector()

{

int[] res = new int[p\_N];

return res;

}

public int[] NewOneVector()

{

int[] res = new int[p\_N];

for (int i = 0; i < p\_N; i++)

{

res[i] = 1;

}

return res;

}

public int[,] NewRandMatrix()

{

int[,] res = new int[p\_N, p\_N];

Random rand = new Random();

for (int i = 0; i < p\_N; i++)

{

for (int j = 0; j < p\_N; j++)

{

res[i, j] = rand.Next(1,5);

}

}

return res;

}

#endregion

public void Matrix\_Matrix\_Mult(int[,] matrix\_h, int[,] matrix\_f, ref int[,] matrix\_out)

{

for (int i = p\_From; i < p\_To; i++)

{

for (int j = 0; j < p\_N; j++)

{

matrix\_out[i, j] = 0;

for (int k = 0; k < p\_N; k++)

{

matrix\_out[i, j] += matrix\_h[i, k] \* matrix\_f[k,j];

}

}

}

}

public void Matrix\_Int\_Mult(int[,] matrix\_h, int num, ref int[,] matrix\_out)

{

for (int i = p\_From; i < p\_To; i++)

{

for (int j = 0; j < p\_N; j++)

{

matrix\_out[i, j] = 0;

}

}

for (int i = p\_From; i < p\_To; i++)

{

for (int j = 0; j < p\_N; j++)

{

matrix\_out[i, j] = num \* matrix\_h[i,j];

}

}

}

public void Vector\_Int\_Mult(int[] vector\_h, int num, ref int[] vector\_out)

{

for (int i = p\_From; i < p\_To; i++)

{

vector\_out[i] = 0;

}

for (int i = p\_From; i < p\_To; i++)

{

vector\_out[i] = num \* vector\_h[i];

}

}

public void Matrix\_Matrix\_Sum(int[,] matrix\_h, int[,] matrix\_f, ref int[,] matrix\_out)

{

for (int i = p\_From; i < p\_To; i++)

{

for (int j = 0; j < p\_N; j++)

{

matrix\_out[i, j] = 0;

}

}

for (int i = p\_From; i < p\_To; i++)

{

for (int j = 0; j < p\_N; j++)

{

matrix\_out[i, j] = matrix\_f[i,j] + matrix\_h[i, j];

}

}

}

public void Matrix\_Vector\_Mult(int[,] matrix\_h, int[] vector\_f, ref int[] vector\_out)

{

for (int i = p\_From; i < p\_To; i++)

{

vector\_out[i] = 0;

}

for (int i = p\_From; i < p\_To; i++)

{

for (int j = 0; j < p\_N; j++)

{

vector\_out[i] += matrix\_h[i,j] \* vector\_f[j];

}

}

}

public void Vector\_Sum(int[] vector\_a, int[] vector\_b, ref int[] vector\_out)

{

for (int i = p\_From; i < p\_To; i++)

{

vector\_out[i] = vector\_a[i] + vector\_b[i];

}

}

public void Vector\_Rizn(int[] vector\_a, int[] vector\_b, ref int[] vector\_out)

{

for (int i = p\_From; i < p\_To; i++)

{

vector\_out[i] = vector\_a[i] - vector\_b[i];

}

}

public int Vector\_Max(int[] vector\_a)

{

int max = vector\_a[p\_From];

for (int i = p\_From; i < p\_To; i++)

{

if (max < vector\_a[i])

{

max = vector\_a[i];

}

}

return max;

}

public void Vector\_Sort(int[] vector\_a, ref int[] vector\_out)

{

bool isSorting = true;

while (isSorting)

{

isSorting = false;

for(int i=p\_From; i < p\_To; i++)

{

if(i+1 < p\_To)

{

if(vector\_a[i] > vector\_a[i+1])

{

int temp = vector\_a[i];

vector\_a[i] = vector\_a[i + 1];

vector\_a[i + 1] = temp;

isSorting = true;

}

}

}

}

for (int i = p\_From; i < p\_To; i++ )

{

vector\_out[i] = vector\_a[i];

}

}

public void Vector\_Sort(ref int[] vector\_out,int from, int to)

{

bool isSorting = true;

while (isSorting)

{

isSorting = false;

for (int i = from; i < to; i++)

{

if (i + 1 < p\_To)

{

if (vector\_out[i] > vector\_out[i + 1])

{

int temp = vector\_out[i];

vector\_out[i] = vector\_out[i + 1];

vector\_out[i + 1] = temp;

isSorting = true;

}

}

}

}

}

public void Vector\_Sort\_All(ref int[] vector\_out,int cur\_P, int m\_H)

{

Vector\_Sort(ref vector\_out,0,p\_To);

}

#region Output

public void Matrix\_Output(int[,] matrix)

{

if (p\_N < 10)

{

for (int i = 0; i < p\_N; i++)

{

string str = "";

for (int j = 0; j < p\_N; j++)

{

str += matrix[i, j] + " ";

}

Console.WriteLine(str);

}

}

}

public void Vector\_Output(int[] vector)

{

string str = "";

if(p\_N<10)

{

for (int i = 0; i < p\_N; i++)

{

str += vector[i] + " ";

}

}

Console.WriteLine(str);

}

#endregion

}

}

## Додаток B.

## Лістинг до ПРГ2

Program

------------------------------------------------

-- Kurs. Ada.Randevu

-- Task

--

-- 23.04.2015

-----------------------------------------------------

with Data;

with ada.text\_IO; use ada.text\_IO;

with ada.integer\_text\_IO; use ada.integer\_text\_IO;

with Ada.Synchronous\_Task\_Control; use Ada.Synchronous\_Task\_Control;

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

with Ada.Calendar; use Ada.Calendar;

with Ada.Float\_Text\_IO; use Ada.Float\_Text\_IO;

procedure Kursova is

N: Integer;

P: Integer;

H: Integer;

C: Integer;

last\_H: Integer;

procedure Start is

startTime, endTime: Time;

package Data\_Base is new Data(N); use Data\_Base;

----------------------------------------------------------------------

task type T1 (id: Integer) is

entry RecvETP(Ai: in Integer; MOi: in Matrix; Bi: in Integer; MTi: in Matrix);

entry RecvKH(KHi: in Vector);

entry RecvKHP(KHi: in Vector);

entry RecvAH(AH: in Vector);

entry RecvAHTP(AHTP: in Vector);

entry Start;

end T1;

task type TP (id: Integer) is

entry LRRecv(Bi: in Vector;Ci: in Vector;Ei: in Vector; MKi: in Matrix;Alphai: in Integer; MOi: in Matrix; Betai: in Integer; MTi: in Matrix);

entry RecvKH(KHi: in Vector);

entry RecvK(Ki: in Vector);

entry RecvAH(AH: in Vector);

entry Start;

end TP;

task type T (id: Integer) is

entry RecvET1(Bi: in Vector; Ci: in Vector; Ei: in Vector; MKi: in Matrix);

entry RecvKH(KHi: in Vector);

entry RecvKH1(KHi: in Vector);

entry RecvAH(AH: in Vector);

entry Start;

end T;

-- tasks array and ptr in task

type T\_Ptr is access T;

type T1\_Ptr is access T1;

type TP\_Ptr is access TP;

type T\_Ptr\_Array is array (1..P) of T\_Ptr;

tasks: T\_Ptr\_Array;

T\_1: T1\_Ptr;

T\_P: TP\_Ptr;

--------------------------------------------------------------------

task body T1 is

A:Vector(1..N);

B:Vector(1..N);

C:Vector(1..N);

E:Vector(1..N);

MT\_H:Matrix(1..N, 1..H);

MO\_H:Matrix(1..N, 1..H);

E\_h:Vector(1..H);

B\_h:Vector(1..H);

K\_h:Vector(1..H);

K:Vector(1..N);

alpha:integer;

Beta:integer;

MO:Matrix(1..N, 1..n);

MT:Matrix(1..N, 1..n);

MK:Matrix(1..N, 1..n);

K\_Send:Vector(1..H\*P/2);

MT\_Send:matrix(1..N, 1..H\*(P/2-1));

MO\_Send:matrix(1..N, 1..H\*(P/2-1));

E\_Send:Vector(1..H\*(P/2-1));

E\_SendP:Vector(1..n-H\*P/2);

B\_Send:Vector(1..H\*(P/2-1));

B\_SendP:Vector(1..n-H\*P/2);

begin

accept Start;

Put\_Line("T" & Integer'Image(id) & " start");

-- Input data

MK:=fillMatrix(1);

B:=fillVector(1);

C:=fillVector(1);

E:=fillVector(1);

if(p=1) then

MO:=fillMatrix(1);

MT:=fillMatrix(1);

Alpha:=1;

Beta:=1;

K := addVector(matrixMultiplicationVector(MO,C),B);

A := RiznMatrix(scalarMultiplicationOnVector(Alpha,matrixMultiplicationVector(matrixMultiplication(MK,MT),K)) ,scalarMultiplicationOnVector(Beta,E));

if (N<11) then

VectorOutput(A);

end if;

else

-- create data to count

E\_H := getVectorPart(E,1,H);

B\_H := getVectorPart(B,1,H);

-- create data to send

E\_Send :=getVectorPart(E, H+1,H\*(P/2-1));

E\_SendP :=getVectorPart(E, H\*P/2,n-H\*P/2);

B\_Send :=getVectorPart(B, H+1,H\*(P/2-1));

B\_SendP :=getVectorPart(B, H\*P/2,n-H\*P/2);

t\_P.RecvET1(B,C,E\_SendP,MK);

accept RecvETP(Ai: in Integer; MOi: in Matrix; Bi: in Integer; MTi: in Matrix) do

Alpha := ai;

Beta := Bi;

MO\_H := GetMatrixPart(MOi,1,h);

MT\_H := GetMatrixPart(MTi,1,h);

MO\_Send := GetMatrixPart(MOi,H+1,H\*(P/2-1));

MT\_Send := GetMatrixPart(MTi,H+1,H\*(P/2-1));

end;

t\_P.RecvET1(B\_SendP; C; E\_SendP; MK);

if (P=2) then

else

-- Send data to next task

tasks(2).RecvET1(B\_Send; C; E\_Send; MK);

end if;

-- get data from next

K\_Send(1..H) := addVector(matrixMultiplicationVector(MO\_H,C),B\_H);

K(1..H):=K\_Send(1..H);

accept RecvKH(KHi: in Vector) do

K\_Send(H+1..H\*P/2) := KHi;

K(H+1..H\*P/2):=KHi;

end;

t\_P.RecvKH1(K\_Send);

accept RecvKHP(KHi: in Vector) do

K(H\*P/2+1..N) := KHi;

end;

tasks(2).RecvK(K);

end if;

-- count A\_H = B\_H + ai\*E\*(MT\*MR\_H)

A(1..H) := RiznMatrix(scalarMultiplicationOnVector(Alpha,matrixMultiplicationVector(matrixMultiplication(MK,MT\_H),K)) ,scalarMultiplicationOnVector(Beta,E\_H));

-- get result from next in A

accept RecvAH(AH: in Vector) do

A(H+1..H\*P/2) := AH;

end ResultRecv;

accept RecvAHTP(AHTP: in Vector) do

A(H\*P/2+1..N) := AHTP;

end ResultRecv;

-- OUTPUT RESULT

if (N<11) then

VectorOutput(A);

end if;

end if;

-- output time

endTime := clock;

Put("Time (s): ");

Ada.Float\_Text\_IO.Put(Float(endTime - startTime), Fore => 10, Aft => 10, exp =>0);

Put\_Line(" ");

Put\_Line("T" & Integer'Image(id) & " end");

end T1;

----------------------------------------------------------------------

task body T is

A\_H:Vector(1..N);

B:Vector(1..N);

C:Vector(1..N);

E:Vector(1..N);

MT\_H:Matrix(1..N, 1..H);

MO\_H:Matrix(1..N, 1..H);

E\_h:Vector(1..H);

B\_h:Vector(1..H);

K\_h:Vector(1..H);

K:Vector(1..N);

alpha:integer;

Beta:integer;

MO:Matrix(1..N, 1..n);

MT:Matrix(1..N, 1..n);

MK:Matrix(1..N, 1..n);

K\_Send:Vector(1..H\*P/2);

MT\_Send:matrix(1..N, 1..H\*(P/2-1));

MO\_Send:matrix(1..N, 1..H\*(P/2-1));

E\_Send:Vector(1..H\*(P/2-1));

E\_SendP:Vector(1..n-H\*P/2);

B\_Send:Vector(1..H\*(P/2-1));

B\_SendP:Vector(1..n-H\*P/2);

begin

accept Start;

Put\_Line("T" & Integer'Image(id) & " start");

accept RecvET1(Bi: in Vector; Ci: in Vector; Ei: in Vector; MKi: in Matrix;Alphai : in integer;Betai :in integer;MTi:in Matrix;MOi:in Matrix) do

C:=Ci;

MK := MKi;

B\_H := GetMatrixPart(B,H\*P/2+1,H);

E\_H := GetMatrixPart(E,H\*P/2+1,H);

B\_Send := GetMatrixPart(B,H\*P/2+H+1,N-H\*(P/2+1));

E\_Send := GetMatrixPart(E,H\*P/2+H+1,N-H\*(P/2+1));

Alpha := Alphai;

Beta := Betai;

MO\_H := GetMatrixPart(MOi,1,h);

MT\_H := GetMatrixPart(MTi,1,h);

MO\_Send := GetMatrixPart(MOi,H+1,H\*(P/2-1));

MT\_Send := GetMatrixPart(MTi,H+1,H\*(P/2-1));

end;

t\_1.RightRecv(Alpha, MK\_Send, MT\_Send);

tasks(Id-1).RecvET1(B\_Send, C, E\_Send, MK,Alpha,Beta,MT\_Send,MO\_Send);

else

tasks(Id+1).RecvET1(B\_Send, C, E\_Send, MK,Alpha,Beta,MT\_Send,MO\_Send);

end if;

K\_Send(1..H) := addVector(matrixMultiplicationVector(MO\_H,C),B\_H);

K(1..H):=K\_Send(1..H);

accept RecvKH(KHi: in Vector) do

K\_Send(H+1..H\*P/2) := KHi;

K(H+1..H\*P/2):=KHi;

end;

accept RecvKHP(KHi: in Vector) do

K(H\*P/2+1..N) := KHi;

end;

if id<C then

tasks(Id+1).RecvK(K);

else

tasks(Id-1).RecvK(K);

end if;

end if;

A\_Send(1..H) := RiznMatrix(scalarMultiplicationOnVector(Alpha,matrixMultiplicationVector(matrixMultiplication(MK,MT\_H),K)) ,scalarMultiplicationOnVector(Beta,E\_H));

accept RecvAH(AH: in Vector) do

A\_Send(1..H\*(P/2-1)) := AH;

end ResultRecv;

if id<C then

tasks(Id+1).RecvAHTP(A\_Send);

else

tasks(Id-1).RecvAHTP(A\_Send);

end if;

Put\_Line("T" & Integer'Image(id) & " end");

end T;

----------------------------------------------------------------

task body TP is

A\_H:Vector(1..N);

A\_Send:Vector(1..N-H\*P/2);

B:Vector(1..N);

C:Vector(1..N);

E:Vector(1..N);

MT\_H:Matrix(1..N, 1..H);

MO\_H:Matrix(1..N, 1..H);

E\_h:Vector(1..H);

B\_h:Vector(1..H);

K\_h:Vector(1..H);

K:Vector(1..N);

alpha:integer;

Beta:integer;

MO:Matrix(1..N, 1..n);

MT:Matrix(1..N, 1..n);

MK:Matrix(1..N, 1..n);

K\_Send:Vector(1..H\*P/2);

MT\_Send:matrix(1..N, 1..H\*(P/2-1));

MO\_Send:matrix(1..N, 1..H\*(P/2-1));

E\_Send:Vector(1..N-H\*(P/2+1));

E\_SendP:Vector(1..H\*P/2);

B\_Send:Vector(1..N-H\*(P/2+1));

B\_SendP:Vector(1..H\*P/2);

begin

accept Start;

Put\_Line("T" & Integer'Image(id) & " start");

-- Send data to prev task

MT:=fillMatrix(1);

MO:=fillMatrix(1);

Alpha := 1;

Beta := 1;

MO\_Send := GetMatrixPart(MO,1,H\*(P/2-1));

MT\_Send := GetMatrixPart(MT,1,H\*(P/2-1));

t\_1.RecvETP(Alpha; MO\_Send; Beta; MT\_Send);

accept RecvET1(Bi: in Vector; Ci: in Vector; Ei: in Vector; MKi: in Matrix) do

C:=Ci;

MK := MKi;

B\_H := GetMatrixPart(B,H\*P/2+1,H);

E\_H := GetMatrixPart(E,H\*P/2+1,H);

B\_Send := GetMatrixPart(B,H\*P/2+H+1,N-H\*(P/2+1));

E\_Send := GetMatrixPart(E,H\*P/2+H+1,N-H\*(P/2+1));

end;

if(P=2) then

-- get data from prev

accept LeftRecv(Ei: in Vector; MOi: in Matrix) do

-- create data to count

E\_H := getVectorPart(Ei,1,H);

MO := MOi;

-- create data to send

end;

t\_1.RightRecv(Alpha, MK\_Send, MT\_Send);

else

tasks(2).RecvET1(B\_Send; C; E\_Send; MK);

end if;

K\_Send(1..H) := addVector(matrixMultiplicationVector(MO\_H,C),B\_H);

K(1..H):=K\_Send(1..H);

accept RecvKH(KHi: in Vector) do

K\_Send(H+1..H\*P/2) := KHi;

K(H+1..H\*P/2):=KHi;

end;

t\_1.RecvKHP(K\_Send);

accept RecvKHP(KHi: in Vector) do

K(H\*P/2+1..N) := KHi;

end;

tasks(2).RecvK(K);

end if;

A\_Send(1..H) := RiznMatrix(scalarMultiplicationOnVector(Alpha,matrixMultiplicationVector(matrixMultiplication(MK,MT\_H),K)) ,scalarMultiplicationOnVector(Beta,E\_H));

accept RecvAH(AH: in Vector) do

A\_Send(1..H\*(P/2-1)) := AH;

end ResultRecv;

t\_1.RecvAHTP(A\_Send);

Put\_Line("T" & Integer'Image(id) & " end");

end TP;

----------------------------------------------------------------

----------------------------------------------------------------

begin

-- Create tasks

startTime := clock;

for i in 2..(P-1) loop

tasks(i) :=new T(id => i);

end loop;

T\_1 := new T1(1);

if(p>1) then

T\_P := new TP(P);

end if;

-- Start tasks

T\_1.Start;

if(p>1) then

T\_P.Start;

end if;

for i in 2..(P-1) loop

tasks(i).Start;

end loop;

end Start;

begin

Put\_Line("Input N:");

Ada.Integer\_Text\_IO.Get(N);

Put\_Line("Input P:");

Ada.Integer\_Text\_IO.Get(P);

--if (P>2) then

H := N/P;

C :=P/2+1;

last\_H := N-(P-1)\*H;

Start;

--else

-- Put\_Line("Error! P must be more than 2.");

--end if;

end Kursova;

Data

generic

N: Positive := 3; -- size of vector and matrix

package Data is

type Vector is array (Positive range <>) of Integer;

-- vector type

type Matrix is array (Positive range <>, Positive range <>) of Integer;

-- matrix type

procedure vectorOutput(v: in Vector); -- output vector in console

procedure matrixOutput(m: in Matrix); -- output matrix in console

function fillMatrix(number: in Integer) return Matrix; -- fill matrix by one number

function fillVector(number: in Integer) return Vector; -- fill vector by one number

function getMatrixPart(MM: in Matrix; start: in Integer; -- get part from matrix

size: in Integer) return Matrix;

function getVectorPart(V: in Vector; start: in Integer; -- get part of vector

size: in Integer) return Vector;

function getMaxVectorElement(v: in Vector) return Integer; -- find max element in vector

function matrixMultiplication(m1 : in Matrix; m2 : in Matrix) return Matrix;

function matrixMultiplicationVector(m1 : in Matrix; V2 : in Vector ) return vector;

-- multiplication matrix on matrix

function matrixMultiplicationOnInteger(m1 : in Matrix; m :in Integer ) return Matrix;

function vectorMultiplicationOnMatrix(v: in Vector; m: in Matrix)

-- multiplication vector on matrix

return Vector;

function scalarMultiplicationOnVector(s: in Integer; v: in Vector) -- multiplication scalar on matrix

return Vector;

function addMatrix(v1: in Matrix; v2: in Matrix) return Matrix;

function addVector(v1: in Vector; v2: in Vector) return Vector;

function RiznVector(v1: in Vector; v2: in Vector) return Vector;

-- add two vectors

function concatMatrix(V1: Matrix; V2: Matrix) return Matrix;

-- concatenation two vectors in one

function max(a1:in Integer; a2: in Integer) return Integer;

-- count max of two numbers

end Data;

with ada.text\_IO; use ada.text\_iO;

WITH Ada.integer\_Text\_IO;

with Ada.Integer\_Text\_IO;

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

package body Data is

-- output vector in console

procedure vectorOutput(v: in Vector) is

begin

for i in 1..v'Length loop

Ada.Integer\_Text\_IO.Put(v(i));

Ada.Text\_IO.Put(" ");

end loop;

Ada.Text\_IO.Put\_Line(" ");

end vectorOutput;

-- output matrix in console

procedure matrixOutput(m: in Matrix) is

begin

for i in 1..m'Length(1) loop

for j in 1..m'Length(2) loop

Ada.Integer\_Text\_IO.Put(m(i,j));

Ada.Text\_IO.Put(" ");

end loop;

Ada.Text\_IO.Put\_Line("");

end loop;

end matrixOutput;

-- fill matrix by one number

function fillMatrix(number: in Integer) return Matrix is

M : Matrix(1..N,1..N);

begin

for i in 1..M'Length(1) loop

for j in 1..M'Length(2) loop

M(i,j):=number;

end loop;

end loop;

return M;

end fillMatrix;

-- fill vector by one number

function fillVector(number: in Integer) return Vector is

V : Vector(1..N);

begin

for i in 1..V'Length loop

V(i):=number;

end loop;

return V;

end fillVector;

function getMatrixPart(MM: in Matrix; start: in Integer;

size: in Integer) return Matrix is

MR: Matrix(1..N,1..size);

begin

for i in 1..MM'Length(1) loop

for j in start..(start+size-1) loop

MR(i,j-start+1):=MM(i,j);

end loop;

end loop;

return MR;

end;

-- get part of vector

function getVectorPart(V: in Vector; start: in Integer;

size: in Integer) return Vector is

R: Vector(1..size);

begin

for i in 1..size loop

R(i):=V((i+start)-1);

end loop;

-- Put\_Line(Integer'Image(size));

return R;

end;

-- find max element in vector

function getMaxVectorElement(v: in Vector) return Integer is

max : Integer;

begin

if (V'Length > 0 ) then

max := v(1);

for i in 1..v'Length loop

if v(i)>max then

max := v(i);

end if;

end loop;

else

max:= -1\*(2 \*\* 31 - 1);

end if;

return max;

end getMaxVectorElement;

-- multiplication matrix on matrix

function matrixMultiplication(m1 : in Matrix; m2 :

in Matrix ) return Matrix is

MR :Matrix(1..m1'Length(1),1..m2'Length(2));

begin

for i in 1..m1'Length(1) loop

for j in 1..m2'Length(2) loop

MR(i, j) := 0 ;

for k in 1..m1'Length loop

MR(i, j) := MR(i, j) + (m1(i, k)\*m2(k, j));

end loop;

end loop;

end loop;

return MR;

end matrixMultiplication;

-- multiplication matrix on vector

function matrixMultiplicationVector(m1 : in Matrix; V2 :

in Vector ) return vector is

R :Vectror(1..V2'Length);

begin

for i in 1..m1'Length(1) loop

R(i):=0;

for j in 1..V2'Length loop

R(j) := R(j) + (m1(i, j)\*V2(j));

end loop;

end loop;

return R;

end matrixMultiplicationVector;

-- multiplication matrix on matrix

function matrixMultiplicationOnInteger(m1 : in Matrix; m :

in Integer ) return Matrix is

MR :Matrix(1..m1'Length(1),1..m1'Length(2));

begin

for i in 1..m1'Length(1) loop

for j in 1..m1'Length(2) loop

MR(i, j) := 0 ;

MR(i, j) := m1(i, j)\*m;

end loop;

end loop;

return MR;

end matrixMultiplicationOnInteger;

-- multiplication vector on matrix

function vectorMultiplicationOnMatrix(v: in Vector;

m: in Matrix) return Vector is

R : Vector(1..m'Length(2));

begin

for i in 1..m'Length(2) loop

R(i) := 0 ;

for k in 1..N loop

R(i) := R(i) + v(k)\*m(k, i);

end loop;

end loop;

return R;

end vectorMultiplicationOnMatrix;

-- multiplication scalar on matrix

function scalarMultiplicationOnVector(s: in Integer;

v: in Vector) return Vector is

R: Vector(1..v'Length);

begin

for i in 1..v'Length loop

R(i) := v(i)\*s;

end loop;

return R;

end scalarMultiplicationOnVector;

-- add two vectors

function addVector(v1: in Vector; v2: in Vector) return Vector is

R: Vector(1..v2'Length);

begin

for i in 1..v2'Length(1) loop

R(i):=v1(i)+v2(i);

end loop;

return R;

end addVector;

-- add two vectors

function RiznVector(v1: in Vector; v2: in Vector) return Vector is

R: Vector(1..v2'Length);

begin

for i in 1..v2'Length(1) loop

R(i):=v1(i)-v2(i);

end loop;

return R;

end RiznVector;

-- add two vectors

function addMatrix(v1: in Matrix; v2: in Matrix) return Matrix is

R: Matrix(1..v2'Length(1), 1..v2'Length(2));

begin

for i in 1..v2'Length(1) loop

for j in 1..v2'Length(2) loop

R(i,j):=v1(i,j)+v2(i,j);

end loop;

end loop;

return R;

end addMatrix;

-- concatenation two vectors in one

function concatMatrix(V1: Matrix; V2: Matrix) return Matrix is

R: Matrix(1..n, 1..V1'Length(2)+V2'Length(2));

begin

for i in 1..V1'Length(1) loop

for j in 1..V1'Length(2) loop

R(i, j):=V1(i, j);

end loop;

end loop;

for i in 1..V1'Length(1) loop

for j in 1..V2'Length(2) loop

R(i, j+V1'Length(2)):=V2(i, j);

end loop;

end loop;

return R;

end concatMatrix;

-- count max of two numbers

function max(a1:in Integer; a2: in Integer) return Integer is

max: Integer;

begin

if(a1>a2) then

max:=a1;

else

max:= a2;

end if;

return max;

end max;

end Data;