Міністерство освіти і науки України
Національний технічний університет України “Київський політехнічний інститут”
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Курсова робота

з дисципліни “Паралельні та розподілені обчислення”

Керівник роботи:
Доц. Корочкін О. В.

Виконавець роботи:
ст. Горобець О. М.
ФІОТ гр. ІО-72

Допущена до захисту
_____«___» 2010 р.
Захищена
_____«___» 2010 р.

Київ — 2010

Технічне завдання

Р1 Порівняння реалізації механізму семафорів в мовах Ада, С #, бібліотеці Win32

Р2

Java

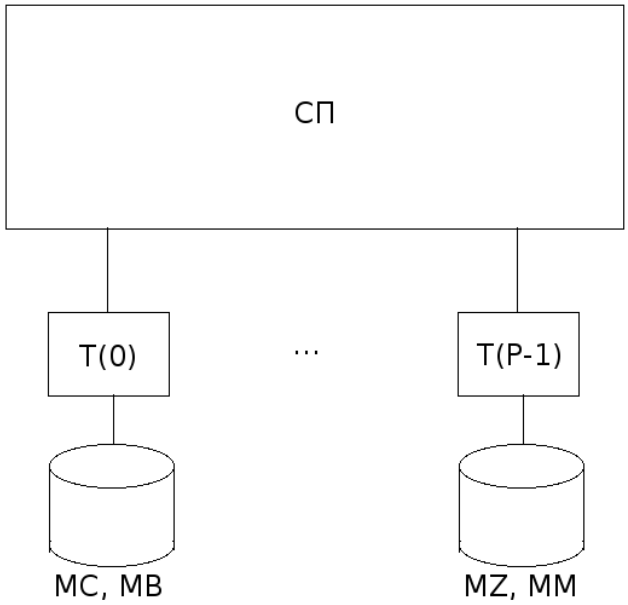

Рис 1. Структура паралельної обчислювальної системи зі спільною пам'яттю

Ada

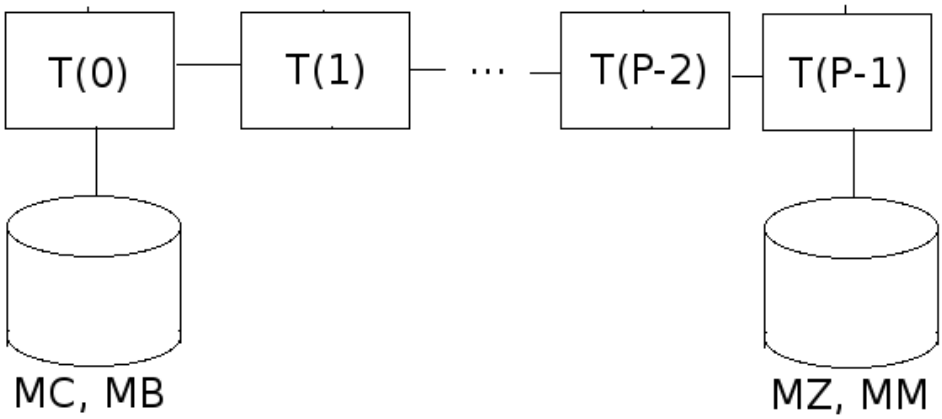

Рис 2. Структура паралельної обчислювальної системи з локальною пам'яттю

MA = MZ*(MB*MC) — MM

Зміст

3.4 Висновки до розділу 3

1. Порівняння реалізації механізму семафорів в мовах Ада, C #, бібліотеці Win32

1.1 Механізм семафорів.

Семафор - це захищена змінна, значення якої можна опитувати і міняти тільки за допомогою спеціальних операцій wait і signal та операції ініціалізації init. Двійкові семафори можуть приймати тільки значення 0 і 1. Семафори з лічильниками можуть бути невід'ємним цілим числом.

Операція wait (s) над семафором s полягає в наступному:

якщо s> 0 то s: = s-1 інакше (очікувати на s)

операція signal (s) полягає в тому, що:

якщо (маються процеси, які чекають на s)

то (дозволити одному з них продовжити роботу)

інакше s: = s +1

Операції є неподільними. Критичні ділянки процесів обрамляються операціями wait (s) і signal (s). Якщо одночасно декілька процесів спробують виконати операцію wait (s), то це буде дозволено тільки одному з них, а решті доведеться чекати.

Семафори з лічильниками використовуються, якщо деякі ресурс виділяється з множини ідентичних ресурсів. При ініціалізації такого семафора в його лічильнику вказується число елементів множини. Кожна операція wait (s) зменшує значення лічильника семафора s на 1, показуючи, що деякому процесу виділений один ресурс із множини. Кожна операція signal (s) збільшує значення лічильника на 1, показуючи, що процес повернув ресурс у множину. Якщо операція wait (s) виконується, коли в лічильнику міститься нуль (більше немає ресурсів), то відповідний процес чекає, поки в множину не буде повернуто звільнився ресурс, тобто поки не буде виконана операція signal.

Семафори традиційно використовувалися для синхронізації процесів, які звертаються до даних, що розділяються. Кожен процес має виключати для всіх інших процесів можливість одночасно з ним звертатися до цих даних (взаємовиключення). [1]

У семафорів є кілька основних проблем. Можна написати програму з «витоком семафора», викликавши wait () і забувши викликати signal (). Рідше зустрічаються помилки, коли двічі викликається signal ().

По-друге, семафори чреваті взаємної блокуванням потоків. Зокрема, небезпечний такий код:

Потік 1:

semaphore1.wait();

semaphore2.wait ();

...

semaphore2.signal();

semaphore1.signal();

Потік 2:

semaphore2.wait();

semaphore1.wait ();

...

semaphore1.signal();

semaphore2.signal();

По-третє, проблема синхронізації процедур самого семафора. Наприклад, можлива така ситуація: два процеси чекають звільнення семафора. Після того, як семафор звільнився, перший процес «впізнає» про це, але не встигає зменшити лічильник, як управління передається другому процесу. Другий процес теж дізнається про звільнення семафора, зменшує лічильник і входить у

захищений ділянку коду. Тут управління передається першому процесу, той ще раз зменшує лічильник і теж входить у захищений ділянку коду. У підсумку маємо перевищення дозволеного числа процесів.

Дана проблема не має алгоритмічного рішення. Вона дозволяється або розміщенням процедури очікування в критичній секції, в якій не дозволяється переключення з процесу на процес, або програмістських прийомами на зразок здійснення перевірки прапора і його збільшення за допомогою однієї машинної команди. [2]

Механізм семафорів є універсальним, може бути використаний як для вирішення задачі взаємного віключення, так і для задачі синхроізації[3].

Малюнок 1 представляє собою загальну схему вирішення задачі взаємного виключення за допомогою семафорів

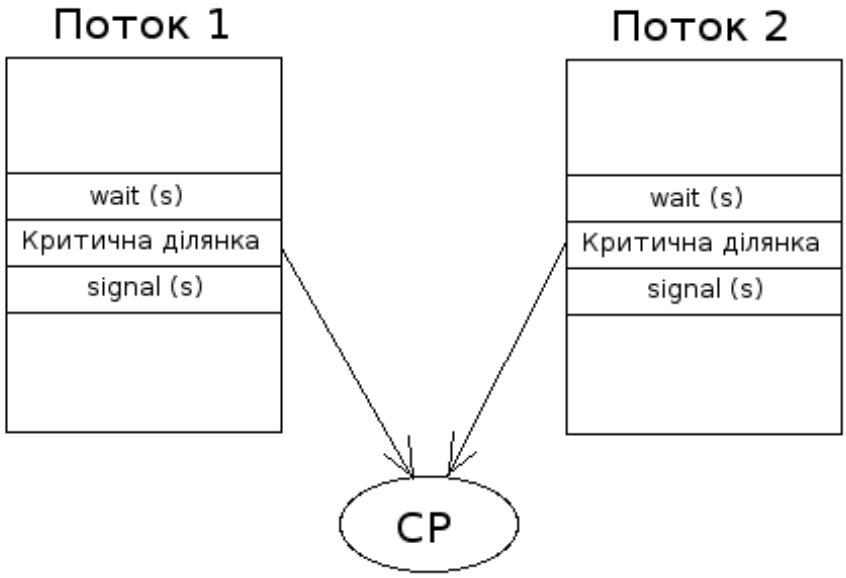

Рис 1.1. Загальна схема вирішення задачі взаємного виключення за допомогою семафорів.

Малюнок 2 представляє собою загальну схему вирішення задачі синхронізації за допомогою семафорів.

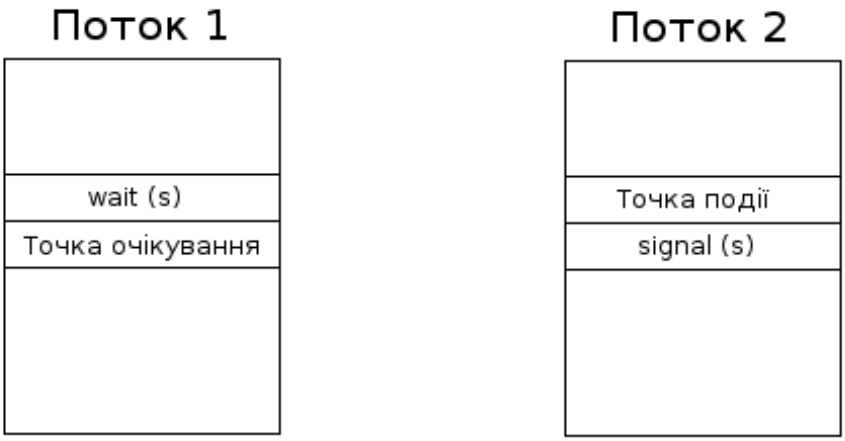

Рис 1.2. Загальна схема вирішення задачі синхронізації за допомогою семафорів.

1.2 **Семафори в мові Ада**

Семафори в мові Ада двійкові логічні, реалізовані як пакет Ada.Synchronous_Task_Control [3]. Специфікація даного пакету описує необхідні для реалізації двійкового семафора процедури [4].

Семафор являє собою об'єкт типу Suspension_Object. Установка початкового значення семафора проводиться за допомогою процедур Set_True і Set_False.

Приклад створення семафора:

```
procedure Example1 is
--Семафори
s1, s2 : Suspension_Object;
begin
       Set_True (s1); --Установка початкового значення s1
end Example1;
```

У даному випадку будуть створено два семафора - s1 і s2. Початкове значення

семафора після створення - False. Для вирішення завдання взаємного виключення початкове значення семафора повинно бути True. За допомогою процедури Set_True семафор s1 встановлюють значення True.

Операція wait (s) реалізована у вигляді процедури Suspend_Until_True. Параметром для цієї процедури є об'єкт Семафорний типу (Suspension_Object).

Операція signal (s) реалізована у вигляді процедури Set_True. Параметром для цієї процедури є об'єкт Семафорний типу (Suspension_Object).

Приклад розв'язання задачі взаємного виключення за допомогою семафора:

```
procedure Example2 is
   --Семафор
          s : Suspension_Object;
   --Общий ресурс
          a: Integer;
procedure Tasks is
   Task T1 is
   end T1;
   Task body T1 is
   a1: Integer; -- локальна копія
   begin
    Suspend_Until_True(s); --якщо значення семафора true проходимо, инакше чекаєм
     a1 := 1; -- критична ділянка
    Set_True (s); --установка значення семафора в true
   end T1;
   Task T2 is
   end T2;
   Task body T2 is
   a2: Integer; -- локальна копія
   begin
    Suspend_Until_True(s); --якщо значення семафора true проходимо, инакше чекаєм
     a2 := 1; -- критична ділянка
    Set_True (s);--установка значення семафора в true
   end T2;
begin
   null;
end Tasks;
begin
   a := 1; -- ініціація общего ресурса
   Set_True(s); -- установка значення семафора в true
   Tasks;
End Example2;
```

Висновок. Семафори в мові Ада представляють собою реалізацію

класичних двійкових семафорів. Не зручні при вирішенні завдання взаємного виключення доступу до декількох спільних ресурсів за допомогою одного семафора, при синхронізації декількох потоків. У таких випадках кількість семафорів зростає, і зростає можливість зробити помилку в логіці.

1.3 **Семафори в мові C#**

Семафори в мові C # реалізовані у вигляді класу Semaphore, що входить у простір System.Threading, який містить класи та інтерфейси для багатопотокового програмування [5]. Семафори множинні, чисельні.

Потоки виконують вхід в семафор, викликаючи метод WaitOne, успадкований від класу WaitHandle [6], і звільняють семафор викликом методу Release.

Лічильник семафору зменшується на одиницю кожного разу, коли в семафор входить потік, і збільшується на одиницю, коли потік звільняє семафор. Коли лічильник дорівнює нулю, подальші запити блокуються, поки інші потоки не звільнять семафор. Коли семафор звільнений всіма потоками, лічильник має максимальне значення, задане при створенні семафора.

Гарантований порядок, в якому б блоковані потоки входили до семафор, наприклад FIFO або LIFO, відсутня. Потік може виконувати вхід в семафор кілька разів, викликаючи багаторазово метод WaitOne. Щоб звільнити деякі з цих входів, потік може викликати перевантажений метод Release () без параметрів кілька разів, або викликати перевантажений метод Release (Int32), що вказує кількість звільняються входів. Створення семафора проводиться викликом конструктора класу Semaphore.

Формат виклику конструктора[7]:

```
public Semaphore(
        int initialCount,
        int maximumCount,
```

string name

)

Параметри

initialCount

Тип: System.Int32

Початкова кількість запитів для семафора, яке може бути забезпечено одночасно.

maximumCount

Тип: System.Int32

Максимальна кількість запитів семафора, яке може бути забезпечено одночасно.

name

Тип: System.String

Назва об'єкту іменованого системного семафора.

Клас Semaphore не забезпечує потокової ідентифікації для викликів методів WaitOne або Release. Забезпечити, щоб потоки не звільняли семафор дуже багато разів - відповідальність програміста. Припустимо, наприклад, що у семафора максимальне значення лічильника дорівнює двом, і два потоки, A і B входять до семафор. Якщо помилка програмування в потоці B змушує його викликати метод Release двічі, обидва виклику закінчуються успішно. Лічильник на семафор досяг максимального значення, і якщо потік A викличе метод Release, буде видано виключення SemaphoreFullException[8].

Семафори в C# бувають двох типів: локальні семафори і іменовані системні семафори. При створенні об'єкта Semaphore за допомогою конструктора, що дозволяє передавати параметр з ім'ям семафора, об'єкт зв'язується з семафором операційної системи, що має таке ім'я. Іменовані системні семафори доступні в межах всієї операційної системи і можуть бути використані для синхронізації дій процесів. Можна створити кілька об'єктів Semaphore, що представляють

один і той же іменований системний семафор, і використовувати метод OpenExisting для відкриття існуючого іменованого системного семафора.

Локальний семафор існує тільки всередині одного процесу. Він може використовуватися будь-якою потоком у процесі, що мають посилання на локальний об'єкт Semaphore. Кожен об'єкт Semaphore є окремим локальним семафором.

Приклад створення семафору з максимальним значенням лічильника рівним 3 та вихідним значенням лічильника рівним 0.

```
using System;
using System.Threading;

public class Example
{
    // Семафор
    private static Semaphore _pool;

    private static int _padding;

    public static void Main()
    {
        // Створення сетафору з максимальним значенням 3 та початковим значенням 0
        _pool = new Semaphore(0, 3);

        // Створення та запуск пяти потоків
        for(int i = 1; i <= 5; i++)
        {
            Thread t = new Thread(new ParameterizedThreadStart(Worker));
            t.Start(i);
        }

        // головний потік збільшує значення лічільника семафору на 3
        Console.WriteLine("Main thread calls Release(3).");
        _pool.Release(3);

        Console.WriteLine("Main thread exits.");
    }

    private static void Worker(object num)
    {
        // Кожен потік очікує на семафор
        Console.WriteLine("Thread {0} begins " +
            "and waits for the semaphore.", num);
        _pool.WaitOne();

        Console.WriteLine("Thread {0} enters the semaphore.", num);

        // Імітація роботи різними затримками
        Thread.Sleep(1000 + padding);

        Console.WriteLine("Thread {0} releases the semaphore.", num);
        Console.WriteLine("Thread {0} previous semaphore count: {1}",
            num, _pool.Release());
    }
}
```

У прикладі запускаються п'ять потоків, які очікують семафор. Головний потік використовує перевантажений метод Release (Int32) для збільшення лічильника семафора до максимального значення, дозволяючи трьом потокам

увійти в семафор. У кожному потоці використовується метод Thread.Sleep[9] для створення імітує роботу односекундной затримки, а потім викликається перевантажений метод Release () для звільнення семафора. Кожного разу при звільненні семафора відображається попереднє значення лічильника семафора.

Висновки. Семафори в мові C# представляють собою реалізацію множинних чисельних семафорів. На відміну від семафорів у мові Ада, з'явились засоби для виявлення та обробки помилок, створення та роботи з системними семафорами.

1.4 Семафори в бібліотеці Win32

Семафори в бібліотеці Win32 представлені спеціальним типом та операціями над ним [3]. Створення об'єкту данного типу виконується за допомогою функції CreateSemaphore [10]. Специфікація данної функції:

```
HANDLE CreateSemaphore(
  LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,
  LONG lInitialCount,
  LONG lMaximumCount,
  LPCTSTR lpName
);
```

Параметри, що передаються до функції:

lpSemaphoreAttributes

(in) Показчик на атрибути захисту. Встановіть у NULL.

lInitialCount

(in) Задає початкове значення семафору. Це значення має бути більше або дорівнює нулю і менше або дорівнює lMaximumCount. Семафор доступний, коли його значення більше нуля і заборонений, коли значення дорівнює нулю. Кількість зменшується на один, коли функція очікування звільнює потік, який чекав семафора. Кількість збільшується на

зазначену суму шляхом виклику функції ReleaseSemaphore.

lMaximumCount

(in) Вказує максимальне значення семафору. Це значення має бути більше нуля.

lpName

(показчик) Показчик на рядок із зазначенням найменування об'єкта семафора. Ім'я не повинно перевищувати MAX_PATH символів і може містити будь-які символи, крім символу зворотній слеш (\). Назва повинна вводитися з урахуванням регістра літер. Якщо lpName збігається з ім'ям існуючого іменованого семафора, lInitialCount і lMaximumCount параметри ігноруються, оскільки вони вже були встановлені в процесі створення.

Значення, що повертає данна функція — Handle. Handle для 64-розрядної версії Windows буде 64-бітове беззнаковое значення, тоді як для 32-розрядні версії, він буде 32-біт без знака. Що стосується Win32, то Handle просто вказівник типу void*. В Win32 Handle (як правило) покажчик на структуру, якою не можливо безпосередньо маніпулювати. Це структура, яка повинна бути передана і використовується тільки функціями системи. Наприклад, виклик CreateWindow () і повертає дескриптор вікна. Покажчик на який саме обєкт не відомо, ця інформація достувна лише операційній системі. Це "вікно структури". Якого типу, чи які поля містить данна структура не відомо. З показчиком не робиться взагалі нічого, за винятком того, що він передається в безліч інших функцій операційної системи, які вже точно знають, з чим саме вони працюють, де розташований цей обєкт, тощо [11].

Приклад створення семафору Win32 (в мові С)

```
#include <stdio.h>
#include "windows.h"
```

```
HANDLE s1; //змінна типу HANDLE, якій буде зберігатися показчик на семафор
int main() {
	s1 = CreateSemaphore (NULL, 0, 3, NULL); //Функція, що створює семафор
	return 0;
}
```

У данному прикладі створюється семафор з максимальним значенням 3, початковим значенням 0. Показчик на створений семафор зберігається у змінній s1.

Очікування семафору потоком виконується за допомогою функції WaitForSingleObject [12]. Функція WaitForSingleObject використовується для перевірки поточного стану зазначеного об'єкта. Якщо стан об'єкту є несигнальним, потік що викликає функцію входить у стан очікування, поки об'єкт не перейде у сигнальний стан або не пройде тайм-аут інтервал.

Функція змінює стан деяких типів об'єктів синхронізації. Зміна відбувається тільки над об'єктом. Наприклад, лічільник семафору зменшується на 1.

Специфікація данної функції:

```
DWORD WINAPI WaitForSingleObject(
  __in  HANDLE hHandle,
  __in  DWORD dwMilliseconds
);
```

Параметри, що передаються до функції:

hHandle

(in) Дескриптор об'єкта, у данному випадку семафору, що очікується. Якщо Handle закривають, а очікування ще не завершено, то поведінка функції не визначена. Дескриптор повинен мати SYNCHRONIZE права доступу.

dwMilliseconds

(in) Тайм-аут інтервал у мілісекундах. Якщо ненульове значення не

вказано, функція чекає, поки об'єкт сигнал або інтервал проходить. Якщо dwMilliseconds дорівнює нулю, то функція не входить в стан очікування, якщо об'єкт не визначений, вона завжди повертається негайно. Якщо dwMilliseconds нескінченна, то функція поверне тільки тоді, коли об'єкт подасть сигнал.

Значення, що повертає данна функція:

WAIT_ABANDONED (0x00000080L)

Вказаний об'єкт не був відпущений потоком, що очікується, до завершення данного потоку. Якщо об'єкть був захищеним, ви повинні перевірити його на наявність невідповідностей.

WAIT_OBJECT_0 (0x00000000L)

Стан зазначеного об'єкта сигнальний.

Wait_timeout (0x00000102L)

Тайм-аут інтервал пройшов, і стан об'єкту є несигнальним.

WAIT_FAILED ((DWORD) 0xFFFFFFFF)

Функція була перервана. Для отримання додаткової інформації про помилку викличте GetLastError [13].

Функція звільнення семафору - ReleaseSemaphore [14]. Кожного разу, коли чекаючий потік закликає ReleaseSemaphore, лічільник семафорів збільшується на 1.

Інше використання ReleaseSemaphore - при ініціалізації програми. Додаток може створити семафор з початковим значенням нуль. Це встановлює стан семафора несигнальним і блокує всі потоки. Коли додаток завершує свою ініціалізацію, він використовує ReleaseSemaphore щоб збільшити лічільник семафору до свого максимального значення, щоб дозволити нормальний доступ до захищеного ресурсу.

Специфікація данної функції:

```
BOOL ReleaseSemaphore(
    HANDLE hSemaphore,
    LONG lReleaseCount,
    LPLONG lpPreviousCount
    );
```

Параметри, що передаються до функції:

hSemaphore

(in) Дескриптор об'єкта семафора.

lReleaseCount

(in) Визначає кількість, на яку повинна бути збільшена поточне значення семафору. Значення має бути більше нуля. Якщо зазначена сума буде перевищувати максимальне значення лічильника, який був зазначений при створенні семафру, значення не змінюється і функція повертає FALSE.

lpPreviousCount

(out) Довгий покажчик на 32-бітну змінну. Цей параметр може бути NULL, якщо попереднє значення не потрібно.

Значення, що повертає данна функція:

Значення, відмінне від нуля вказує на успішне завершення. Нуль вказує на помилку при виконанні. Для отримання детальної інформації щодо помилки, потрібно використати функцію GetLastError [13].

Знищення семафору виконується за допомогою функції CloseHandle [15].

Ця функція робить недійсним значення HANDLE, а також виконує перевірку

об'єкта на який вказує HANDLE. Після того, як останній HANDLE, що вказував на об'єкт закрито, об'єкт видаляється з системи.

Специфікація данної функції:

```
BOOL CloseHandle(
  HANDLE hObject
);
```

Параметри, що передаються до функції:

hObject
(in) HANDLE, що потрібно закрити.

Значення, що повертає данна функція:

Значення, відмінне від нуля вказує на успішне завершення. Нуль вказує на помилку при виконанні. Для отримання детальної інформації щодо помилки, потрібно використати функцію GetLastError [13].

Окрім вище зазначених функцій для роботи з семафорами, в бібліотеці win32 існують розширені функції для роботи з семафорами. Вони дозволяють більш широко використовувати можливості семафорів.

Розширена функція для створення семафору CreateSemaphoreEx [15]. Данна функція аналогічна функції CreateSemaphore, але окрім можливості задати початкове значення, максимальне значення чи відкрити вже існуючий системний семафор, за допомогою цієї функції можливо задати права доступу до даного семафору.

Специфікація данної функції:

```
HANDLE WINAPI CreateSemaphoreEx(
```

```
  __in_opt    LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,
  __in        LONG lInitialCount,
  __in        LONG lMaximumCount,
  __in_opt    LPCTSTR lpName,
  __reserved  DWORD dwFlags,
  __in        DWORD dwDesiredAccess
);
```

Параметри, що передаються до функції:

lpSemaphoreAttributes

(in, optional) Покажчик на структуру SECURITY_ATTRIBUTES. Якщо цей параметр дорівнює NULL, семафор не може успадковуватися дочірніми процесами.

LinitialCount

(in) Початкове значення лічільника семафора. Це значення має бути більше або дорівнює нулю і менше або дорівнює lMaximumCount.

lMaximumCount

(in) Максимальне значення лічільника семафору. Це значення має бути більше нуля.

lpName

(in, optional) Покажчик на рядок із зазначенням найменування об'єкта семафора. Ім'я не повинна перевищувати MAX_PATH символів. Назва повинна вводитися з урахуванням регістра літер. Якщо lpName збігається з ім'ям існуючого іменованого семафора, lInitialCount і lMaximumCount параметри ігноруються, тому що вони вже встановлені. Якщо lpSemaphoreAttributes параметр не NULL, він визначає, чи дескриптор може бути успадкований. Якщо lpName дорівнює NULL, семафор

створюється без імені. Якщо lpName збігається з ім'ям існуючої події, мьютексу, waitable таймеру, потоку, або файл-відображення, функція не виконується і функція GetLastError повертає ERROR_INVALID_HANDLE. Це відбувається тому, що ці об'єкти мають той же простір імен. Ім'я може мати "Global \" або "Local \" префікс для явного створення об'єкта в глобальному або локальному просторі імен. Інша частина ім'я може містити будь-які символи, крім зворотного слеша (\). Об'єкт може бути створений в приватному просторі імен. Для отримання додаткової інформації див Об'єкт імен.

DwFlags

Цей параметр зарезервований і повинен бути рівним 0.

dwDesiredAccess

(in) Маска для доступу семафора об'єкта. Для отримання списку прав доступу (Synchronization Object Security and Access Rights [16]).

Значення, що повертає данна функція — Handle.

Розширена функція WaitForSingleObjectEx [17] визначає, чи всі критерії очікування були дотримані. Якщо критерії не дотримані, потік що викликав функцію входить у стан очікування, поки критерії умов очікування будуть дотримані або тайм-аут інтервал пройде.

Специфікація данної функції:

```
DWORD WINAPI WaitForSingleObjectEx(
  __in  HANDLE hHandle,
  __in  DWORD dwMilliseconds,
  __in  BOOL bAlertable
);
```

Параметри, що передаються до функції:

hHandle

(in) Дескриптор об'єкта. Якщо Handle закривають, а очікування ще не завершено, то поведінка функції не визначена. Дескриптор повинен мати SYNCHRONIZE права доступу.

dWMilliseconds

(in) Тайм-аут інтервал у мілісекундах. Якщо ненульове значення не вказано, функція чекає, поки об'єкт перейде в сигнальний стан, I/O процедури завершення або APC з'являться у черзі, або інтервал очікування пройде. Якщо dwMilliseconds дорівнює нулю, то функція не входить стан очікування, якщо критерії не дотримуються, вона завжди повертається негайно. Якщо dwMilliseconds нескінченна, то функція поверне тільки тоді, коли об'єкт перейде в сигнальний стан, чи I/O процедури завершення або APC з'являться в черзі.

bAlertable

Якщо цей параметр має значення ІСТИНА, і потік перебуває у стані очікування, то функція повертає, коли система приймае в чергу I/O процедуру завершення або APC, і потік виконує продцедури або функції. В іншому випадку, функція не повертається, а також процедура завершення або APC функція не виконується.

Значення, що повертає данна функція:

WAIT_ABANDONED (0x00000080L)

Вказаний об'єкт не був відпущений потоком, що очікується, до завершення данного потоку. Якщо об'єкть був захищеним, ви повинні перевірити його на наявність невідповідностей. Якщо мьютекс був стійким захисту інформації про стан, ви повинні перевірити його на

наявність невідповідностей.

WAIT_IO_COMPLETION (0x000000C0L)

Очікування закінчилося в одному або більше потоках через додання в режимі користувача асинхронних викликів процедур (APC) в черзу потоку.

WAIT_OBJECT_0 (0x00000000L)

Стан зазначеного об'єкта сигнальний.

Wait_timeout (0x00000102L)

Тайм-аут інтервал пройшов, і стан об'єкту є несигнальним.

WAIT_FAILED ((DWORD) 0xFFFFFFFF)

Функція була перервана. Для отримання додаткової інформації про помилку викличте GetLastError [13].

Висновки. Реалізація семафорів у бібліотеці Win32 дозволяє створювати та налашловувати множинні чисельні семафори. Засоби для виявлення та обробки помилок, створення та роботи з системними семафорами представлені більш широко, ніж в інших мовах.

1.5 Узагальнення

Вище були розглянуті три реалізації семафорів у різних мовах та бібліотеках. Семафори в мові Ада більш придатні для вирішення задачі взаємного виключення, у випадку коли одним семафором обмежується доступ до одного спільного ресурсу. Оскільки семафор двійковий, то задача синхронізації більше ніх двох потоків одним семафором буде надто ускладнювати программу, тому для синхронізації великої кількости потоків доцільно буде використувувати стільки семафорів, скільки потоків потрібно синхронізувати.

На відміну від семафорів у мові Ада, семафори у мові C# чисельні та можуть приймати більш ніж два значення. Це дозволяє організувати доступ до множини спільних ресурсів одним семафором, вказавши максимальним значенням семафору кількість спільних ресурсів. Задача синхронізації багатьох потоків за допомогою одного семафору в данному випадку теж вирішується. Максимальне значення семафору буде дорівнювати кількості потоків, що потрібно синхронізувати, але, на відміну від задачі взаємного виключення, початкове значення семафору повинно бути рівним нулю.

Реалізація семафорів в бібліотеці Win32 найбільш широка. Семафори множинні та чисельні, як семафори в мові C#, але кількість параметрів значно більша, що дозволяє створити семафор, що якнайкраще підходить для данної программи. Операції, що можуть проводитися над такими семафорами аналогічні, але за допомогою розширених функцій ці дії можна робити з більш точними умовами та налаштовувати системні параметри семафорів. Недоліком є те, що бібліотеку Win32 можна використовувати лише в операційній системі Windows, а программи написані на мовах C# та Ада а потім скомпільовані компілятором, що створений саме для данної операційної системи (наприклад, компілятор мови Ада для Linux — gnat [18]) будуть працювати в данній операційній системі. На данний момент існують компілятори мови Ада для наступних операційних систем [19]:

- Solaris SPARC
- GNU/Linux x86
- Windows
- OpenVMS
- DEC Alpha OSF/1
- PC Linux
- SGI IRIX
- SCO

- UnixWare
- Interactive
- MS-DOS
- Linux/Xeon
- PowerPC
- AIX (Ада 95)
- Alpha/VAX
- IBM System 370/390 (Ада 83)

Реалізації віртуальних машин для мови C# існують як для операційної системи Windows, так і для Unix-подібних систем, але все ж для меншої кількости операційних систем [20].

1.6 Висновки до розділу 1.

1) Реалізація семафорів в мові C#, Ада та бібліотеці Win32 дозволяє вирішувати задачі взаємного виключення та синхронізації, та відповідає теоретичному опису семафорів.

2) Множинні семафори мови C# та бібліотеки Win32 дозволяють більш просто реалізовувати роботу при великих кількостях потоків, що треба синхронізувати, або при наявності множини спільних ресурсів.

3) Велика кількість реалізацій компіляторів мови Ада для різних операційних систем дозволяє припустити, що семафори в мові Ада також широко використовуються.

2. Розробка програмного забезпечення для ПКС

2.1 Розробка програмного забезпечення на мові Java (структура з спільною пам'яттю)

2.1.1 Розробка паралельного математичного алгоритму

Паралельний алгоритм для данної задачі можна представити в наступному вигляді:

1) $MA_H = MB*(MC*MZ_H) — MM_H$

де:

- H = N / P
- MZ_H — H рядків матриці MZ
- MM_H — H рядків матриці MM
- MA_H — H рядків матриці MA

Спільний ресурс: MB, MC.

2.1.2 Розробка алгоритмів задач

Оскільки программне забезпечення має бути маштабованим, то алгоритми всіх задач будуть однаковими.

Всі потоки

1) Якщо tid = 0 то ввести MB, MC

2) Якщо tid = 0 то сигнал про завершення вводу задачам 1 — P-1	$S_{j,1}$ j = (1, P-1)
3) Якщо tid = P-1 то ввести MZ, MM	
4) Якщо tid = P-1 то послати сигнал про завершення вводу задачам 0 — P-2	$S_{j,2}$ j = (0, P-2)
5) Якщо tid != 0 то чекати на завершення вводу в задачі 0	$W_{0,1}$
6) Якщо tid != P-1 то чекати завершення вводу в задачі P-1	$W_{P-1,2}$
7) Копія MB_j = MB, MC_j = MC	КД j = (0, P-1)
8) Обчислення $MA_H = MBj*(MCj*MZ_H) — MM_H$	
9) Якщо tid = 0 то чекати завершення обчислень	$W_{j,3}$ j = (1, P-1)
10) Якщо tid != 0 то послати сигнал про завершення обчислень	$S_{0,3}$
11) Якщо tid = 0 то вивести MA	

2.1.3 Розробка схеми взаємодії задач

Під час виконання даного етапу була розроблена структура класу-монітора TaskControl. Він використовується для синхронізації паралельних потоків. На даному етапі визначався набір захищених елементів, що будуть знаходитись у моніторі, а також множина захищених операцій. Набір захищених елементів визначається множиною спільних ресурсів (див. паралельний математичний алгоритм) та множиною змінних, що використовуються в якості умов. Семантика захищених операцій обиралася виходячи з завдання мінімізації кількості захищених операцій.

Клас TaskControl містить поля MC, MB для зберігання відповідних спільнихресурсів, а також поля inputCount, calcCount для організації умов виконання

методів монітору. В класі містяться наступні методи:

– waitInput — для очікування введення даних в потоках T(0) та T(P-1);

– waitCalc — для очікування закінчення обчислень;

– inputMB — для введення MB;

– inputMC — для введення MC;

– copyMB — для копіювання спільного ресурсу MB;

– copyMC — для копіювання спільного ресурсу MC;

– signalInputDone — для сигналу про завершення вводу даних;

– signalCalcDone — для сигналу про завершення обчислень;

Структура данного класу описана в Додатку А, малюнок 2.1

2.1.4 Розробка програми

Програма для системи з спільною пам'яттю написана на мові Java, и реалізована у вигляді трьох класів:

– Main — основний клас. Містить головний метод, що запускається JVM при старті програми. Головний метод формує ідентифікатори потоків, запускає потоки та вимірює час їх виконання. В основному класі знаходиться константа P, змінюючи яку можна виконати налаштування програми під конкретну комп’ютерну систему;

– TaskType — задачний тип, реалізує інтерфейс Runnable;

– TaskControl — клас-монітор, який вирішує задачі синхронізації та взаємного виключення, а також зберігає спільні ресурси;

Повний лістинг програми наведено у додатку Д.

2.2 Розробка програмного забезпечення на мові Ада (структура з локальною пам'яттю)

2.2.1 Розробка паралельного математичного алгоритму

Паралельний алгоритм для данної задачі можна представити в наступному вигляді:

1) $MA_H = MB*(MC*MZ_H) — MM_H$

де:

– H = N / P

– MZ_H — H рядків матриці MZ

– MM_H — H рядків матриці MM

– MA_H — H рядків матриці MA

2.2.2 Розробка алгоритмів задач

Оскільки программне забезпечення має бути маштабованим, то алгоритми всіх задач будуть однаковими.

Всі потоки

1) Якщо tid = 0 то ввести MB, MC

2) Якщо tid = P-1 то ввести MM, MZ

3)Якщо tid > 0 та tid <= P/2 то прийняти MC, MB від задачі tid — 1

4) Якщо tid >= P/2 та tid < P - 1 то прийняти $MM_{(tid+1)H}$, $MZ_{(tid+1)H}$ від задачі tid + 1

5) Якщо tid <= P/2 то передати MC, MB в задачу tid + 1

6) Якщо tid >= P/2 то передати $MM_{(tid)H}$, $MZ_{(tid)H}$ до задачі tid - 1

7) Якщо tid < P/2 то прийняти $MM_{(tid+1)H}$, $MZ_{(tid+1)H}$ від задачі tid + 1

8) Якщо tid > P/2 то прийняти MC, MB від задачі tid — 1

9) Якщо tid < P/2 та tid > 0 то передати $MM_{(tid)H}$, $MZ_{(tid)H}$ до задачі tid — 1

10) Якщо tid > P/2 та tid < P-1 то передати MC, MB в задачу tid + 1

11) Обчислення $MA_H = MB*(MC*MZ_H) — MM_H$

12) Якщо tid = 0 то прийняти $MA_{(P-1)H}$ від задачі tid+1

13) Якщо tid > 0 та tid < P-1 то прийняти MA(P-tid)H від задачі tid+1

14) Якщо tid > 0 то послати MA(P-tid)H до задачі tid - 1

15) Якщо tid = 0 то вивести MA

2.2.3 Розробка схеми взаємодії задач

Структура паралельної обчислювальної системи з локальною пам'яттю описана в Додатку Б, малюнок 2.2

2.2.4 Розробка програми

Програма для системи з локальною пам'яттю написана на мові Ада з використанням механізму рандеву для передачі данних міх потоками.

Повний лістинг програми наведено у додатку Д.

2.3 Висновки до розділу 2

1) За допомогою засобів мови Java можливе створення багатопоточних программ.

2) Синхронізовані методи класів в мові java можуть використовуватися для вирішення задач синхронізації та взаємного виключення.

3) Структура програми з локальною пам'яттю, яка описана на малюнку 2.2 може працювати з будь-якою кількістю потоків, що перевищує 3.

4) Програму з локальною пам'яттю складніше реалізовувати через складність оптимальної структури передачі данних.

3. Тестування програмного забезпечення

3.1 Опис обчислювальної системи

Для тестування використовувалась паралельна обчислювальна система з наступним апаратним забезпеченням:

- процесор: Intel Core i5-750 (2{,}66~ГГц, 4 ядра, 8~Мб кешу третього рівня);
- оперативна пам'ять: DDR3 2 * 2048~Мб 1600~МГц;

В якості програмного забезпечення використовувались:

- операційна система: Windows Server Enterprise, 2007, Service Pack 1;
- компілятор та віртуальна машина Java: Sun Java 1.6.0 20, 64-бітна версія;
- компілятор Ада: ObjectAda Version 8.4 Special Edition.

Для виконання програми на заданій кількості ядер використовувалась програма PcExec, призначена для налаштування параметрів планувальника ОС Windows.

3.2 Тестування програмного забезпечення для структури з спільною пам'яттю

Таблиця 3.1. Час виконання програми з спільною пам'яттю (значення в мілісекундах)

N	T1	T2	T3	T4
1000	44297	23140	16828	12531
1500	215000	109297	82828	60594
2000	801266	419484	341172	219672

2500	1353625	690094	492781	367531

Підрахунок коефіцієнту прискорення (КП) виконується за формулою

$$КП = T_1 / T_P$$

Таблиця 3.2. Значення КП для програми з спільною пам'яттю

N	P			
	1	2	3	4
1000	1	1,91	2,63	3,53
1500	1	1,96	2,59	3,54
2000	1	1,91	2,34	3,64
2500	1	1,96	2,74	3,68

Підрахунок коефіцієнту ефективності (КЕ) відбувається за формулою

$$КЕ = КП / P$$

Таблиця 3.3. Значення КЕ для програми з спільною пам'яттю

N	P			
	1	2	3	4
1000	1	0,95	0,87	0,88
1500	1	0,98	0,86	0,88
2000	1	0,95	0,78	0,91
2500	1	0,98	0,91	0,92

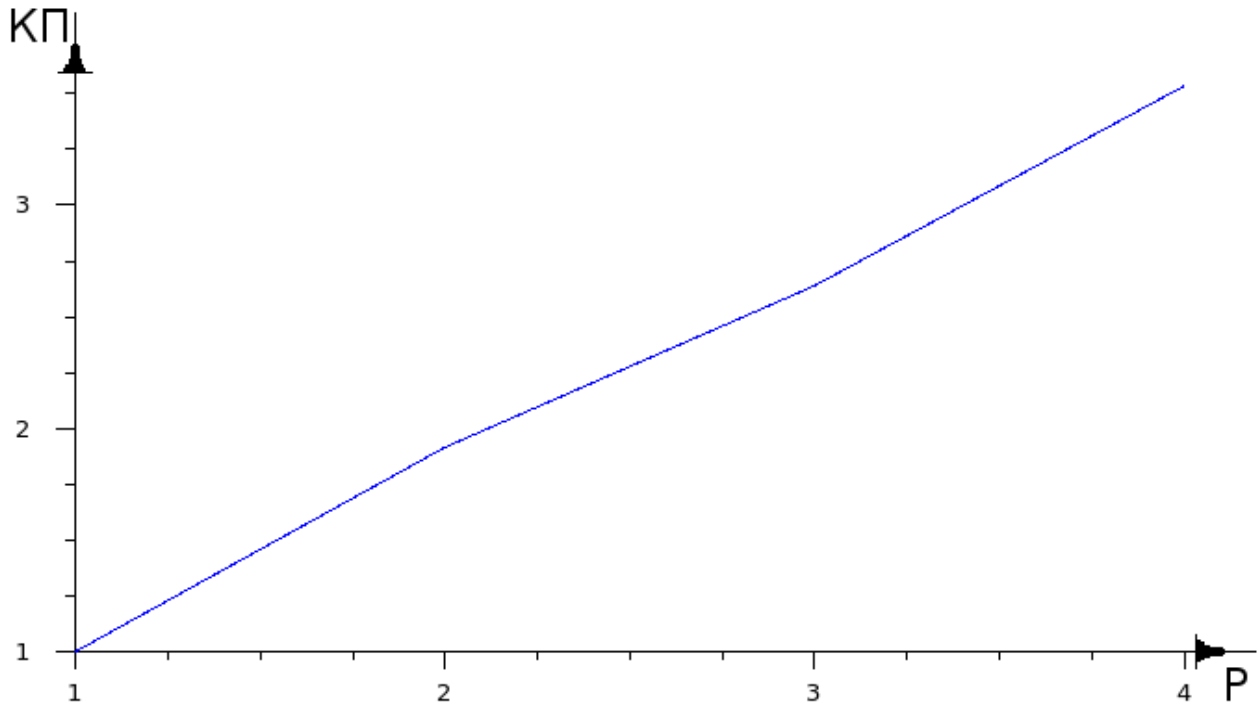

Рис 3.1. Графік залежності коефіцієнту прискорення від кількості процесорів при N = 1000

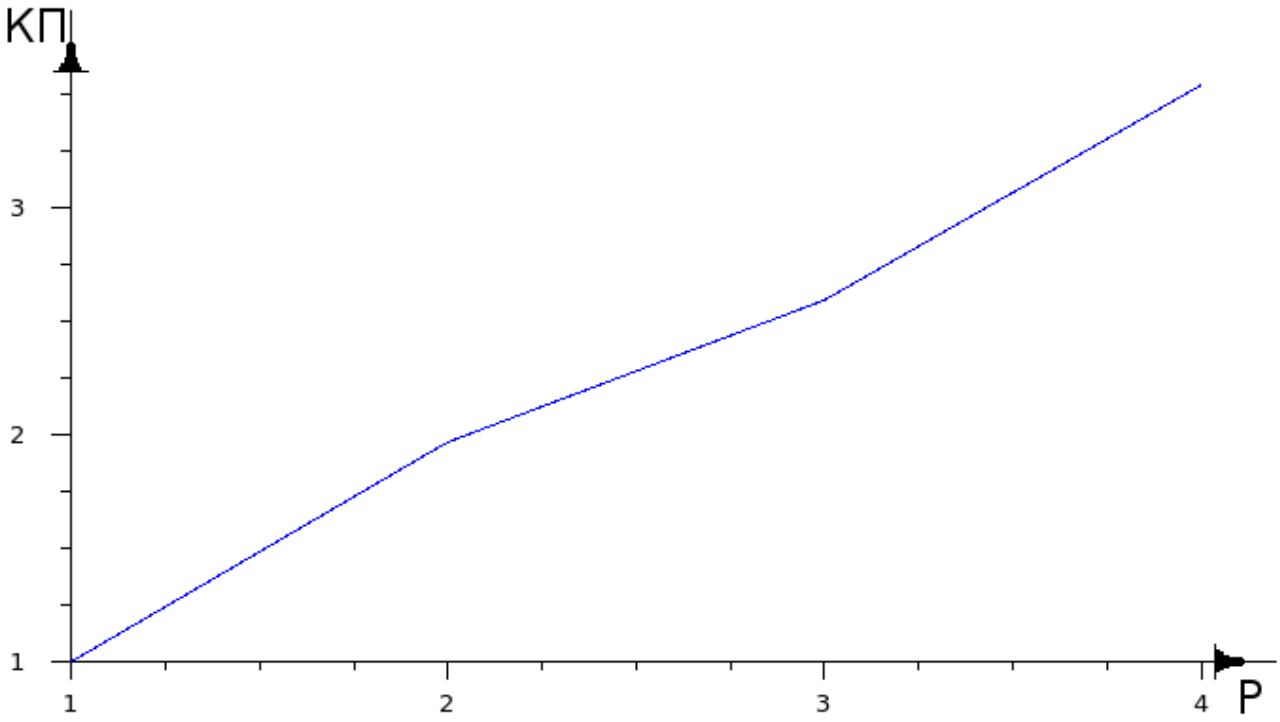

Рис 3.2. Графік залежності коефіцієнту прискорення від кількості процесорів при N = 1500

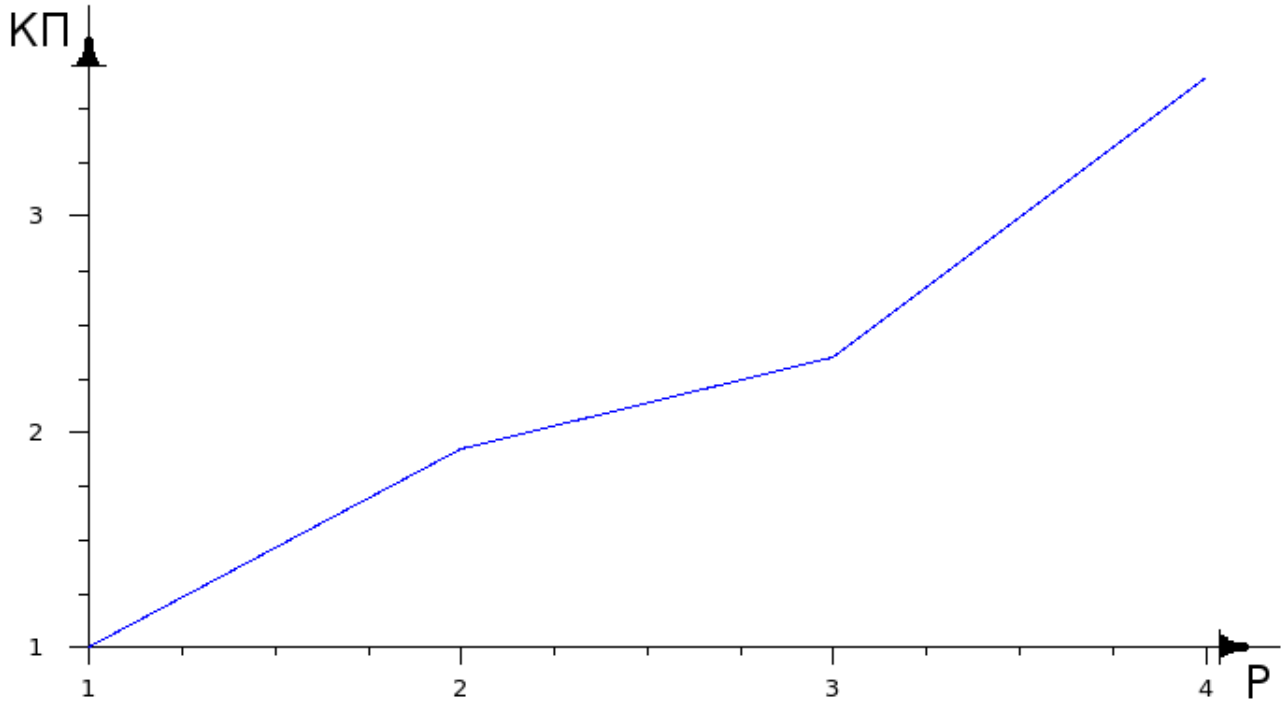

Рис 3.3. Графік залежності коефіцієнту прискорення від кількості процесорів при N = 2000

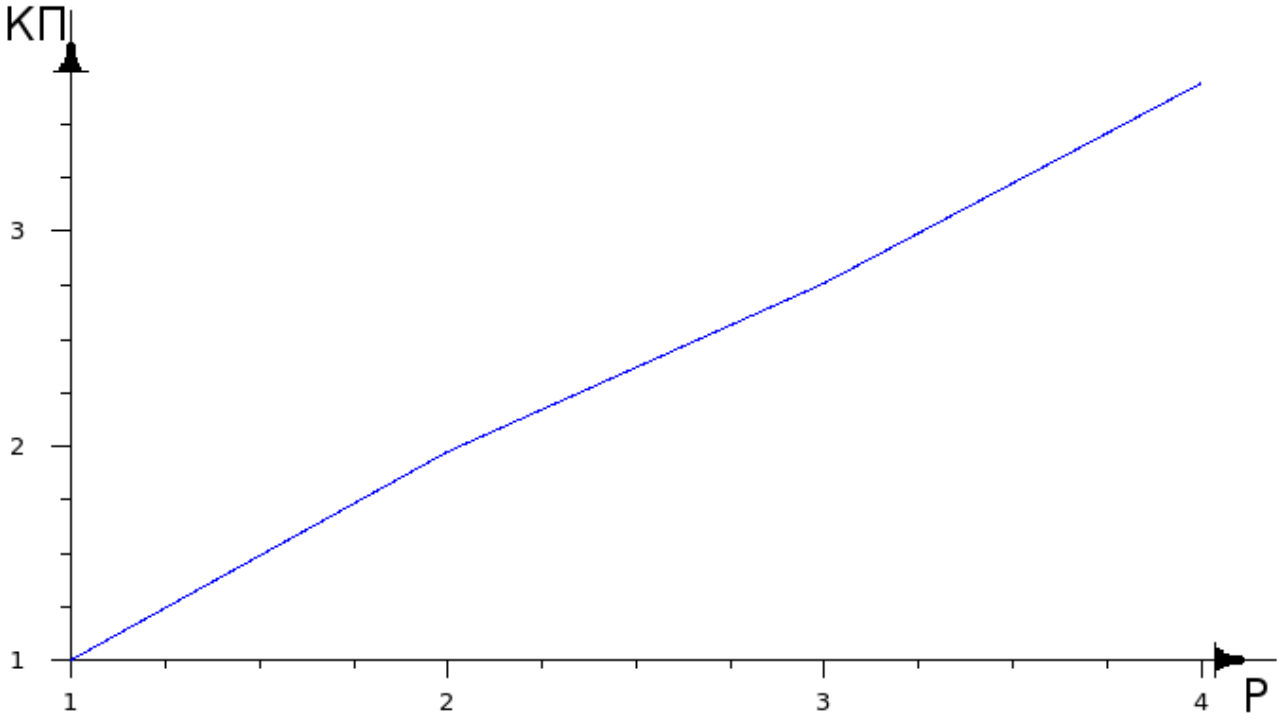

Рис 3.4. Графік залежності коефіцієнту прискорення від кількості процесорів при N = 2500

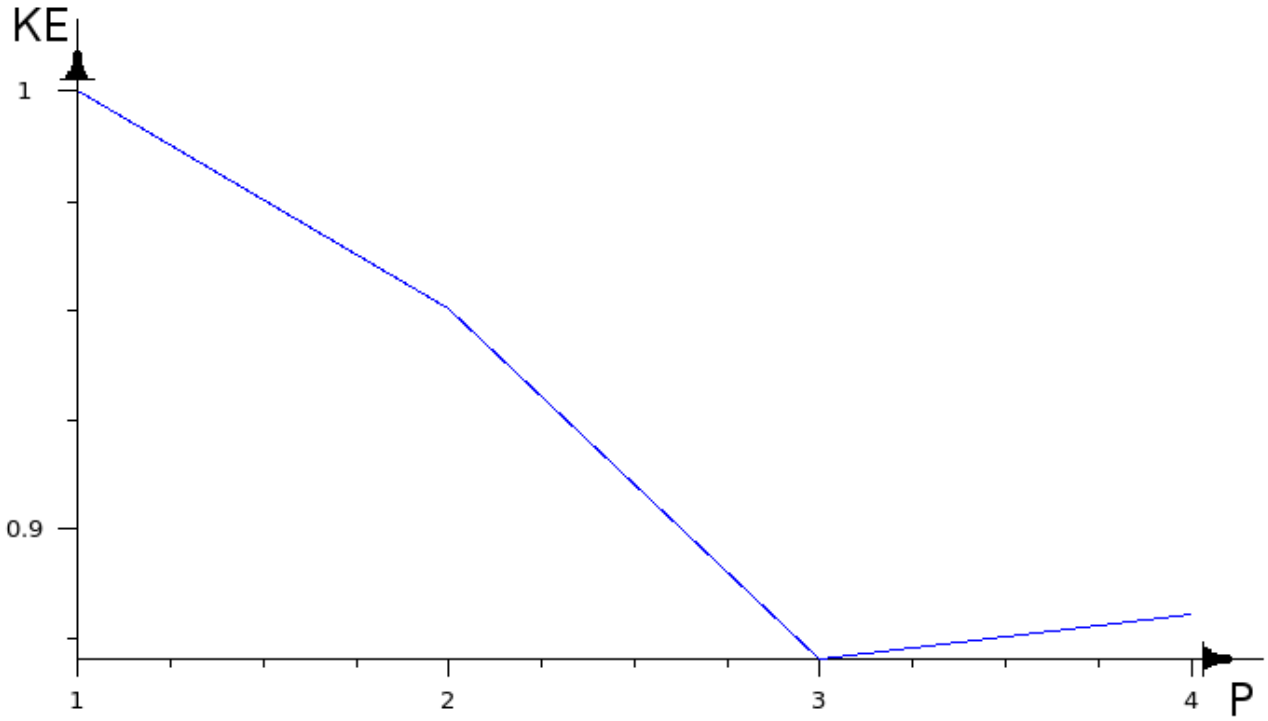

Рис 3.5. Графік залежності коефіцієнту ефективності від кількості процесорів при N = 1000

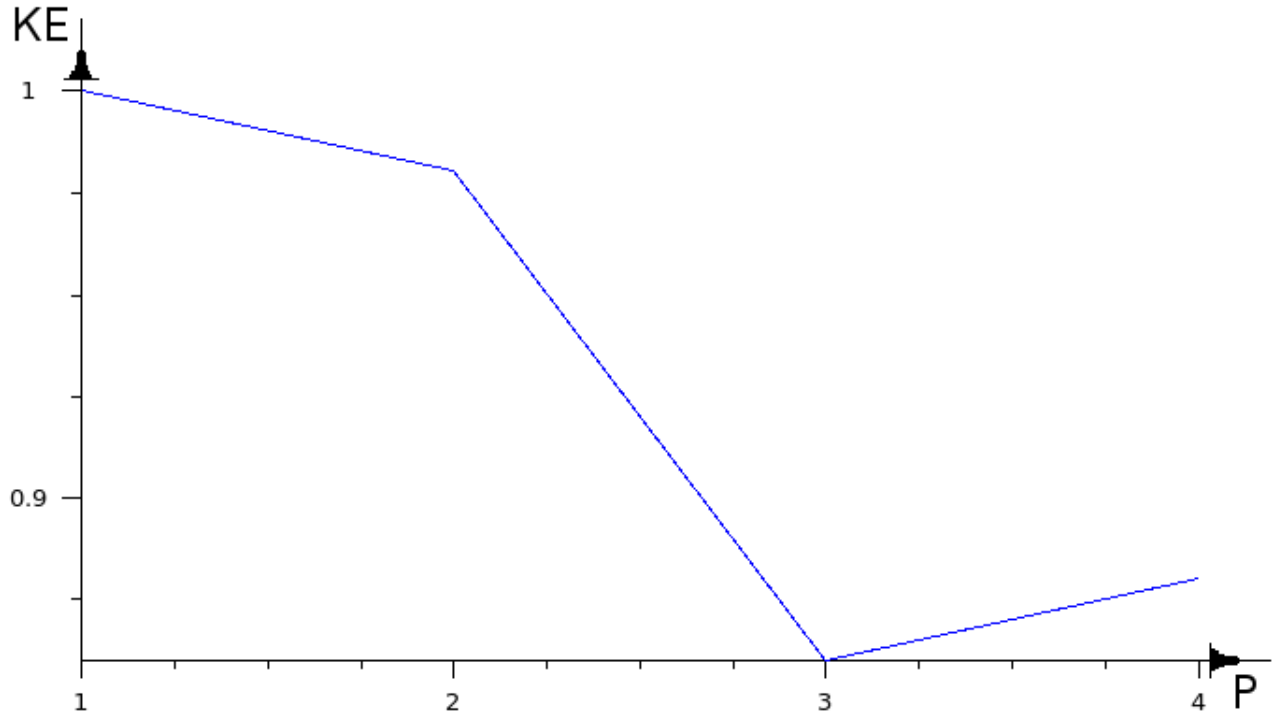

Рис 3.6. Графік залежності коефіцієнту ефективності від кількості процесорів при N = 1500

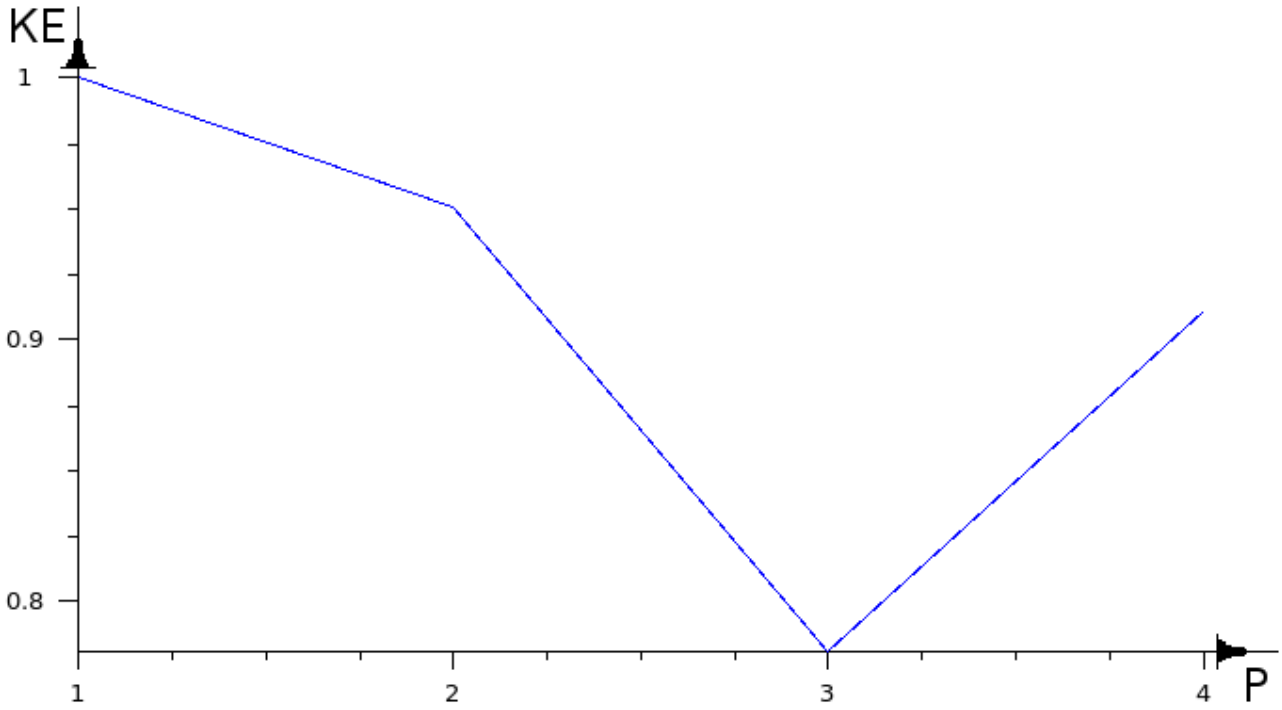

Рис 3.7. Графік залежності коефіцієнту ефективності від кількості процесорів при N = 2000

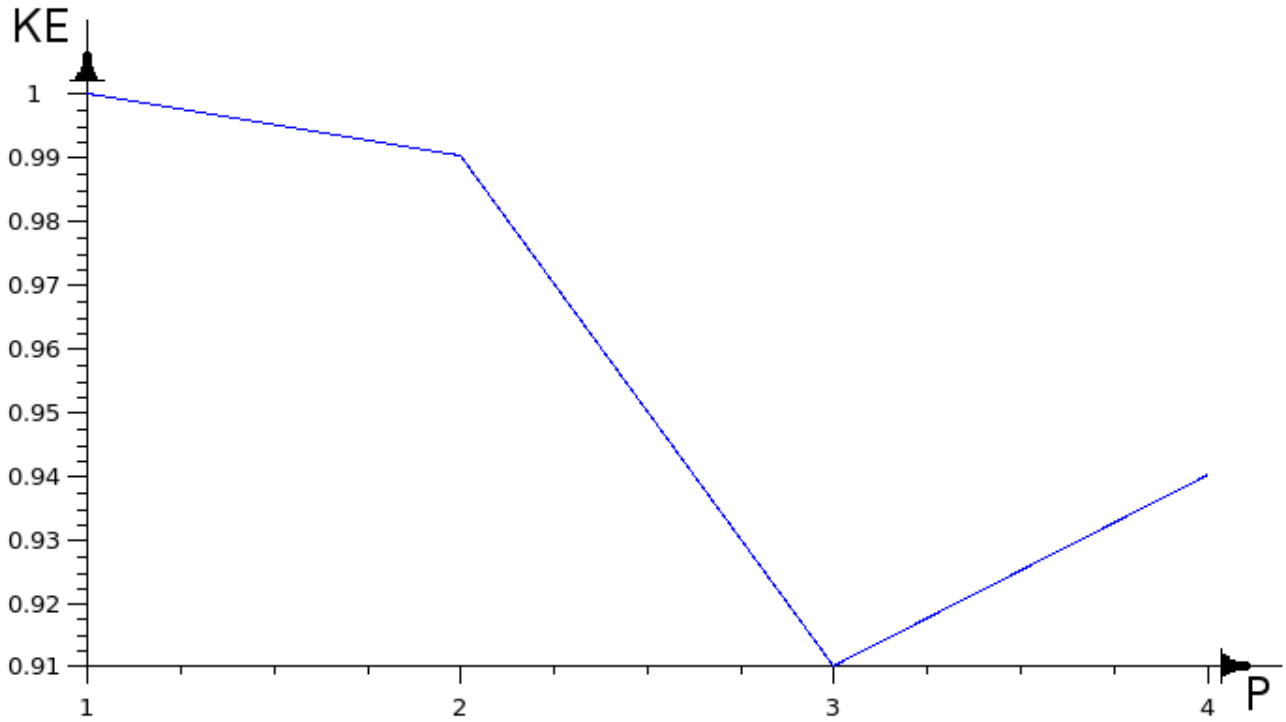

Рис 3.8. Графік залежності коефіцієнту ефективності від кількості процесорів при N = 2500

3.3 Тестування програмного забезпечення для структури з локальною пам'яттю

Таблиця 3.4. Час виконання програми з локальною пам'яттю (значення в секундах)

N	T1	T2	T3	T4
1000	77	43	34	24
1500	362	189	144	98
2000	955	487	366	260
2500	1909	962	693	507

Підрахунок коефіцієнту прискорення (КП) виконується за формулою

$$КП = T_1 / T_P$$

Таблиця 3.5. Значення КП для програми з локальною пам'яттю

N	P			
	1	2	3	4
1000	1	1,79	2,26	3,2
1500	1	1,94	2,51	3,36
2000	1	1,96	2,6	3,67
2500	1	1,98	2,75	3,76

Підрахунок коефіцієнту ефективності (КЕ) відбувається за формулою

$$КЕ = КП / P$$

Таблиця 3.6. Значення КЕ для програми з локальною пам'яттю

N	P			
	1	2	3	4
1000	1	0,89	0,75	0,8
1500	1	0,97	0,83	0,92
2000	1	0,98	0,86	0,91
2500	1	0,99	0,91	0,94

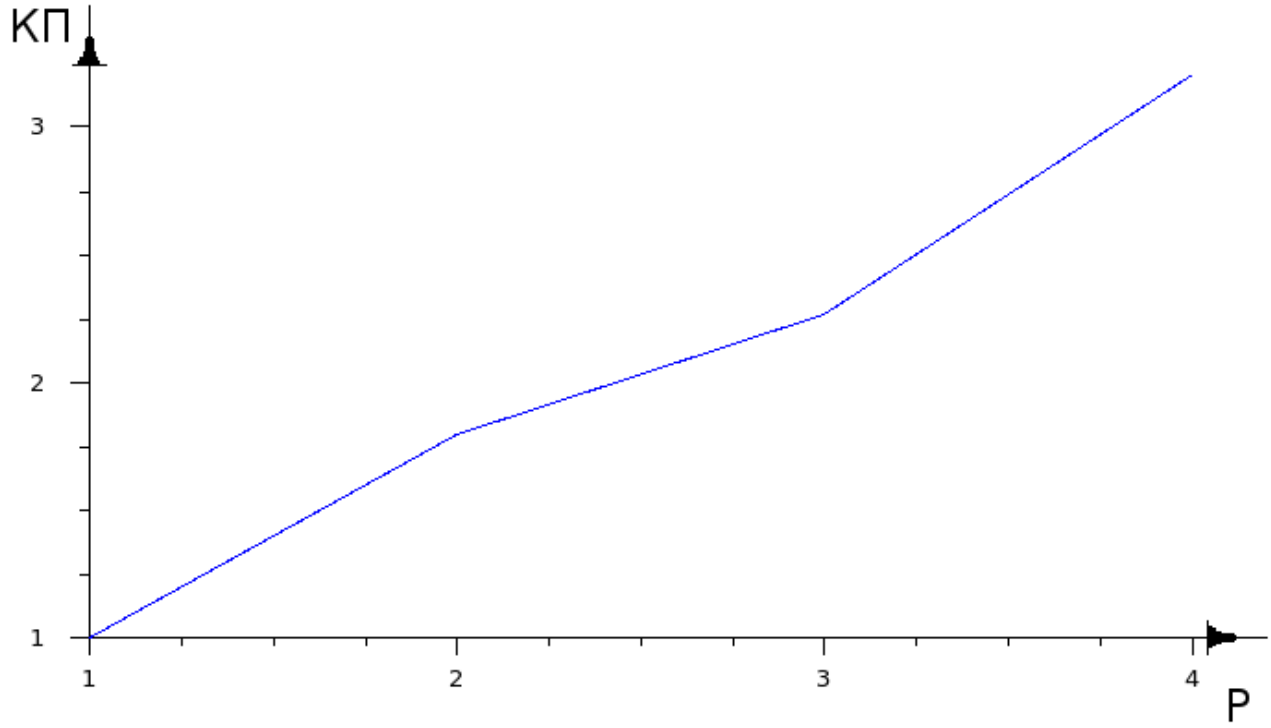

Рис 3.9. Графік залежності коефіцієнту прискорення від кількості процесорів при N = 1000

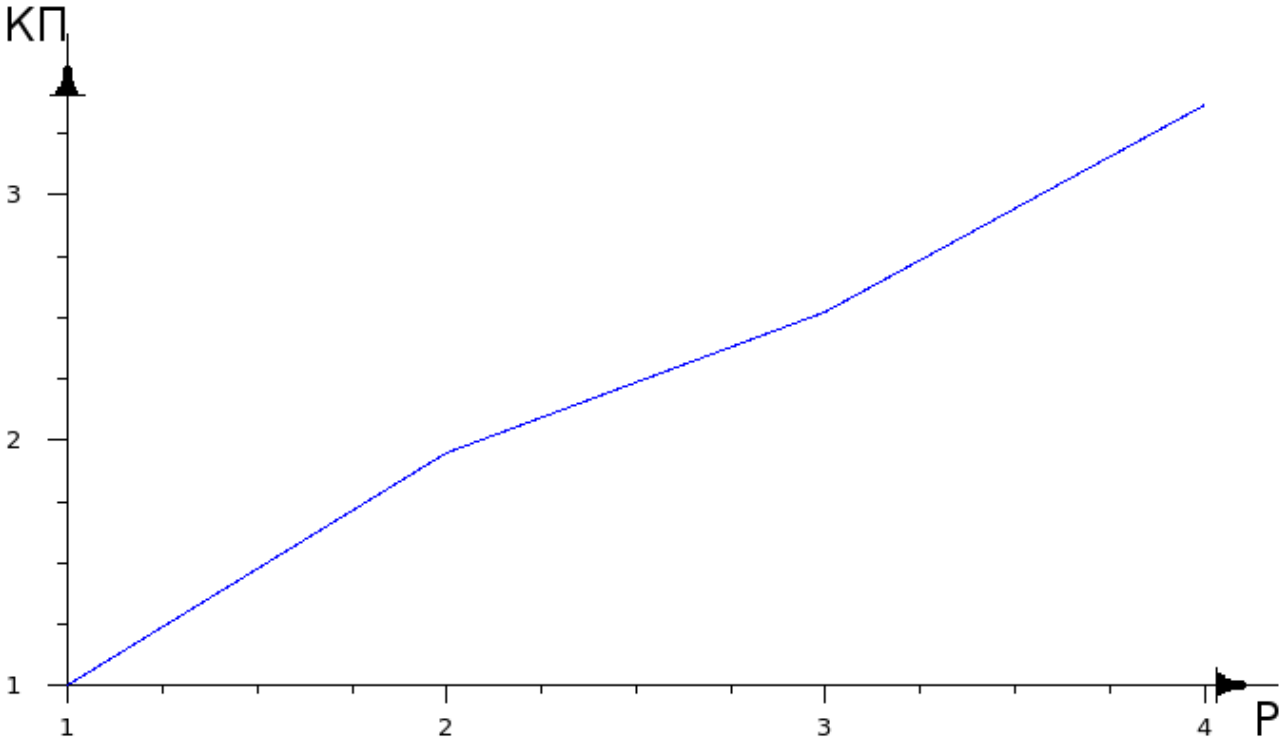

Рис 3.10. Графік залежності коефіцієнту прискорення від кількості процесорів при N = 1500

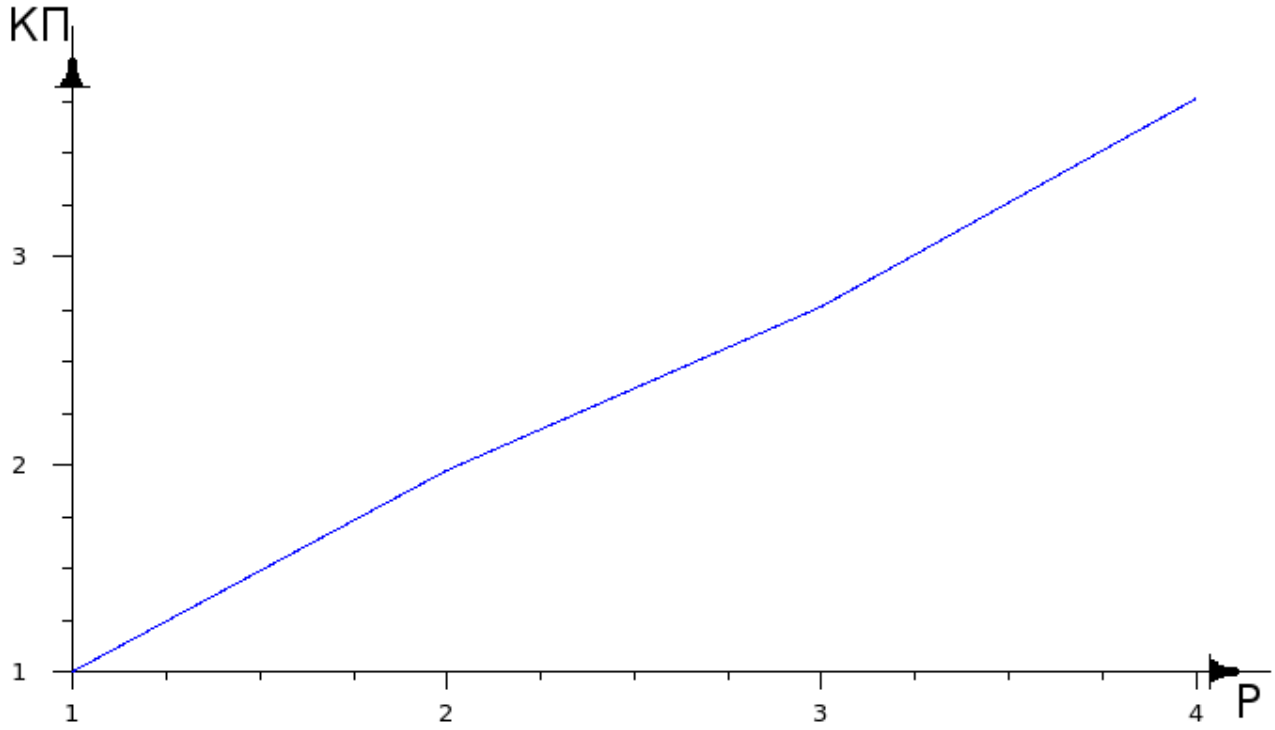

Рис 3.11. Графік залежності коефіцієнту прискорення від кількості процесорів при N = 2000

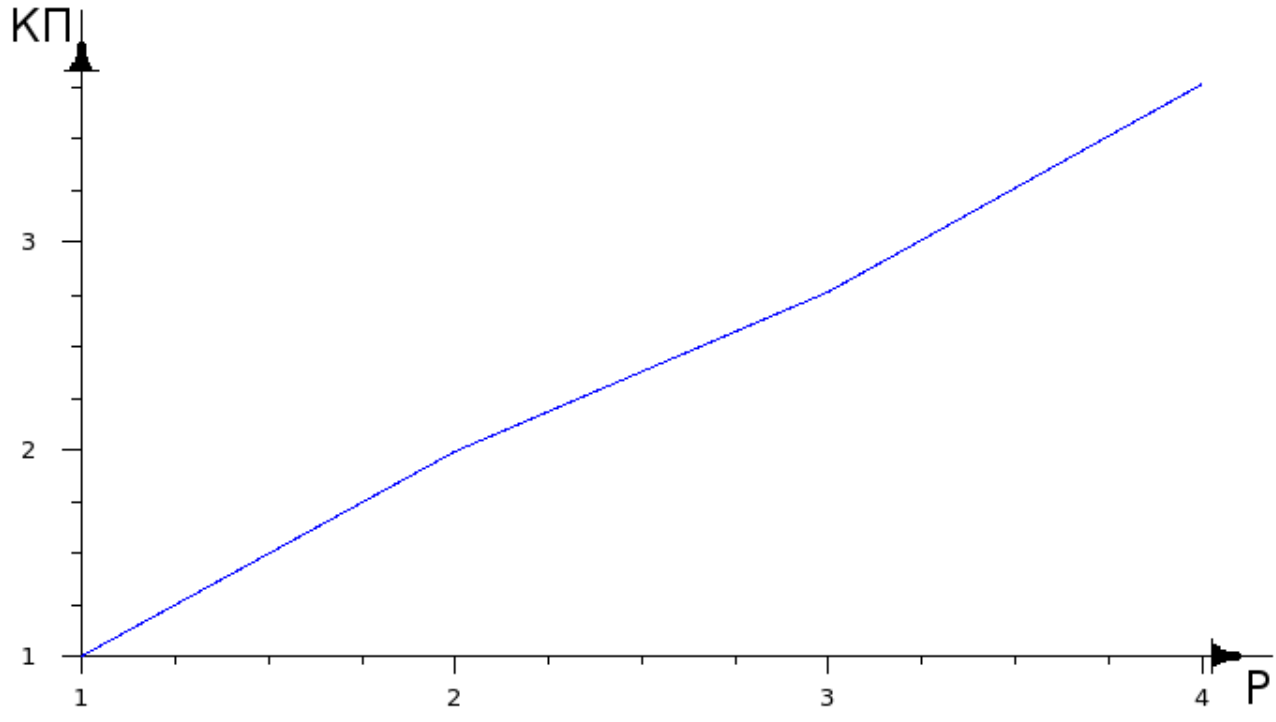

Рис 3.12. Графік залежності коефіцієнту прискорення від кількості процесорів при N = 2500

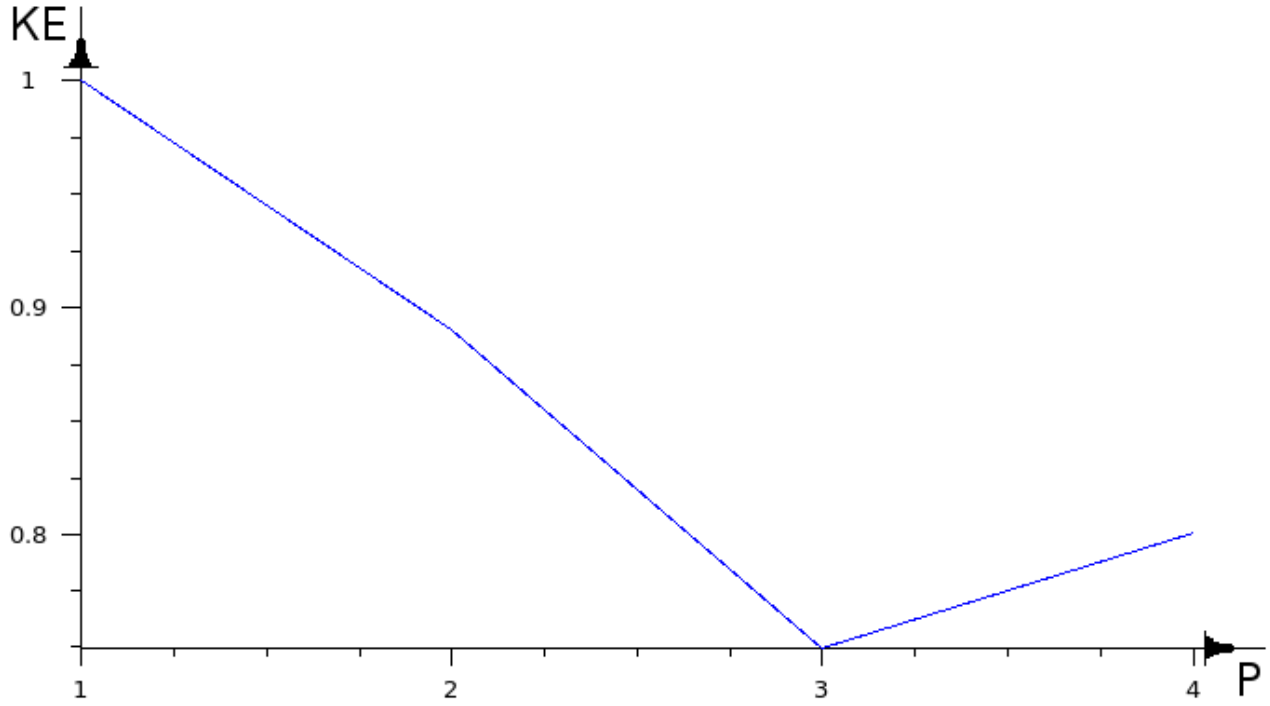

Рис 3.13. Графік залежності коефіцієнту ефективності від кількості процесорів при N = 1000

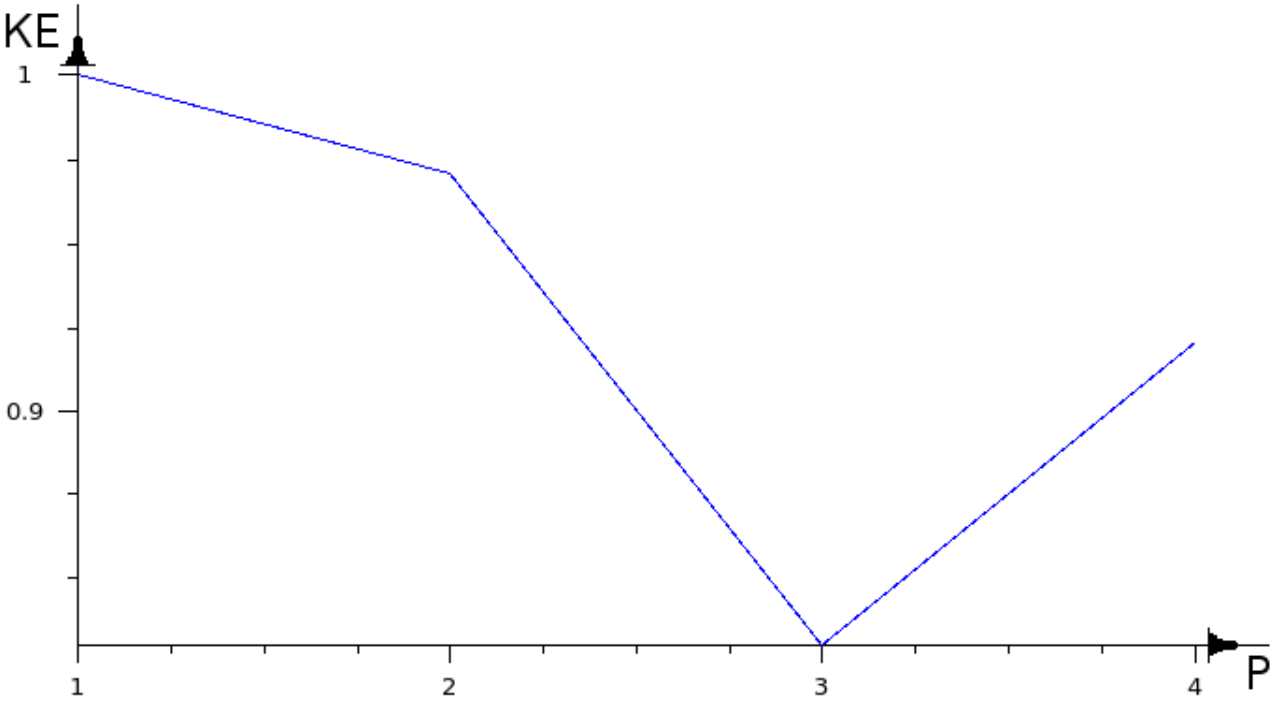

Рис 3.14. Графік залежності коефіцієнту ефективності від кількості процесорів при N = 1500

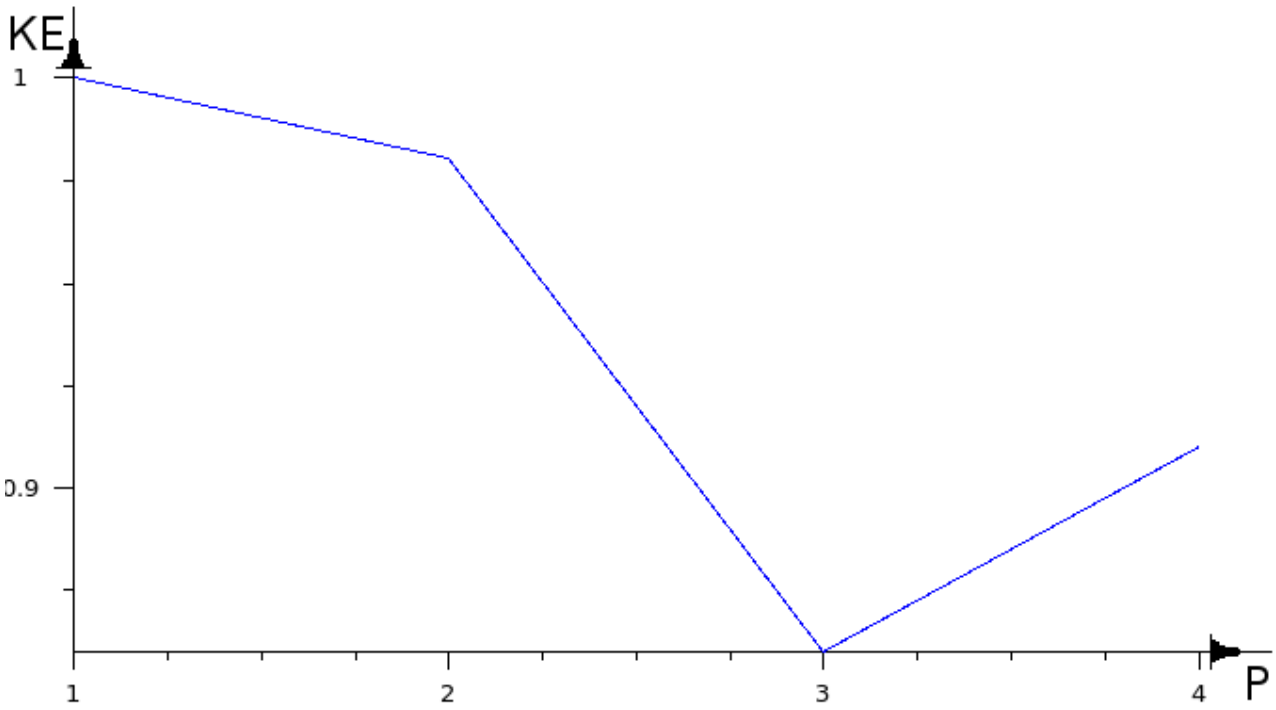

Рис 3.15. Графік залежності коефіцієнту ефективності від кількості процесорів при N = 2000

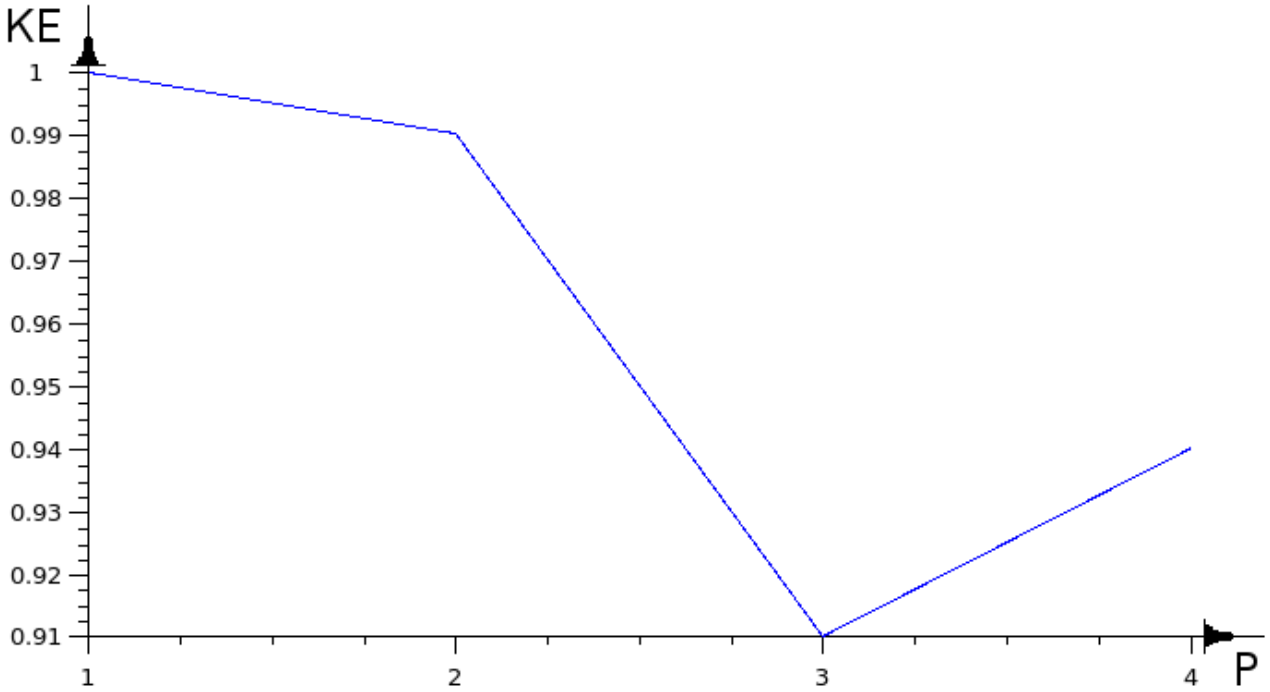

Рис 3.16. Графік залежності коефіцієнту ефективності від кількості процесорів при N = 2500

3.4 Висновки до розділу 3

Додаток А. Структура класу TaskControl

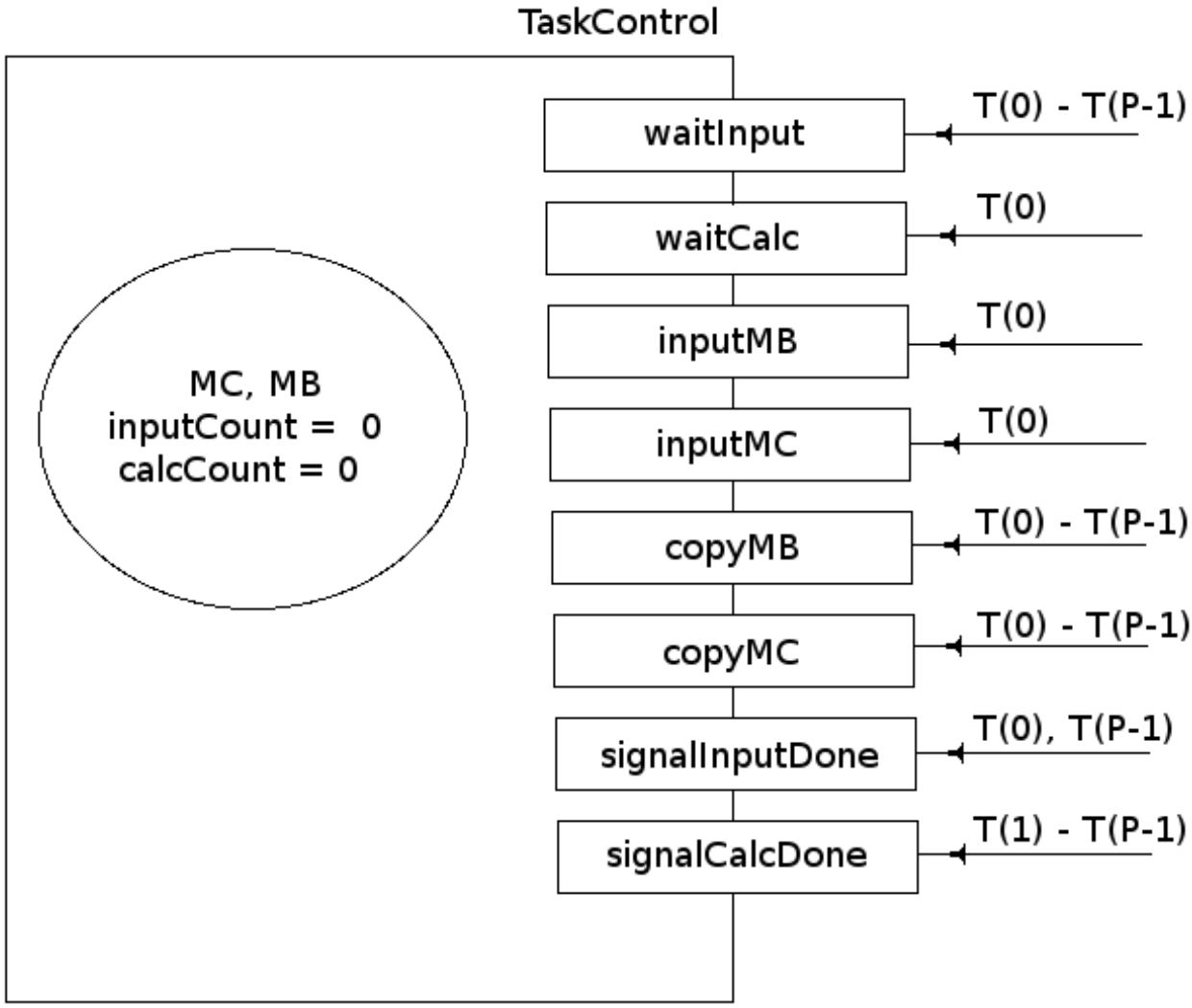

Рис 2.1. Структура класу TaskControl

Додаток Б. Структура паралельної обчислювальної системи з локальною пам'яттю

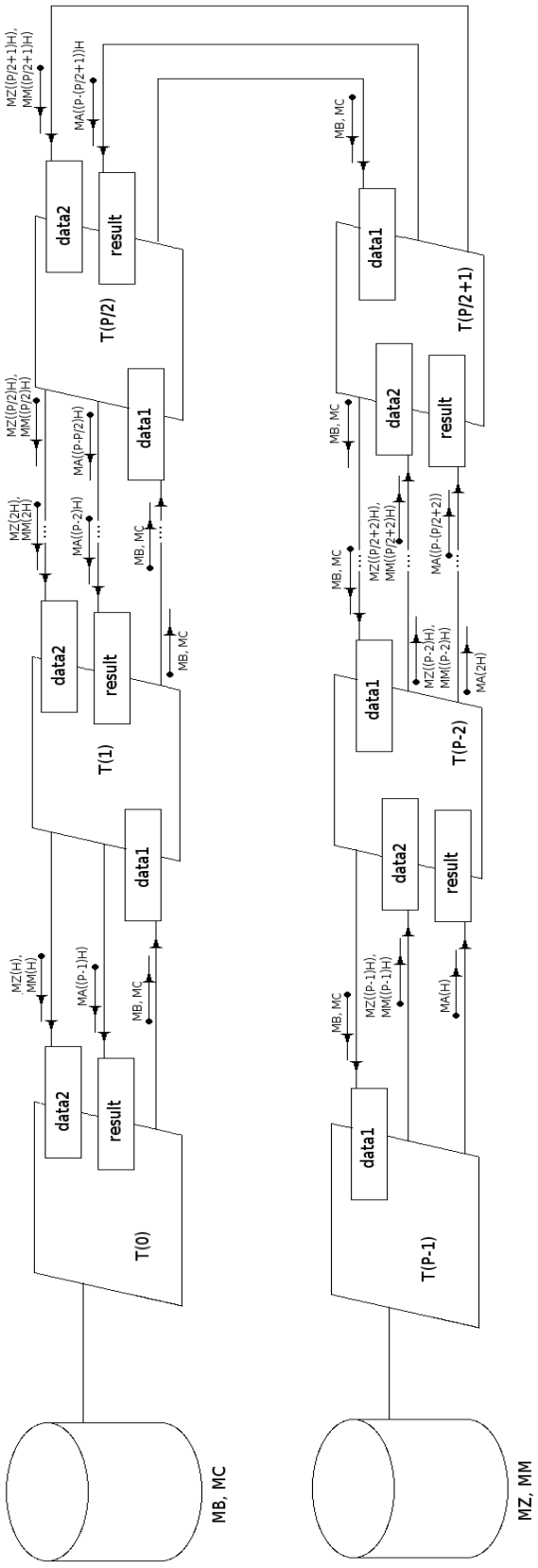

Рис 2.2. Структура паралельної обчислювальної системи з локальною пам'яттю.

Додаток В

Додаток Г

Додаток Д. Лістинги

Лістинг програми для ПОС зі спільною пам'яттю

```
ObjecAda V8.4
Copyright 1992-2008, Free Software Foundation, Inc.


Compiling: pro_couser.adb (source file time stamp: 2010-05-16 17:53:07)

     1. with Ada.Synchronous_Task_Control, Ada.Text_IO, Ada.Integer_Text_IO, Ada.Calendar;
     2. use Ada.Synchronous_Task_Control, Ada.Text_IO, Ada.Integer_Text_IO, Ada.Calendar;
     3.
     4. procedure Pro_couser is
     5.        P: integer := 5;
     6.        N: integer := 10000;
     7.        H: integer := N / P;
     8.        startTime, endTime: Time;
     9.         workTime: Duration;
    10.
    11.
    12.
    13.   type Vector is array ( Integer range<>) of Integer ;
    14.   type Matrix_Base is array (Integer range <>) of Vector (1..N);
    15.   type Matrix is array (1..N) of Vector (1 .. N);
    16.
    17. procedure startTasks is
    18.   task type T (tid:Integer) is
    19.        entry data1 (MBN : in Matrix_Base; MCN: in Matrix_Base);
    20.        entry data2 (MZN : in Matrix_Base; MMN: in Matrix_Base);
    21.        entry result (MAN: in Matrix_Base);
    22.   end T;
    23.
    24.        type TTP is access T;
    25.        tasks :array (1..P) of TTP;
    26.
    27.   task body T is
    28.        MC, MB: Matrix_Base (1..N);
    29.        MZ, MM: Matrix_Base (1..(tid+1)*H);
    30.        MA: Matrix_Base (1..(P-tid)*H);
    31.        MT: Matrix_Base (1..N);
    32.        tmp, l: integer;
    33.   begin
    34.        delay 0.5;
    35.        if (tid = 0) then
    36.        for i in 1 .. N loop
```

```
37.               for j in 1 .. N loop
38.                      MC(i)(j) := 1;
39.                      MB(i)(j) := 1;
40.               end loop;
41.       end loop;
42.       end if;
43.       if tid = P-1 then
44.       for i in 1 .. N loop
45.                 for j in 1 .. N loop
46.                         MZ(i)(j) := 1;
47.                         MM(i)(j) := 1;
48.                 end loop;
49.        end loop;
50.       end if;
51.
52.       if tid <= P/2 then
53.       if tid /= 0 then
54.               accept data1 (MBN: in Matrix_Base; MCN: in Matrix_Base) do
55.                      MB := MBN;
56.                      MC := MCN;
57.               end;
58.       end if;
59.       end if;
60.
61.       if tid >= P/2 then
62.        if tid /= P-1 then
63.                 accept data2 (MZN: in Matrix_Base; MMN: in Matrix_Base) do
64.                              MZ := MZN;
65.                               MM := MMN;
66.               end;
67.        end if;
68.        end if;
69.
70.       if tid <= P/2 then
71.               tasks(tid+2).data1 (MB, MC);
72.       end if;
73.
74.        if tid >= P/2 then
75.                 tasks(tid).data2 (MZ(1..tid*H), MM(1..tid*H));
76.          end if;
77.
78.
79.       if tid < P/2 then
80.               accept data2(MZN: in Matrix_Base; MMN: in Matrix_Base) do
81.                         MZ := MZN;
82.                         MM := MMN;
83.                 end;
84.       end if;
85.
86.
```

```
87.        if tid > P/2 then
88.                 accept data1 (MBN: in Matrix_Base; MCN: in Matrix_Base) do
89.                         MB := MBN;
90.                         MC := MCN;
91.                 end;
92.         end if;
93.
94.
95.          if tid < P/2 then
96.         if tid /= 0 then
97.                 tasks(tid).data2 (MZ(1..H*tid), MM(1..H*tid));
98.          end if;
99.         end if;
100.
101.        if tid > P/2 then
102.        if tid /= P-1 then
103.                tasks(tid+2).data1 (MB, MC);
104.        end if;
105.         end if;
106.
107.
108.       for i in tid*H+1 .. (tid +1)*H loop
109.              tmp := 0;
110.              for j in 1..N loop
111.                     for k in 1..N loop
112.                            tmp := tmp + MC(k)(j) * MZ(i)(k);
113.                     end loop;
114.                     MT(i)(j) := tmp;
115.                     tmp := 0;
116.             end loop;
117.       end loop;
118.       l := 1;
119.       for i in tid *H +1..(tid + 1)* H loop
120.             tmp := 0;
121.             for j in 1..N loop
122.                    for k in 1..N loop
123.                           tmp := tmp + MB(k)(j) * MT(i)(k);
124.                    end loop;
125.                    tmp := tmp - MM(i)(j);
126.                    MA(l)(j) := tmp;
127.                    tmp := 0;
128.             end loop;
129.             L := l + 1;
130.       end loop;
131.
132.       if tid /=  P-1 then
133.              accept result (MAN: in Matrix_Base) do
134.                     for i in 1 .. MAN'Last loop
135.                            for j in 1..N loop
136.                                   MA(i+H)(j) := MAN(i)(j);
```

```
137.                            end loop;
138.                     end loop;
139.              end;
140.       end if;
141.
142.       if tid > 0 then
143.              tasks(tid).result (MA);
144.       end if;
145.
146.       if tid = 0 then
147.              if N < 10 then
148.                     for i in MA'First .. MA'Last loop
149.                            for j in 1 .. N loop
150.                                   put (MA(i)(j));
151.                            end loop;
152.                            put_line("");
153.                     end loop;
154.              end if;
155.              endTime := Clock;
156.              workTime := endTime - startTime;
157.              put("Execution Time:");
158.                put(integer(workTime));
159.       end if;
160.   end T;
161.
162.       begin
163.              for i in 1..P loop
164.                 tasks(i) := new T(i-1);
165.                end loop;
166.       end startTasks;
167.
168.   begin
169.       New_Line;
170.        Put_Line("Vvedite N");
171.        New_Line;
172.        Get(N);
173.        Put_Line("Vvedite kol-vo procesorov");
174.        New_Line;
175.        Get(P);
176.       H:=N/P;
177.       startTime:=Clock;
178.       startTasks;
179.   end Pro_couser;

179 lines: No errors
```

Лістинг програми для ПОС з локальною пам'яттю

```
javac 1.6.78
Copyright (C) 2009 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Main.java

```
  1. import java.io.BufferedReader;
  2. import java.io.IOException;
  3. import java.io.InputStreamReader;
  4.
  5. import javax.xml.crypto.Data;
  6.
  7. public class Main {
  8.    public static int N;
  9.    public static int P;
 10.    public static int H;
 11.    public static int[][] MZ;
 12.    public static int[][] MM;
 13.    public static int[][] MA;
 14.    public static long startTime;
 15.    public static long stopTime;
 16.
 17.    public static void main(String[] args) {
 18.            String inString = "";
 19.            BufferedReader consoleIn = new BufferedReader(new InputStreamReader(
 20.                            System.in));
 21.            System.out.print("Enter number of tasks: ");
 22.            try {
 23.                    inString = consoleIn.readLine();
 24.            } catch (IOException e) {
 25.                    e.printStackTrace();
 26.            }
 27.            try {
 28.                    P = Integer.parseInt(inString);
 29.            } catch (NumberFormatException e) {
 30.                    e.printStackTrace();
 31.                    return;
 32.            }
 33.            System.out.print("Enter Matrix dimension: ");
 34.            try {
 35.                    inString = consoleIn.readLine();
 36.            } catch (IOException e) {
 37.                    e.printStackTrace();
 38.            }
 39.            try {
 40.                    N = Integer.parseInt(inString);
 41.            } catch (NumberFormatException e) {
 42.                    e.printStackTrace();
```

```
 43.                 return;
 44.           }
 45.
 46.           MA = new int [N][N];
 47.           H = N / P;
 48.           Thread [] tread = new Thread[P];
 49.           TaskType[] task = new TaskType[P];
 50.           TaskControl monitor = new TaskControl();
 51.
 52.           startTime = System.currentTimeMillis();
 53.           for (int i = 0; i < P; i++) {
 54.                 task[i] = new TaskType(i, monitor);
 55.                 tread[i] = new Thread (task[i]);
 56.                 tread[i].start();
 57.           }
 58.    }
 59. }
-e No errors.


javac 1.6.78
Copyright (C) 2009 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

TaskControl.java
  1. public class TaskControl {
  2.    private int[][] MC;
  3.    private int[][] MB;
  4.    private int inputCount = 0;
  5.    private int calcCount = 0;
  6.
  7.    public synchronized void waitInput () {
  8.           if (inputCount < 2)
  9.                 try {
 10.                       wait();
 11.                 } catch (InterruptedException e) {
 12.                       e.printStackTrace();
 13.                 }
 14.    }
 15.
 16.    public synchronized void waitCalc () {
 17.           if (calcCount < Main.P - 1)
 18.                 try {
 19.                       wait();
 20.                 } catch (InterruptedException e) {
 21.                       e.printStackTrace();
 22.                 }
 23.    }
 24.
```

```
25.    public synchronized void inputMB (int[][] MB) {
26.            this.MB = new int [MB.length][MB[0].length];
27.            for (int i = 0; i < MB.length; i++)
28.                    for (int j = 0; j < MB.length; j++)
29.                            this.MB[i][j] = MB[i][j];
30.    }
31.
32.    public synchronized void inputMC (int[][] MC) {
33.            this.MC = new int [MC.length][MC[0].length];
34.            for (int i = 0; i < MC.length; i++)
35.                    for (int j = 0; j < MC.length; j++)
36.                            this.MC[i][j] = MC[i][j];
37.    }
38.
39.    public synchronized int[][] copyMB () {
40.            int[][] result = new int [this.MB.length][this.MB[0].length];
41.            for (int i = 0; i < this.MB.length; i++)
42.                    for (int j = 0; j < this.MB.length; j++)
43.                            result[i][j] = this.MB[i][j];
44.            return result;
45.    }
46.
47.    public synchronized int[][] copyMC () {
48.            int[][] result = new int [this.MC.length][this.MC[0].length];
49.            for (int i = 0; i < this.MC.length; i++)
50.                    for (int j = 0; j < this.MC.length; j++)
51.                            result[i][j] = this.MC[i][j];
52.            return result;
53.    }
54.
55.    public synchronized void signalInputDone () {
56.            this.inputCount++;
57.            if (inputCount == 2)
58.                    notifyAll();
59.    }
60.
61.    public synchronized void signalCalcDone () {
62.            this.calcCount++;
63.            if (calcCount == Main.P - 1)
64.                    notifyAll();
65.    }
66.
67. }
-e No errors.


javac 1.6.78
Copyright (C) 2009 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

TaskType.java

```
 1. public class TaskType implements Runnable {
 2.
 3.    private int tid;
 4.    private int[][] MC;
 5.    private int[][] MB;
 6.    private TaskControl monitor;
 7.
 8.    public TaskType(int tid, TaskControl monitor) {
 9.           this.tid = tid;
10.           this.monitor = monitor;
11.           this.MB = new int[Main.N][Main.N];
12.           this.MC = new int[Main.N][Main.N];
13.    }
14.
15.    @Override
16.    public void run() {
17.           if (this.tid == 0) {
18.                  int[][] tmpMC = new int[Main.N][Main.N];
19.                  int[][] tmpMB = new int[Main.N][Main.N];
20.                  for (int i = 0; i < Main.N; i++)
21.                         for (int j = 0; j < Main.N; j++) {
22.                                tmpMC[i][j] = 1;
23.                                tmpMB[i][j] = 1;
24.                         }
25.                  monitor.inputMB(tmpMB);
26.                  monitor.inputMC(tmpMC);
27.                  monitor.signalInputDone();
28.           }
29.
30.           if (this.tid == Main.P - 1) {
31.                  Main.MZ = new int[Main.N][Main.N];
32.                  Main.MM = new int[Main.N][Main.N];
33.                  for (int i = 0; i < Main.N; i++)
34.                         for (int j = 0; j < Main.N; j++) {
35.                                Main.MZ[i][j] = 1;
36.                                Main.MM[i][j] = 1;
37.                         }
38.                  monitor.signalInputDone();
39.           }
40.
41.           monitor.waitInput();
42.
43.           this.MB = monitor.copyMB();
44.           this.MC = monitor.copyMC();
45.
46.           int[][] MD = new int[Main.N][Main.N];
47.           for (int i = this.tid * Main.H; i < (this.tid +1)* Main.H; i++) {
48.                  int tmp = 0;
```

```
 49.                 for (int j = 0; j < Main.N; j++) {
 50.                         for (int k = 0; k < Main.N; k++) {
 51.                                 tmp = tmp + MC[k][j] * Main.MZ[i][k];
 52.                         }
 53.                         MD[i][j] = tmp;
 54.                         tmp = 0;
 55.                 }
 56.           }
 57.           for (int i = this.tid * Main.H; i < (this.tid + 1)* Main.H; i++) {
 58.                 int tmp = 0;
 59.                 for (int j = 0; j < Main.N; j++) {
 60.                         for (int k = 0; k < Main.N; k++) {
 61.                                 tmp = tmp + MB[k][j] * MD[i][k];
 62.                         }
 63.                         tmp = tmp - Main.MM[i][j];
 64.                         Main.MA[i][j] = tmp;
 65.                         tmp = 0;
 66.                 }
 67.           }
 68.           if (this.tid == 0)
 69.                 monitor.waitCalc();
 70.           else
 71.                 monitor.signalCalcDone();
 72.           if (this.tid == 0 )
 73.                 if (Main.N < 11){
 74.                         for (int i = 0; i < Main.N; i++) {
 75.                                 for (int j = 0; j < Main.N; j++)
 76.                                         System.out.print (Main.MA[i][j]+" ");
 77.                                 System.out.println ("");
 78.                         }
 79.                         Main.stopTime = System.currentTimeMillis();
 80.                         System.out.println ("Execution time: "+ (Main.stopTime -
Main.startTime)+"ms");
 81.                 } else {
 82.                         System.out.println ("<Result>");
 83.                         Main.stopTime = System.currentTimeMillis();
 84.                         System.out.println ("Execution time: "+ (Main.stopTime -
Main.startTime)+"ms");
 85.                 }
 86.
 87.    }
 88.
 89. }
-e No errors.
```

Література

[1] http://www.ccas.ru/paral/prog/semaphore.html

[2] http://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80_%28%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29

[3] Жуков І. А. Корочкін О. В. Паралельні та розподілені обчислення [Текст]. — Корнійчук, 2005. — ISBN 996-7599-36-1

[4] http://www.seas.gwu.edu/~adagroup/adalib_html/ada-html/a-sytaco.html

[5]http://msdn.microsoft.com/ru-ru/library/system.threading%28v=VS.90%29.aspx

[6]http://msdn.microsoft.com/ru-ru/library/system.threading.waithandle%28v=VS.90%29.aspx

[7]http://msdn.microsoft.com/ru-ru/library/54wk4yfd%28VS.90%29.aspx

[8]http://msdn.microsoft.com/ru-ru/library/system.threading.semaphorefullexception%28v=VS.90%29.aspx

[9]http://msdn.microsoft.com/ru-ru/library/system.threading.thread.sleep%28v=VS.90%29.aspx

[10]http://msdn.microsoft.com/en-us/library/aa911525.aspx

[11]http://ubuntuforums.org/showthread.php?t=614433

[12]http://msdn.microsoft.com/en-us/library/ms687032%28VS.85%29.aspx

[13]http://msdn.microsoft.com/en-us/library/ms679360%28v=VS.85%29.aspx

[14]http://msdn.microsoft.com/en-us/library/bb202751.aspx

[15]http://msdn.microsoft.com/en-us/library/ms682446%28v=VS.85%29.aspx

[16]http://msdn.microsoft.com/en-us/library/ms686670%28v=VS.85%29.aspx

[17]http://msdn.microsoft.com/en-us/library/ms687036%28v=VS.85%29.aspx

[18]http://libre.adacore.com/libre/

[19]http://ru.wikipedia.org/wiki/%D0%90%D0%B4%D0%B0_%28%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE

%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%29

[20]http://ru.wikipedia.org/wiki/C_Sharp