НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кафедра обчислювальної техніки\_\_\_\_\_\_\_\_\_\_\_\_

(повна назва кафедри, циклової комісії)

**РОЗРАХУНКОВО - ГРАФІЧНА РОБОТА**

з дисципліни «Паралельне програмування»

(назва дисципліни)

на тему: «Розробка програмного забезпечення

для паралельних комп’ютерних систем»

Студента (ки) 3 курсу ІП-54 групи спеціальності 121 «Програмна інженерія»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Макаренко А.О.\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник доцент Корочкін О.В.

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

Кількість балів: \_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_\_\_\_

Київ – 2018 рік

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) інформатики та обчислювальної техніки

( повна назва )

Кафедра обчислювальної техніки

( повна назва )

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 121 «Програмна інженерія»

(шифр і назва)

***З А В Д А Н Н Я***

НА РГР СТУДЕНТУ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Макаренко Антону Олександровичу\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, імя, по батькові)

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 21 травня 2018 р.

3. Вхідні дані до роботи

- математична задача: *a = mах(MB\*MC+ ММ)*

- структури ПКС CП та ПКС ЛП

- мови і бібліотеки програмування: *C#, Ada*

- засоби організації взаємодії процесів: ПКС СП – Семафори, мютекси, події, критичні секції, ПКС ЛП – механізм рандеву мови *Ada*

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

- розробка і тестування програми ПРГ1 для ПКС СП

- розробка і тестування програми ПРГ2 для 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 \_\_12.04.2018\_\_

***КАЛЕНДАРНИЙ ПЛАН***

|  |  |  |
| --- | --- | --- |
| №  з/п | Назва етапів виконання РГР | Строк виконання етапів РГР |
| 1 | Виконання Розділу 1 | 01.05.2018 |
| 2 | Виконання Розділу 2 | 10.05.2018 |
| 5 | Оформлення РГР | 20.05.2018 |
| 6 | Перевірка РГР викладачем | 21.05.2018 |

|  |  |  |
| --- | --- | --- |
| **Студент** | **\_\_\_\_\_\_\_\_\_\_\_\_** | **\_\_\_\_\_\_Макаренко А.О.\_\_** |
|  | ( підпис ) | (прізвище та ініціали) |
| **Керівник роботи** | **\_\_\_\_\_\_\_\_\_\_\_\_\_** | **\_\_\_\_\_\_Корочкін О.В.\_\_\_\_** |
|  | ( підпис ) | (прізвище та ініціали) |

ЗМІСТ

[ВСТУП 5](#_Toc513578690)

[РОЗДІЛ 1. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС СП 7](#_Toc513578691)

[1.1. Розробка паралельного математичного алгоритму 7](#_Toc513578692)

[1.2. Розробка алгоритмів процесів 8](#_Toc513578693)

[1.3. Розробка схеми взаємодії процесів 10](#_Toc513578694)

[1.4. Розробка програми ПРГ1 11](#_Toc513578695)

[1.5. Тестування програми ПРГ1 12](#_Toc513578696)

[1.6. Висновки до розділу 1 15](#_Toc513578697)

[РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС ЛП 17](#_Toc513578698)

[2.1. Розробка паралельного математичного алгоритму 17](#_Toc513578699)

[2.2. Розробка алгоритмів процесів 18](#_Toc513578700)

[2.3. Розробка схеми взаємодії процесів 20](#_Toc513578701)

[2.4. Розробка програми ПРГ2 20](#_Toc513578702)

[2.5. Тестування програми ПРГ2 21](#_Toc513578703)

[2.6. Висновки до розділу 2 24](#_Toc513578704)

[ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ ДО РОБОТИ 26](#_Toc513578705)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 27](#_Toc513578706)

[ДОДАТКИ 28](#_Toc513578707)

[Додаток А 28](#_Toc513578708)

[Додаток Б 29](#_Toc513578709)

[Додаток В 30](#_Toc513578710)

[Додаток Г 32](#_Toc513578711)

[Додаток Д 37](#_Toc513578712)

[Додаток Е 38](#_Toc513578713)

[Додаток Ж 39](#_Toc513578714)

[Додаток И 41](#_Toc513578715)

# ВСТУП

Програмування для паралельних і розподілених комп’ютерних систем стало невід’ємною складовою загального мистецтва програмування. Сьогодні неможливо уявити мову програмування, яка не мала б механізмів роботи з процесами (*Ada*, *Java*). Засоби роботи з процесами реалізовано також у вигляді спеціальних бібліотек (*PVM*, *MPI*) або є частиною операційної системи (*Pthreads*, *Win32*). Крім програмування для багатопроцесорних та розподілених комп’ютерних системи, процеси використовують для розроблення програм для систем реального часу[1].

Паралельна програма містить в собі деяку кількість процесів, які працюють разом для виконання деякої задачі. Кожний процес – це послідовна програма, а точніше – послідовність операторів, які виконуються один за одним. Послідовна програма має один потік виконання, а паралельна – декілька.

Спільна робота процесів паралельної програми здійснюється за допомогою їх взаємодії. Взаємодія програмується за допомогою використання розподілених змінних або пересилання повідомлень.

При будь-якому способі взаємодії процесам необхідна взаємна синхронізація. Існує два основні види синхронізації – взаємне виключення та умовна синхронізація[2].

Уміння розробити паралельний алгоритм задачі, описати алгоритми паралельних процесів, організувати взаємодію процесів, вирішити завдання взаємного виключення та синхронізації процесів, передати дані між процесами, створити та налагодити паралельну або розподілену програму, розмістити та виконати її в реальній паралельній комп’ютерній системі – актуальне завдання для сучасного програміста.

Як і інші прикладні області комп’ютерних наук, паралельне програмування пройшло декілька стадій. Воно виникло завдяки новим можливостям, які надав розвиток апаратного забезпечення, та розвинулось згідно з технологічними змінами. Через деякий час спеціалізовані методи були об’єднані в набір основних принципів та загальних методів програмування.

Ефективність розв’язання задач шляхом паралельного програмування залежить не лише від потужності машини, наявних технічних ресурсів, а й від навиків програміста, його вміння скласти правильний та ефективний паралельний алгоритм.

В даній роботі будуть розроблені алгоритми обчислення заданого математичного виразу в паралельній комп’ютерній системі зі спільною та локальною пам’яттю.

В розділі 1 розробляється програма ПРГ1 для паралельної комп’ютерної системи зі спільною пам’яттю з чотирма процесорами. Проводиться тестування розробленої програми для визначення коефіцієнтів прискорення та ефективності.

В розділі 2 розробляється програма ПРГ2 для паралельної комп’ютерної системи з локальною пам’яттю з чотирма процесорами, що має кільцеву структуру, визначаються коефіцієнти прискорення та ефективності.

# РОЗДІЛ 1. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС СП

Математична задача:

Мова програмування: *C#.*

Засоби взаємодії процесів: Семафори, мютекси, події, критичні секції.

Структура паралельної комп’ютерної системи зі спільною пам’яттю (ПКС СП) представлена на рис 1.1.

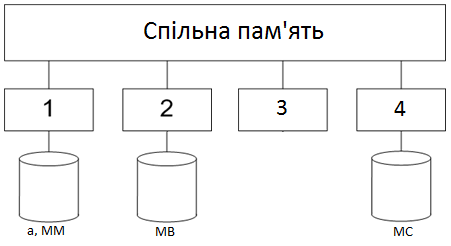


Рис.1.1. Структура ПКС зі спільною пам’яттю

## 1.1. Розробка паралельного математичного алгоритму

Паралельний математичний алгоритм відповідно до рекомендованої методики [1] подано у табл. 1.1.

Таблиця 1.1. Паралельний алгоритм задачі

|  |  |
| --- | --- |
| № кроку | Дія |
| 1 |  |

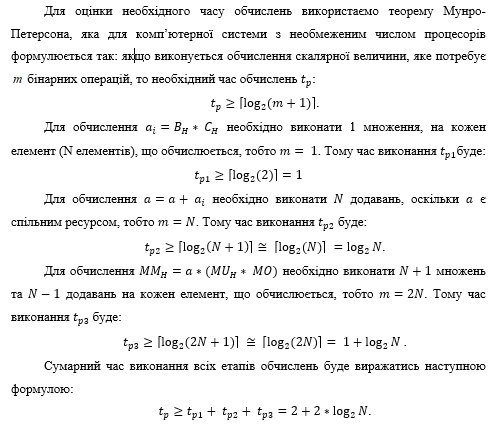
де:

* *N* – розмірність матриць;
* *P* – кількість процесорів;
* ;
* *MBH* — *H* рядкiв матриці *MB*;
* *MMH* — *H* рядкiв матрицi *MM*;
* *a* – результат – доступ до *a* організований через критичну секцію

Спiльнi ресурси: *MC*.

Для оцінки необхідного часу обчислень використаємо **теорему Мунро-Петерсона**, яка для комп’ютерної системи з необмеженим числом процесорів формулюється так: якщо виконується обчислення скалярної величини, яка потребує m бінарних операцій, то необхідний час обчислень *tp*:

Для обчислення необхідно виконати 1 множення на кожен елемент (N елементів), що обчислюється, тобто . Тому час виконання буде:



де – найменше ціле число, більше, або таке, що дорівнює *x*, логарифм береться за основою 2 [1].

1. При виконанні операції множення матриць розмірності , для отримання кожного елементу результуючої матриці, необхідно виконати операцій множення та операцій додавання.

Отже, , .

1. Операція віднімання двох матриць потребує одного такту .

Отже, сумарний час для обчислення кожного елементу результуючої матриці, при виконанні операції: :

## 1.2. Розробка алгоритмів процесів

Алгоритм задачі

1. Введення
2. Чекати сигнал від .
3. Сигнал задачы про введення ;
4. Чекати на введення даних у задачах ;
5. Копіювати: ; (КД1)
6. Обчислення:
7. Сигнал задачам про завершення обчислення ;
8. Чекати на завершення обчислення у задачах ;
9. Копіювати: ; (КД2)
10. Обчислення:
11. Чекати на завершення обчислень в задачах ;
12. Виведення результату

Алгоритм задачі

1. Введення
2. Сигнал задачам про введення ;
3. Чекати на введення даних у задачах ;
4. Копіювати: ; (КД1)
5. Обчислення:
6. Сигнал задачам про завершення обчислення ;
7. Чекати на завершення обчислення у задачах ;
8. Копіювати: ; (КД2)
9. Обчислення:
10. Сигнал задачі про завершення обчислень;

Алгоритм задачі

1. Чекати на введення даних у задачах ;
2. Копіювати: ; (КД1)
3. Обчислення:
4. Сигнал задачам про завершення обчислення ;
5. Чекати на завершення обчислення у задачах ;
6. Копіювати: ; (КД2)
7. Обчислення:
8. Сигнал задачі про завершення обчислень;

Алгоритм задачі

1. Введення
2. Сигнал задачам про введення ;
3. Чекати на введення даних у задачах ;
4. Копіювати: ; (КД1)
5. Обчислення:
6. Сигнал задачам про завершення обчислення ;
7. Чекати на завершення обчислення у задачах ;
8. Копіювати: ; (КД2)
9. Обчислення:
10. Сигнал задачі про завершення обчислень;

## 1.3. Розробка схеми взаємодії процесів

На основі алгоритму процесів було розроблено структурну схему взаємодії процесів.

Задача синхронізації та задача взаємного виключення вирішується за допомогою механізму захищених модулів.

Для вирішення задачі синхронізації використовується захищений модуль *Synchro*, що містить:

* захищений вхід *WaitInput* та захищену процедуру *SignalInput* для синхронізації по вводу даних;
* захищений вхід *WaitCountMK* та захищену процедуру *SignalCountMK* для синхронізації по завершенню обчислення матриці *MK*;
* захищений вхід *WaitEndCount* та захищену процедуру *SignalEndCount* для синхронізації по завершенню обчислення.

Для вирішення задачі взаємного виключення використовується захищений модуль *Resources*, що містить:

* захищену функцію *CopyME* та захищену процедуру *SetME*, що контролюють доступ до спільного ресурсу *ME*;
* захищену функцію *CopyMK* та захищену процедуру *SetMK*, що контролюють доступ до спільного ресурсу *MK*.

Схема взаємодії процесів наведена у додатку А.

## 1.4. Розробка програми ПРГ1

Програма ПРГ1, згідно з технічним завданням, розроблена на мові програмування *Ada*. Для взаємодії процесів використовується механізм захищених модулів.

Програма складається з файлу *prg1.abd*, що містить в собі:

* Процедуру *PRG1* – точка входу в програму, запускає процедуру, яка запускає задачі, вимірює час виконання програми ПРГ1;
* Визначення констант *N, P, H*;
* Специфікації та тіла захищених модулів *Resources* та *Synchro*;
* Процедуру *StartTasks*, яка містить специфікації та тіла задач *T1, T2, T3, T4*, запускає ці задачі;
* Процедури *MatrixInput, MatrixOutput*, які призначені для організації вводу/виводу матриць;
* Процедури *CountMK*, *CountMA,* які призначені для обчислення значення заданого векторно-матричного виразу.

Алгоритми роботи основної програми та процесів наведено у додатках Б, В.

Лістинг розробленої програми наведено у додатку Г.

## 1.5. Тестування програми ПРГ1

Метою проведення тестування є оцінка коефіцієнтів прискорення і коефіцієнтів ефективності для розробленої програми ПРГ1 при її виконанні на реальній паралельній обчислювальній системі. Для визначення вищевказаних коефіцієнтів було проведено ряд експериментів із різними розмірностями матриць і різною кількістю фізичних ядер процесора .

Для виміру часу використовується метод *Clock* мови програмування *Ada*, що міститься в пакеті *Ada.Calendar*, який повертає поточний час. Щоб визначити час роботи програми, необхідно отримати час початку та закінчення виконання програми та обчислити різницю цих двох часів.

Для оцінки ефективності програми використовуються коефіцієнти прискорення та ефективності.

Коефіцієнт прискорення показує скорочення часу виконання паралельної програми в паралельній системі з процесорами в порівнянні з часом виконання послідовної програми в однопроцесорній системі :

Коефіцієнт ефективності застосування комп’ютерної системи показує ступінь використання процесорів системи:

Результати тестування і проведених досліджень ефективності розробленої програми наведено в табл.1.2 – 1.4.

Таблиця 1.2. Час виконання програми для ПРГ1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 900 | 43 | 26 | 19 | 15 |
| 1800 | 438 | 288 | 217 | 151 |
| 2400 | 884 | 541 | 407 | 369 |

На основі даних із табл. 1.2 виконано розрахунок значень коефіцієнтів прискорення, які наведені в табл. 1.3.

Таблиця 1.3. Значення коефіцієнтів прискорення для ПРГ1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  |  |  |  |
| 900 | 1 | 1,65 | 2,26 | 2,87 |
| 1800 | 1 | 1,52 | 2,02 | 2,9 |
| 2400 | 1 | 1,63 | 2,17 | 2,4 |

На основі даних із табл. 1.3 виконано розрахунок значень коефіцієнтів ефективності, які наведені в табл. 1.4.

Таблиця 1.4. Значення коефіцієнтів ефективності для ПРГ1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  |  |  |  |
| 900 | 100 | 82,7 | 75,4 | 71,7 |
| 1800 | 100 | 76 | 67,3 | 72,5 |
| 2400 | 100 | 81,7 | 72,4 | 59,9 |

За даними табл. 1.2 – 1.4 побудовано графіки зміни часу виконання обчислень та коефіцієнтів прискорення і ефективності в залежності від і (рис. 1.2 – 1.4).

Рис. 1.2. Графік залежності часу виконання програми ПРГ1 від кількості процесорів

Рис. 1.3. Графік залежності коефіцієнту прискорення від кількості процесорів

Рис. 1.4. Графік залежності коефіцієнту ефективності від кількості процесорів

## 1.6. Висновки до розділу 1

Виконано розробку програми ПРГ1 для ПКС із СП. При розробці використовувалась мова програмування *Ada.* Для взаємодії процесів використовувались механізм захищених модулів.

Тестування програми показало наступне:

* Використання багатоядерної ПКС забезпечує скорочення часу обчислення заданої математичної задачі.
* Значення лежить в межах від до ;
* Максимальне значення забезпечує ПКС з та ;
* Мінімальне значення забезпечує ПКС з та ;
* Коефіцієнт прискорення збільшується пропорційно до кількості використовуваних процесорів;
* Значення лежить в межах від до , що є достатнім приводом для використання багатоядерних ПКС для вирішення даної векторно-матричної задачі;
* Максимальне значення забезпечує ПКС з та ;
* Мінімальне значення забезпечує ПКС з та ;
* Коефіцієнт ефективності зменшується пропорційно до кількості використовуваних процесорів.

# РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС ЛП

Математична задача:

Мова програмування: *Ada.*

Засоби взаємодії процесів: механізм рандеву.

Структура паралельної комп’ютерної системи з локальною пам’яттю (ПКС ЛП) представлена на рис 2.1.

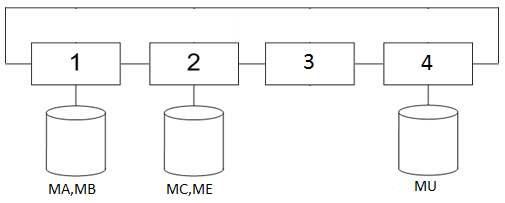


Рис.2.1. Структура ПКС з локальною пам’яттю (кільцева)

## 2.1. Розробка паралельного математичного алгоритму

Паралельний математичний алгоритм відповідно до рекомендованої методики [1] подано у табл. 2.1.

Таблиця 2.1. Паралельний алгоритм задачі

|  |  |
| --- | --- |
| № кроку | Дія |
| 1 |  |
| 2 |  |

де:

* *N* – розмірність матриць;
* *P* – кількість процесорів;
* ;
* *MBH* — *H* рядкiв матриці *MB*;
* *MCH* — *H* рядкiв матрицi*MC*;
* *MKH* — *H* рядкiв матрицi*MK*;
* *MUH* — *H* рядкiв матрицi *MU;*
* *MAH* — *H* рядкiв матрицi *MA*.

## 2.2. Розробка алгоритмів процесів

Алгоритм задачі

1. Введення
2. Надіслати в задачу
3. Надіслати в задачу
4. Отримати від задачі
5. Отримати від задачі
6. Надіслати в задачу
7. Обчислення:
8. Надіслати в задачу
9. Отримати від задачі
10. Обчислення:
11. Отримати від задачі
12. Отримати від задачі
13. Виведення результату

Алгоритм задачі

1. Введення
2. Надіслати в задачу
3. Отримати від задачі
4. Надіслати в задачу
5. Отримати від задачі
6. Обчислення:
7. Отримати від задачі
8. Надіслати в задачу
9. Отримати від задачі
10. Обчислення:
11. Отримати від задачі
12. Надіслати в задачу

Алгоритм задачі

1. Отримати від задачі
2. Отримати від задачі
3. Надіслати в задачу
4. Отримати від задачі
5. Обчислення:
6. Отримати від задачі
7. Отримати від задачі
8. Надіслати в задачу
9. Надіслати в задачу
10. Обчислення:
11. Надіслати в задачу

Алгоритм задачі

1. Введення
2. Надіслати в задачу
3. Отримати від задачі
4. Надіслати в задачу
5. Надіслати в задачу
6. Отримати від задачі
7. Обчислення:
8. Надіслати в задачу
9. Отримати від задачі
10. Надіслати в задачу
11. Обчислення:
12. Надіслати в задачу

## 2.3. Розробка схеми взаємодії процесів

На основі алгоритму процесів було розроблено структурну схему взаємодії процесів. За допомогою цієї схеми можна наочно побачити як саме відбувається пересилка даних.

Схема взаємодії процесів кільцевої структури ПКС ЛП наведена у додатку Д.

## 2.4. Розробка програми ПРГ2

Програма ПРГ2, згідно з технічним завданням, розроблена на мові програмування *Ada*. Для взаємодії процесів використовуються механізм рандеву.

Передача повідомлень між процесами та синхронізація процесів відбувається за допомогою виклику входу *entry* та прийняття виклику входу *accept*.

Програма складається з файлу *prg2.abd*, що містить в собі:

* Процедуру *PRG2* – точка входу в програму, запускає процедуру, яка запускає задачі, вимірює час виконання програми ПРГ2;
* Визначення констант *N, P, H*;
* Процедуру *StartTasks*, яка містить специфікації та тіла задач *T1, T2, T3, T4*, запускає ці задачі;
* Процедури *MatrixInput, MatrixOutput*, які призначені для організації вводу/виводу матриць;
* Процедури *CountMK*, *CountMA,* які призначені для обчислення значення заданого векторно-матричного виразу.

Алгоритми роботи основної програми та процесів наведено у додатках Е, Ж.

Лістинг розробленої програми наведено у додатку И.

## 2.5. Тестування програми ПРГ2

Метою проведення тестування є оцінка коефіцієнтів прискорення і коефіцієнтів ефективності для розробленої програми ПРГ2 при її виконанні на реальній паралельній обчислювальній системі. Для визначення вищевказаних коефіцієнтів було проведено ряд експериментів із різними розмірностями матриць і різною кількістю фізичних ядер процесора .

Для виміру часу використовується метод *Clock* мови програмування *Ada*, що міститься в пакеті *Ada.Calendar*, який повертає поточний час. Щоб визначити час роботи програми, необхідно отримати час початку та закінчення виконання програми та обчислити різницю цих двох часів.

Для оцінки ефективності програми використовуються коефіцієнти прискорення та ефективності.

Коефіцієнт прискорення показує скорочення часу виконання паралельної програми в паралельній системі з процесорами в порівнянні з часом виконання послідовної програми в однопроцесорній системі:

Коефіцієнт ефективності застосування комп’ютерної системи показує ступінь використання процесорів системи:

Результати тестування і проведених досліджень ефективності розробленої програми наведено в табл.2.2 – 2.4.

Таблиця 2.2. Час виконання програми для ПРГ2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 900 | 46 | 23 | 18 | 16 |
| 1800 | 427 | 245 | 184 | 146 |
| 2400 | 865 | 528 | 360 | 288 |

На основі даних із табл. 2.2 виконано розрахунок значень коефіцієнтів прискорення, які наведені в табл. 2.3.

Таблиця 2.3. Значення коефіцієнтів прискорення для ПРГ2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  |  |  |  |
| 900 | 1 | 2 | 2,56 | 2,88 |
| 1800 | 1 | 1,74 | 2,32 | 2,92 |
| 2400 | 1 | 1,64 | 2,4 | 3 |

На основі даних із табл. 2.3 виконано розрахунок значень коефіцієнтів ефективності, які наведені в табл. 2.4.

Таблиця 2.4. Значення коефіцієнтів ефективності для ПРГ2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  |  |  |  |
| 900 | 100 | 100 | 85,2 | 71,9 |
| 1800 | 100 | 87,1 | 77,4 | 73,1 |
| 2400 | 100 | 81,9 | 80,1 | 75,1 |

За даними табл. 2.2 – 2.4 побудовано графіки зміни часу виконання обчислень та коефіцієнтів прискорення і ефективності в залежності від і (рис. 2.2 – 2.4).

Рис. 2.2. Графік залежності часу виконання програми ПРГ2 від кількості процесорів

Рис. 2.3. Графік залежності коефіцієнту прискорення від кількості процесорів

Рис. 2.4. Графік залежності коефіцієнту ефективності від кількості процесорів

## 2.6. Висновки до розділу 2

Виконано розробку програми ПРГ2 для ПКС із ЛП. При розробці використовувалась мова програмування *Ada.* Для взаємодії процесів використовувався механізм рандеву.

Тестування програми показало наступне:

* Використання багатоядерної ПКС забезпечує скорочення часу обчислення заданої математичної задачі.
* Значення лежить в межах від до ;
* Максимальне значення забезпечує ПКС з та ;
* Мінімальне значення забезпечує ПКС з та ;
* Коефіцієнт прискорення збільшується пропорційно до кількості використовуваних процесорів;
* Значення лежить в межах від до , що є достатнім приводом для використання багатоядерних ПКС для вирішення даної векторно-матричної задачі;
* Максимальне значення забезпечує ПКС з та ;
* Мінімальне значення забезпечує ПКС з та ;
* Коефіцієнт ефективності зменшується пропорційно до кількості використовуваних процесорів.

# ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ ДО РОБОТИ

1. Виконано проектування, розробку та тестування програми ПРГ1 для вирішення векторно-матричної задачі на ПКС з СП на мові програмування *Ada* за допомогою використання механізму захищених модулів.
2. Виконано проектування, розробку та тестування програми ПРГ2 для вирішення векторно-матричної задачі на ПКС з ЛП на мові програмування *Ada* за допомогою використання механізму рандеву.
3. Перевагою ПКС з СП над ПКС з ЛП є ресурси, які не копіюються для кожного процесу. При виконанні програма, яка реалізує задачу в системі зі СП буде займати менше пам’яті, ніж програма, яка реалізує таку саму задачу в системі з ЛП. Алгоритм роботи для ПКС з СП значно легший як для проектування, так і для реалізації програми, ніж у ПКС з ЛП. У випадку використання ПКС з ЛП, велика кількість повідомлень різного розміру ускладнюють систему з пересилкою повідомлень, а також підвищують вірогідність помилки при реалізації.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Жуков І. А. Паралельні та розподілені обчислення: Навч.посіб. / І. А. Жуков, О. В. Корочкін. – Київ : «Корнійчук», 2005. – 226 с.

Эндрюс Г. Основы многопоточного, параллельного и распределенного програмирования./ Г. Эндрюс. – Пер. с англ. – М.: Изд. Дом «Виль – ямс», 2003. – 512 с.

# ДОДАТКИ

# Додаток А

*Змн.*.

*Арк.*

*№ докум.*

*Підпис*

*Дата*

*Арк.*

*1*

*ОТ.6.050103 ІП-54*

*Розроб.*

*Макаренко А.О*

*Перевір.*

*Корочкін О.В.*

*Н. Контр.*

*Затверд.*

*Корочкін О.В.*

*ПРГ1. Схема взаємодії процесів*

*Літ.*

*Аркушів*

*1*

*НТУУ «КПІ ім. І.Сікорського», ФІОТ*

*ІП-32*



# Додаток Б

*Змн.*.

*Арк.*

*№ докум.*

*Підпис*

*Дата*

*Арк.*

*1*

*ОТ.6.050103 ІП-54*

*Розроб.*

*Макаренко А.О*

*Перевір.*

*Корочкін О.В.*

*Н. Контр.*

*Затверд.*

*Корочкін О.В.*

*ПРГ1. Алгоритм роботи основної програми*

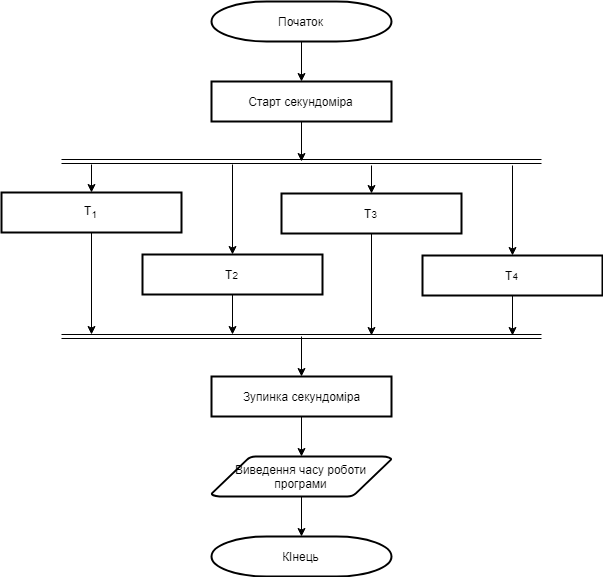
*Літ.*

*Аркушів*

*1*

*НТУУ «КПІ ім. І.Сікорського», ФІОТ*

*ІП-32*



# Додаток В

*Змн.*.

*Арк.*

*№ докум.*

*Підпис*

*Дата*

*Арк.*

*1*

*ОТ.6.050103 ІП-54*

*Розроб.*

*Макаренко А.О*

*Перевір.*

*Корочкін О.В.*

*Н. Контр.*

*Затверд.*

*Корочкін О.В.*

*ПРГ1. Алгоритм роботи задач T1,T2,T3,T4*

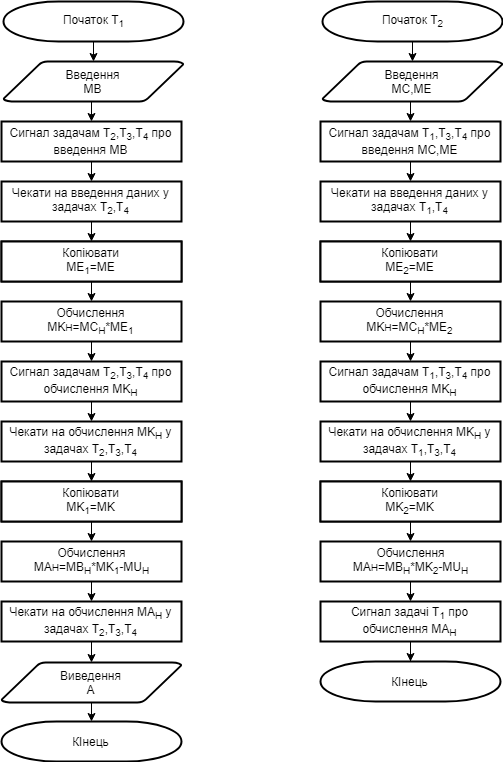
*Літ.*

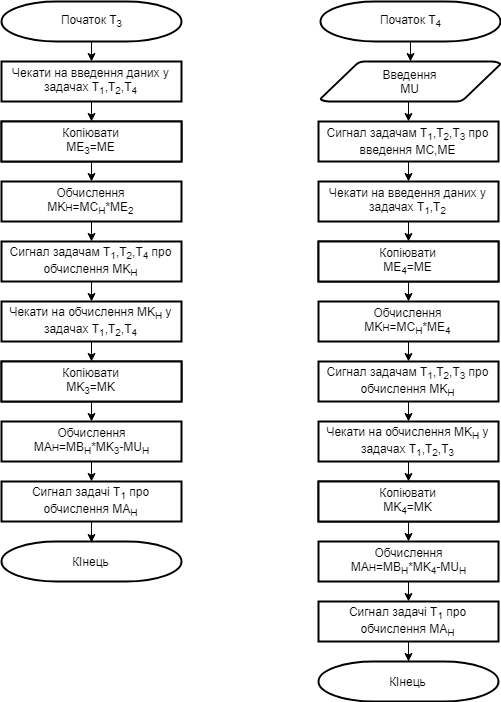
*Аркушів*

*2*

*НТУУ «КПІ ім. І.Сікорського», ФІОТ*

*ІП-32*





*Змн.*

Арк.

*№ докум.*

*Підпис*

*Дата*

*Арк.*

*2*

*ОТ.6.050103 ІП-54*

# Додаток Г

Лістинг ПРГ1

GNAT GPL 2017 (20170515-63)

Copyright 1992-2017, Free Software Foundation, Inc.

Compiling: prg1.adb

Source file time stamp: 2018-05-01 11:31:52

Compiled at: 2018-05-06 20:19:52

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

2. -- Paralel'ne prohramuvannia - 2 --

3. -- PRG1 --

4. -- MA = MB \* (MC \* ME) - MU --

5. -- Boiko Serhii --

6. -- IP-53 --

7. -- 27.04.2018 --

8. -----------------------------------------

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

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

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

12. with Ada.Calendar; use Ada.Calendar;

13.

14. procedure PRG1 is

15. N: Integer := 4;

16. P: Integer := 4;

17. H: Integer := N/P;

18. f: Integer := 1;

19.

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

21. subtype VectorH is Vector(1..H);

22. subtype VectorN is Vector(1..N);

23. type Matrix is array (Integer range <>) of VectorN;

24. subtype MatrixH is Matrix(1..H);

25. subtype MatrixN is Matrix(1..N);

26.

27. MA, MB, MC, ME, MU : MatrixN;

28.

29. protected Resources is

30. procedure SetME(X : in MatrixN);

31. function CopyME return MatrixN;

32. procedure SetMK(taskId : in Integer; X : in MatrixH);

33. function CopyMK return MatrixN;

34. private

35. ME, MK : MatrixN;

36. end Resources;

37. protected body Resources is

38. procedure SetME (X : in MatrixN) is

39. begin

40. ME := X;

41. end SetME;

42. function CopyME return MatrixN is

43. begin

44. return ME;

45. end CopyME;

46. procedure SetMK (taskId : in Integer; X : in MatrixH) is

47. begin

48. MK(H\*(taskId-1)+1..H\*taskId) := X;

49. end SetMK;

50. function CopyMK return MatrixN is

51. begin

52. return MK;

53. end CopyMK;

54. end Resources;

55.

56. protected Synchro is

57. procedure SignalInput;

58. procedure SignalCountMK;

59. procedure SignalEndCount;

60. entry WaitInput;

61. entry WaitCountMK;

62. entry WaitEndCount;

63. private

64. F1 : Integer := 0;

65. F2 : Integer := 0;

66. F3 : Integer := 0;

67. end Synchro;

68. protected body Synchro is

69. procedure SignalInput is

70. begin

71. F1 := F1 + 1;

72. end SignalInput;

73. procedure SignalCountMK is

74. begin

75. F2 := F2 + 1;

76. end SignalCountMK;

77. procedure SignalEndCount is

78. begin

79. F3 := F3 + 1;

80. end SignalEndCount;

81. entry WaitInput when F1 = 3 is

82. begin

83. null;

84. end WaitInput;

85. entry WaitCountMK when F2 = 4 is

86. begin

87. null;

88. end WaitCountMK;

89. entry WaitEndCount when F3 = 3 is

90. begin

91. null;

92. end WaitEndCount;

93. end Synchro;

94.

95. procedure MatrixInput(VMA : out MatrixN) is

96. begin

97. for i in 1..N loop

98. for j in 1..N loop

99. VMA(i)(j) := f;

100. end loop;

101. end loop;

102. end;

103. procedure MatrixOutput(VMA : in MatrixN) is

104. begin

105. if N < 10 then

106. for i in 1..N loop

107. for j in 1..N loop

108. Put(VMA(i)(j), 10);

109. end loop;

110. Put\_Line("");

111. end loop;

112. end if;

113. end;

114. procedure CountMK(taskId : in Integer; VME : in MatrixN; VMK : out MatrixH) is

115. begin

116. for i in 1..H loop

117. for j in 1..N loop

118. VMK(i)(j) := 0;

119. end loop;

120. end loop;

121. for i in H\*(taskId-1)+1..H\*taskId loop

122. for j in 1..N loop

123. for k in 1..N loop

124. VMK(i-H\*(taskId-1))(j) := VMK(i-H\*(taskId-1))(j) + MC(i)(k) \* VME(k)(j);

125. end loop;

126. end loop;

127. end loop;

128. end;

129. procedure CountMA(taskId : in Integer; VMK : in MatrixN) is

130. begin

131. for i in H\*(taskId-1)+1..H\*taskId loop

132. for j in 1..N loop

133. MA(i)(j) := 0;

134. end loop;

135. end loop;

136. for i in H\*(taskId-1)+1..H\*taskId loop

137. for j in 1..N loop

138. for k in 1..N loop

139. MA(i)(j) := MA(i)(j) + MB(i)(k) \* VMK(k)(j);

140. end loop;

141. MA(i)(j) := MA(i)(j) - MU(i)(j);

142. end loop;

143. end loop;

144. end;

145. procedure StartTasks is

146. task T1;

147. task body T1 is

148. taskId : Integer := 1;

149. MK : MatrixH;

150. ME1, MK1 : MatrixN;

151. begin

152. Put\_Line(" Process T1 started ");

153. -- 1. input MB

154. MatrixInput(MB);

155. -- 2. signal to T2,T3,T4 end input

156. Synchro.SignalInput;

157. -- 3. wait end input from T2,T4

158. Synchro.WaitInput;

159. -- 4. copy

160. ME1 := Resources.CopyME;

161. -- 5. count MK

162. CountMK(taskId, ME1, MK);

163. Resources.SetMK(taskId, MK);

164. -- 6. signal to T2,T3,T4 end count MK

165. Synchro.SignalCountMK;

166. -- 7. wait end count MK from T2,T3,T4

167. Synchro.WaitCountMK;

168. -- 8. copy

169. MK1 := Resources.CopyMK;

170. -- 9. count

171. CountMA(taskId, MK1);

172. -- 10. wait end count from T2,T3,T4

173. Synchro.WaitEndCount;

174. -- 11. output result MA

175. MatrixOutput(MA);

176. Put\_Line(" Process T1 finished ");

177. end T1;

178. task T2;

179. task body T2 is

180. taskId : Integer := 2;

181. MK : MatrixH;

182. ME2, MK2 : MatrixN;

183. begin

184. Put\_Line(" Process T2 started ");

185. -- 1. input MC,ME

186. MatrixInput(MC);

187. MatrixInput(ME);

188. Resources.SetME(ME);

189. -- 2. signal to T1,T3,T4 end input

190. Synchro.SignalInput;

191. -- 3. wait end input from T1,T4

192. Synchro.WaitInput;

193. -- 4. copy

194. ME2 := Resources.CopyME;

195. -- 5. count MK

196. CountMK(taskId, ME2, MK);

197. Resources.SetMK(taskId, MK);

198. -- 6. signal to T1,T3,T4 end count MK

199. Synchro.SignalCountMK;

200. -- 7. wait end count MK from T1,T3,T4

201. Synchro.WaitCountMK;

202. -- 8. copy

203. MK2 := Resources.CopyMK;

204. -- 9. count

205. CountMA(taskId, MK2);

206. -- 10. signal to T1 end count

207. Synchro.SignalEndCount;

208. Put\_Line(" Process T2 finished ");

209. end T2;

210. task T3;

211. task body T3 is

212. taskId : Integer := 3;

213. MK : MatrixH;

214. ME3, MK3 : MatrixN;

215. begin

216. Put\_Line(" Process T3 started ");

217. -- 1. wait end input from T1,T2,T4

218. Synchro.WaitInput;

219. -- 2. copy

220. ME3 := Resources.CopyME;

221. -- 3. count MK

222. CountMK(taskId, ME3, MK);

223. Resources.SetMK(taskId, MK);

224. -- 4. signal to T1,T2,T4 end count MK

225. Synchro.SignalCountMK;

226. -- 5. wait end count MK from T1,T2,T4

227. Synchro.WaitCountMK;

228. -- 6. copy

229. MK3 := Resources.CopyMK;

230. -- 7. count

231. CountMA(taskId, MK3);

232. -- 8. signal to T1 end count

233. Synchro.SignalEndCount;

234. Put\_Line(" Process T3 finished ");

235. end T3;

236. task T4;

237. task body T4 is

238. taskId : Integer := 4;

239. MK : MatrixH;

240. ME4, MK4 : MatrixN;

241. begin

242. Put\_Line(" Process T4 started ");

243. -- 1. input MU

244. MatrixInput(MU);

245. -- 2. signal to T1,T2,T3 end input

246. Synchro.SignalInput;

247. -- 3. wait end input from T1,T2

248. Synchro.WaitInput;

249. -- 4. copy

250. ME4 := Resources.CopyME;

251. -- 5. count MK

252. CountMK(taskId, ME4, MK);

253. Resources.SetMK(taskId, MK);

254. -- 6. signal to T1,T2,T3 end count MK

255. Synchro.SignalCountMK;

256. -- 7. wait end count MK from T1,T2,T3

257. Synchro.WaitCountMK;

258. -- 8. copy

259. MK4 := Resources.CopyMK;

260. -- 9. count

261. CountMA(taskId, MK4);

262. -- 10. signal to T1 end count

263. Synchro.SignalEndCount;

264. Put\_Line(" Process T4 finished ");

265. end T4;

266. begin

267. null;

268. end StartTasks;

269. StartTime, EndTime : Time;

270. Char : Character;

271. begin

272. Put("Press any key to begin:");

273. Get\_Immediate(Char);

274. StartTime := Clock;

275. StartTasks;

276. EndTime := Clock;

277. Put("Time: ");

278. Put(Integer(EndTime - StartTime), 10);

279. Put\_Line("");

280. Put("Press any key to end:");

281. Get\_Immediate(Char);

282. end PRG1;

282 lines: No errors

# Додаток Д

*Змн.*.

*Арк.*

*№ докум.*

*Підпис*

*Дата*

*Арк.*

*1*

*ОТ.6.050103 ІП-54*

*Розроб.*

*Макаренко А.О*

*Перевір.*

*Корочкін О.В.*

*Н. Контр.*

*Затверд.*

*Корочкін О.В.*

*ПРГ2. Схема взаємодії процесів*

*Літ.*

*Аркушів*

*1*

*НТУУ «КПІ ім. І.Сікорського», ФІОТ*

*ІП-32*



# Додаток Е

*Змн.*.

*Арк.*

*№ докум.*

*Підпис*

*Дата*

*Арк.*

*1*

*ОТ.6.050103 ІП-54*

*Розроб.*

*Макаренко А.О*

*Перевір.*

*Корочкін О.В.*

*Н. Контр.*

*Затверд.*

*Корочкін О.В.*

*ПРГ2. Алгоритм роботи основної програми*

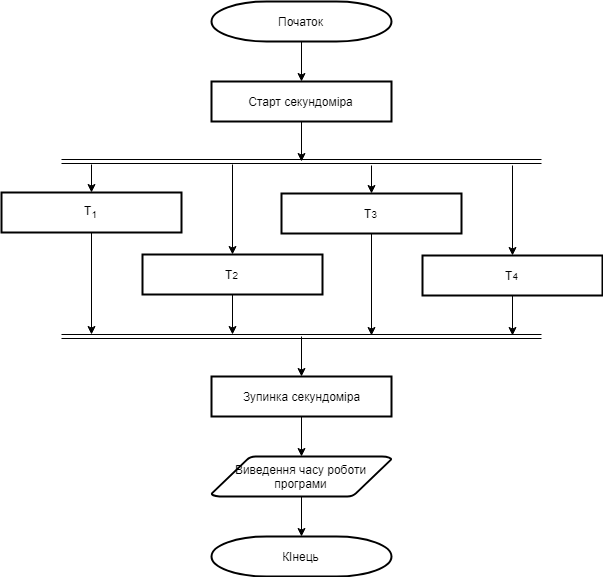
*Літ.*

*Аркушів*

*1*

*НТУУ «КПІ ім. І.Сікорського», ФІОТ*

*ІП-32*



# C:\Users\Maksym\Downloads\Bo_1 (3).pngДодаток Ж

*Змн.*.

*Арк.*

*№ докум.*

*Підпис*

*Дата*

*Арк.*

*1*

*ОТ.6.050103 ІП-54*

*Розроб.*

*Макаренко А.О*

*Перевір.*

*Корочкін О.В.*

*Н. Контр.*

*Затверд.*

*Корочкін О.В.*

*ПРГ2. Алгоритм роботи задач T1,T2,T3,T4*

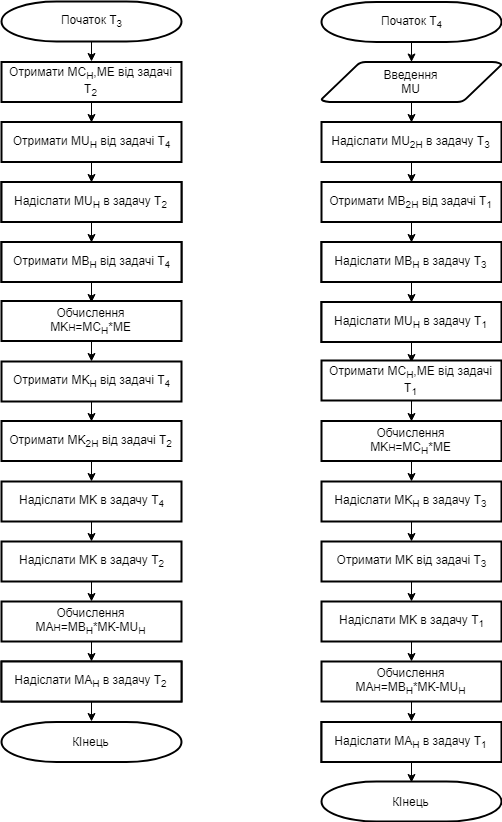
*Літ.*

*Аркушів*

*2*

*НТУУ «КПІ ім. І.Сікорського», ФІОТ*

*ІП-32*



*Змн.*

Арк.

*№ докум.*

*Підпис*

*Дата*

*Арк.*

*2*

*ОТ.6.050103 ІП-54*

# Додаток И

Лістинг ПРГ2

GNAT GPL 2017 (20170515-63)

Copyright 1992-2017, Free Software Foundation, Inc.

Compiling: prg2.adb

Source file time stamp: 2018-05-01 11:01:28

Compiled at: 2018-05-06 20:21:04

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

2. -- Paralel'ne prohramuvannia - 2 --

3. -- PRG2 --

4. -- MA = MB \* (MC \* ME) - MU --

5. -- Boiko Serhii --

6. -- IP-53 --

7. -- 27.04.2018 --

8. -----------------------------------------

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

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

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

12. with Ada.Calendar; use Ada.Calendar;

13.

14. procedure PRG2 is

15. N: Integer := 4;

16. P: Integer := 4;

17. H: Integer := N/P;

18. f: Integer := 1;

19.

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

21. subtype VectorN is Vector(1..N);

22. type Matrix is array (Integer range <>) of VectorN;

23. subtype MatrixH is Matrix(1..H);

24. subtype Matrix2H is Matrix(1..H\*2);

25. subtype MatrixN is Matrix(1..N);

26.

27. procedure MatrixInput(VMA : out MatrixN) is

28. begin

29. for i in 1..N loop

30. for j in 1..N loop

31. VMA(i)(j) := f;

32. end loop;

33. end loop;

34. end;

35. procedure MatrixOutput(VMA : in MatrixN) is

36. begin

37. if N < 10 then

38. for i in 1..N loop

39. for j in 1..N loop

40. Put(VMA(i)(j), 10);

41. end loop;

42. Put\_Line("");

43. end loop;

44. end if;

45. end;

46. procedure CountMK(VMC : in MatrixH; VME : in MatrixN; VMK : out MatrixH) is

47. begin

48. for i in 1..H loop

49. for j in 1..N loop

50. VMK(i)(j) := 0;

51. end loop;

52. end loop;

53. for i in 1..H loop

54. for j in 1..N loop

55. for k in 1..N loop

56. VMK(i)(j) := VMK(i)(j) + VMC(i)(k) \* VME(k)(j);

57. end loop;

58. end loop;

59. end loop;

60. end;

61. procedure CountMA(VMB : in MatrixH; VMK : in MatrixN; VMU : in MatrixH; VMA : out MatrixH) is

62. begin

63. for i in 1..H loop

64. for j in 1..N loop

65. VMA(i)(j) := 0;

66. end loop;

67. end loop;

68. for i in 1..H loop

69. for j in 1..N loop

70. for k in 1..N loop

71. VMA(i)(j) := VMA(i)(j) + VMB(i)(k) \* VMK(k)(j);

72. end loop;

73. VMA(i)(j) := VMA(i)(j) - VMU(i)(j);

74. end loop;

75. end loop;

76. end;

77. procedure StartTasks is

78. task T1 is

79. entry Data22(VMC : in Matrix2H; VME : in MatrixN);

80. entry Data44(VMU : in MatrixH);

81. entry MKh(VMK : out MatrixH);

82. entry MKn(VMK : in MatrixN);

83. end T1;

84. task T2 is

85. entry Data11(VMB : in MatrixH);

86. entry Data34(VMU : in MatrixH);

87. entry MKh(VMK : out Matrix2H);

88. entry MKn(VMK : in MatrixN);

89. entry Res2(VMA : out Matrix2H);

90. end T2;

91. task T3 is

92. entry Data22(VMC : in MatrixH; VME : in MatrixN);

93. entry Data44(VMU : in Matrix2H);

94. entry Data41(VMB : in MatrixH);

95. entry Res3(VMA : out MatrixH);

96. end T3;

97. task T4 is

98. entry Data11(VMB : in Matrix2H);

99. entry Data12(VMC : in MatrixH; VME : in MatrixN);

100. entry MKh(VMK : out MatrixH);

101. entry MKn(VMK : in MatrixN);

102. entry Res4(VMA : out MatrixH);

103. end T4;

104. task body T1 is

105. MU : MatrixH;

106. MC : Matrix2H;

107. MB, ME, MK, MA : MatrixN;

108. begin

109. Put\_Line(" Process T1 started ");

110. -- 1. input MB

111. MatrixInput(MB);

112. -- 2. send to T4 (input 1)

113. T4.Data11(MB(H\*2+1..H\*4));

114. -- 3. send to T3 (input 1)

115. T2.Data11(MB(H+1..H\*2));

116. -- 4. receive from T2 (input 2)

117. accept Data22 (VMC : in Matrix2H; VME : in MatrixN) do

118. MC := VMC;

119. ME := VME;

120. end Data22;

121. -- 5. receive from T4 (input 4)

122. accept Data44 (VMU : in MatrixH) do

123. MU := VMU;

124. end Data44;

125. -- 6. send to T4 (input 2)

126. T4.Data12(MC(H+1..H\*2), ME);

127. -- 7. count MK

128. CountMK(MC(1..H), ME, MK(1..H));

129. -- 8. send to T2 (MKh)

130. accept MKh (VMK : out MatrixH) do

131. VMK := MK(1..H);

132. end MKh;

133. -- 9. receive from T4 (MK)

134. accept MKn (VMK : in MatrixN) do

135. MK := VMK;

136. end MKn;

137. -- 10. count

138. CountMA(MB(1..H), MK, MU, MA(1..H));

139. -- 11. receive from T4 (MA)

140. T4.Res4(MA(H\*3+1..H\*4));

141. -- 12. receive from T2 (MA)

142. T2.Res2(MA(H+1..H\*3));

143. -- 13. output result MA

144. MatrixOutput(MA);

145. Put\_Line(" Process T1 finished ");

146. end T1;

147. task body T2 is

148. MB, MU : MatrixH;

149. MC14, MA : Matrix2H;

150. MC, ME, MK : MatrixN;

151. begin

152. Put\_Line(" Process T2 started ");

153. -- 1. input MC,ME

154. MatrixInput(MC);

155. MatrixInput(ME);

156. -- 2. send to T3 (input 2)

157. T3.Data22(MC(H\*2+1..H\*3), ME);

158. -- 3. receive from T1 (input 1)

159. accept Data11 (VMB : in MatrixH) do

160. MB := VMB;

161. end Data11;

162. -- 4. send to T1 (input 2)

163. MC14(1..H) := MC(1..H);

164. MC14(H+1..H\*2) := MC(H\*3+1..H\*4);

165. T1.Data22(MC14, ME);

166. -- 5. receive from T3 (input 4)

167. accept Data34 (VMU : in MatrixH) do

168. MU := VMU;

169. end Data34;

170. -- 6. count MK

171. CountMK(MC(H+1..H\*2), ME, MK(H+1..H\*2));

172. -- 7. receive from T1 (MKh)

173. T1.MKh(MK(1..H));

174. -- 8. send to T3 (MKh)

175. accept MKh (VMK : out Matrix2H) do

176. VMK := MK(1..H\*2);

177. end MKh;

178. -- 9. receive from T3 (MK)

179. accept MKn (VMK : in MatrixN) do

180. MK := VMK;

181. end MKn;

182. -- 10. count

183. CountMA(MB, MK, MU, MA(1..H));

184. -- 11. receive from T3 (MA)

185. T3.Res3(MA(H+1..H\*2));

186. -- 12. send to T1 (MA)

187. accept Res2 (VMA : out Matrix2H) do

188. VMA := MA;

189. end Res2;

190. Put\_Line(" Process T2 finished ");

191. end T2;

192. task body T3 is

193. MC, MB, MA : MatrixH;

194. MU : Matrix2H;

195. ME, MK : MatrixN;

196. begin

197. Put\_Line(" Process T3 started ");

198. -- 1. receive from T2 (input 2)

199. accept Data22 (VMC : in MatrixH; VME : in MatrixN) do

200. MC := VMC;

201. ME := VME;

202. end Data22;

203. -- 2. receive from T4 (input 4)

204. accept Data44 (VMU : in Matrix2H) do

205. MU := VMU;

206. end Data44;

207. -- 3. send to T2 (input 4)

208. T2.Data34(MU(1..H));

209. -- 4. receive from T4 (input 1)

210. accept Data41 (VMB : in MatrixH) do

211. MB := VMB;

212. end Data41;

213. -- 5. count MK

214. CountMK(MC, ME, MK(H\*2+1..H\*3));

215. -- 6. receive from T4 (MKh)

216. T4.MKh(MK(H\*3+1..H\*4));

217. -- 7. receive from T2 (MKh)

218. T2.MKh(MK(1..H\*2));

219. -- 8. send to T4 (MKn)

220. T4.MKn(MK);

221. -- 9. send to T2 (MKn)

222. T2.MKn(MK);

223. -- 10. count

224. CountMA(MB, MK, MU(H+1..H\*2), MA);

225. -- 11. send to T2 (MA)

226. accept Res3 (VMA : out MatrixH) do

227. VMA := MA;

228. end Res3;

229. Put\_Line(" Process T3 finished ");

230. end T3;

231. task body T4 is

232. MC, MA : MatrixH;

233. MB : Matrix2H;

234. MU, ME, MK : MatrixN;

235. begin

236. Put\_Line(" Process T4 started ");

237. -- 1. input MU

238. MatrixInput(MU);

239. -- 2. send to T3 (input 4)

240. T3.Data44(MU(H+1..H\*3));

241. -- 3. receive from T1 (input 1)

242. accept Data11 (VMB : in Matrix2H) do

243. MB := VMB;

244. end Data11;

245. -- 4. send to T3 (input 1)

246. T3.Data41(MB(1..H));

247. -- 5. send to T1 (input 4)

248. T1.Data44(MU(1..H));

249. -- 6. receive from T1 (input 2)

250. accept Data12 (VMC : in MatrixH; VME : in MatrixN) do

251. MC := VMC;

252. ME := VME;

253. end Data12;

254. -- 7. count MK

255. CountMK(MC, ME, MK(H\*3+1..H\*4));

256. -- 8. send to T3 (MKh)

257. accept MKh (VMK : out MatrixH) do

258. VMK := MK(H\*3+1..H\*4);

259. end MKh;

260. -- 9. receive from T3 (MK)

261. accept MKn (VMK : in MatrixN) do

262. MK := VMK;

263. end MKn;

264. -- 10. send to T1 (MK)

265. T1.MKn(MK);

266. -- 11. count

267. CountMA(MB(H+1..H\*2), MK, MU(H\*3+1..H\*4), MA);

268. -- 12. send to T1 (MA)

269. accept Res4 (VMA : out MatrixH) do

270. VMA := MA;

271. end Res4;

272. Put\_Line(" Process T4 finished ");

273. end T4;

274. begin

275. null;

276. end StartTasks;

277. StartTime, EndTime : Time;

278. Char : Character;

279. begin

280. Put("Press any key to begin:");

281. Get\_Immediate(Char);

282. StartTime := Clock;

283. StartTasks;

284. EndTime := Clock;

285. Put("Time: ");

286. Put(Integer(EndTime - StartTime), 10);

287. Put\_Line("");

288. Put("Press any key to end:");

289. Get\_Immediate(Char);

290. end PRG2;

290 lines: No errors