НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

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

(повна назва кафедри, циклової комісії)

**КУРСОВА РОБОТА**

з дисципліни «Паралельні та розподілені обчислення»

(назва дисципліни)

на тему: «Розробка програмного забезпечення для паралельних комп’ютерних систем»

Студента 3 курсу групи ІО-21

Напряму підготовки 050102 «Комп’ютерна інженерія»

Бурачика О.О.\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник доцент Корочкін О.В.

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

Кількість балів: \_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_\_\_\_

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

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

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

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

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

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

Київ- 2015 рік

Національний технічний університет України

“Київський політехнічний інститут”

Факультет (інститут) інформатики та обчислювальної техніки

(повна назва )

Кафедра обчислювальної техніки

(повна назва )

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 6.050102 «Комп’ютерна інженерія»

(шифр і назва)

**З А В Д А Н Н Я**

НА КУРСОВУ РОБОТУ СТУДЕНТУ

Бурачик Олексій Олександрович

(прізвище, ім’я, по батькові)

1. Тема роботи «Огляд засобів роботи з процесами в бібліотеці Win32»

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

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

2. Строк подання студентом роботи 18 травня 2015 р.

3. Вхідні дані до роботи

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

- огляд засобів роботи з процесами в бібліотеці Win32

- розробка і тестування програми ПРГ1 для ПКС ОП

- розробка і тестування програми ПРГ2 для ПКС ЛП

5. Перелік графічного матеріалу

- структурна схема ПКС ОП

- структурна схема ПКС ЛП

- схеми алгоритмів процесів і головної програми для ПРГ1

- схеми алгоритмів процесів і головної програми для ПРГ2.

7. Дата видачі завдання 02.02.2015

КАЛЕНДАРНИЙ ПЛАН

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

**Студент \_\_\_\_\_\_\_\_\_ \_\_\_\_**Бурачик О.О**.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(підпис ) (прізвище та ініціали)

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

(підпис ) (прізвище та ініціали)

**ПОЯСНЮВАЛЬНА  
ЗАПИСКА**

ЗМІСТ

[ВСТУП 6](#_Toc412474400)

[РОЗДІЛ 1. ОГЛЯД ЗАСОБІВ РОБОТИ З ПРОЦЕСАМИ В БІБЛІОТЕЦІ WIN32 7](#_Toc412474401)

[1.1Процеси в бібліотеці Win32 8](#_Toc412474402)

[1.2 Семафори 11](#_Toc412474403)

[1.3 М’ютекси 12](#_Toc412474404)

[1.4 Події 13](#_Toc412474405)

[1.5Висновки до першого розділ 14](#_Toc412474406)

[РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС ОП 15](#_Toc412474407)

[РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС ЛП 16](#_Toc412474408)

[ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ ДО РОБОТИ 17](#_Toc412474409)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 18](#_Toc412474410)

[Додатки 19](#_Toc412474411)

# ВСТУП

Паралельним обчисленням в останній час приділяється досить велика увага. Загалом це пов’язане з двома факторами. Перший фактор обумовлений науково-технічним прогресом, в результаті якого з’явилися нові області знань, які потребують ресурсномістких розрахунків, виконання яких можливе лише на базі високовиробничої техніки, за допомогою методів паралельних і розподілених обчислень. Другим суттєвим фактором являється повсюдне розповсюдження паралельних комп’ютерних систем. Паралельні інформаційні технології перетворилися з вузько направленої дисципліни в необхідну складову комплексу знань розробника сучасного програмного забезпечення.

В даній курсовій роботі розглядаються питання розробки програмного забезпечення для паралельних комп’ютерних систем.

В першому розділі «Огляд засобів роботи з процесами в бібліотеці Win32» описані основні принципи роботи бібліотеки Win32 і наведено приклади найголовніших функцій.

Другий та третій розділи присвячені розробці програми для обчислення математичної задачі в паралельній комп’ютерній системі зі спільною та локальною пам’яттю відповідно. Програмне забезпечення для комп’ютерної системи зі спільною пам’яттю розроблено на мові Java з використанням стандартної бібліотеки. Проведено тестування отриманих програмних продуктів і зроблено висновки по їх ефективності.

Лістинги та алгоритми розроблених програм наведено у додатках.

# РОЗДІЛ 1. ОГЛЯД ЗАСОБІВ РОБОТИ З ПРОЦЕСАМИ В БІБЛІОТЕЦІ WIN32

Windows API (application programming interfaces) — загальне найменування для цілого набору базових функцій інтерфейсів програмування, додатків операційних систем сімейства Windows корпорації «Microsoft».

Windows API представляє собою множину функцій, структур даних і числових констант. Усі мови програмування, здатні викликати такі функції і оперувати такими типами даних в програмах, які виконуються в системі Windows, можуть користуватися цим API. В приватному випадку це мови C++, Pascal, Visual Basic і багато інших.

Робота через Windows API — це найближчий до системи спосіб взаємодії з нею прикладних програм. Нижчий рівень доступу, необхідніший тільки для драйверів пристроїв, в поточних версіях Windows надається через Windows Driver Model.

Версії WinAPI:

* Win16 — перша версія Windows API для 16-розрядних версій Windows. Спочатку називався просто Windows API, потім почав називатися Win16 для розрізнення із Win32.
* Win32 — 32-розрядний API для сучасних версій Windows. Найпопулярніша нині версія. Базові функції цього API реалізовані в DLL kernel32.dll і advapi32.dll. Win32 з'явився разом з Windows NT і потім був перенесений (у декілька обмеженому вигляді) в системи серії Windows 9x.
* Win64 — 64-розрядна версія Win32, що містить додаткові функції для використання на 64-розрядних комп'ютерах. Win64 API можна знайти тільки в 64-розрядних версіях Windows XP і Windows Server 2003[1].

## **Процеси в бібліотеці Win32**

Процес – абстрактне поняття, що містить в собі опис певних операцій, пов’язаних з виконанням програми в комп’ютерній системі[2]. Виконання кожного процесу починається з первинного потоку, також в процесі свого виконання процес може створювати інші потоки.

Бібліотека Win32 включає в себе досить великий набір функцій, призначених для роботи з процесами.

Табл.1.1 Функції Win32API для роботи з процесами

|  |  |
| --- | --- |
| Функція Win32 АРI | Призначення |
| CreateProcess | Створює новий процес і його головний потік. |
| ExitProcess | Завершує виконання процесу і всіх його потоків |
| GetCurrentProcess | Повертає псевдо дескриптор поточного процесу |
| DuplicateHandle | Створює копію об'єкту ядра |
| GetCurrentProcess ID | Повертає ідентифікатор поточного процесу(для кожного процесу він унікальний) |
| GetExitCodeProeess | Отримання статусу завершення потоку |
| SetPriorityClass | Повертає клас пріоритету конкретного процесу |
| TerminateProcess | Завершення виконання процесу і всіх його потоків |

Більш детально розглянемо деякі з функції створення, завершення, очікування процесів.

Функція CreateThread():

Handle New\_Thread = CreateThread(

LPSECURITY\_ATTRIBUTES atr, //атрибут безпеки

SIZE\_T size, //розмір стеку

LPTHREAD\_STERT\_ROUTUNE func, //функція потоку

LPVOID arg, //аргумент функції потоку

DWORD flag, //прапорець виконання

LPWORD ip); // ідентифікатор потоку

В результаті виконання функції CreateThread буде створено новий потік, функція котрого почне виконуватися одразу(або буде призупинена до виклику ResumeThread).Також функція повертає спеціальне значення типу Handle – хендл потоку, котрий може бути використаний для призупинки, знищування потоку, синхронізації[3].

Потік можна завершити чотирма способами:

* Функція потоку повертає управління
* Потік самознищується викликом функції ExitThread
* Інший потік процесу викликає функцію TerminateThread
* Завершується процес, який містить даний потік

Всі способи за виключенням першого, являються небажаними і використовують в основному в форс мажорних ситуаціях.

У бібліотеці Win32 існує три види функцій очікування:

1. Одиничні
2. Множинні
3. Попереджувальні

До одиночних відносяться SignalObjectAndWait, WaitForSingleOblect, WaitForSingleObjectEx, вони використовують ідентифікатор (типу Handle) об’єкта синхронізації і блокують процес. Наведемо приклад найбільш розповсюдженої з них WaitForSingleObject

DWORD WaitForSingleObject(

Handle hHandle, //ідентифікатор об’єкта, від якого очікується сигнал

DWORD dwMilliseconds, //тайм-аут в мілісекундах

BOOL bAlertable //прапорець раннього виконання

);  
 Множинні функції дозволяють потоку, який викликає процес, визначити масив, який містить один або декілька ідентифікаторів об’єктів синхронізації. До таких функцій відносяться: WaitForMultipleObject, WaitForMultipleObjectsEx, sgWaitForMultipleObjects, MsgWaitForMultipleObjects.

DWORD WaitForSingleObjects(

DWORD ncount, //кілікість об’єктів у масиві

CONST HANDLE \*lpHandles, // покажчик на масив ідентифікаторів

BOOL bWaitAll, //чекати одного або всіх

DWORD dwMilliseconds //тайм-аут

);

Попереджувальні функції MsgWaitForMultipleObjectsEx, SignalObjectAndWait, WaitForMultipleObjectEx, WaitForSingleObjectEx можуть додатково виконувати попереджувальні функції. Вони також можуть закінчуватися з досягненням деякої умови, але також і в разі появи в системній черзі введення, виведення інформації, або віддаленого виклику процедур.

## **1.2 Семафори**

Семафор - особливий тип розподільної змінної, яка обробляється двома неподільними операціями: P і V. Значення семафора - невід'ємні цілі числа. Операція V сигналізує, що подія відбулася: вона збільшує значення семафора на 1. Операція P призупиняє процес до моменту, коли подія відбудеться: вона чекає, коли значення семафора стане позитивним, і зменшує його на 1.

Поведінка семафорів більш складна, ніж у інших об’єктів синхронізації. Не дивлячись на те, що в них немає прив’язки до контексту потоку, як у м’ютекса. Кожен раз, коли WaitFor\*\* - функція визначає семафор в сигнальному стані, цей лічильник зменшується на одиницю[4]. Як тільки лічильник досягне нуля, семафор перейде в нейтральний стан. Створення семафорів у Win32 відбувається за допомогою функції CreateSemaphore().

**CreateSemaphore():**

HANDLE WINAPI CreateSemaphore(  
\_\_in\_opt LPSECURITY\_ATTRIBUTES lpSemaphoreAttributes,  
\_\_in LONG lInitialCount,  
\_\_in LONG lMaximumCount,  
\_\_in\_opt LPCTSTR lpName  
);

Як видно з прототипу функції, можна задати максимальне число лічильника (параметр LONG lMaximumCount) і одразу ініціалізувати цей лічильник деяким числом (LONG lInitialCount). ReleaseSemaphore може збільшувати цей лічильник, причому не обов’язково на одиницю, а на необхідне задане значення.

Об’єкт семафор може використовуватися для обмеження числа активних потоків в додатку або для інших більш складних задач.

## **1.3 М’ютекси**

М'ю́текс (*mutex*) – один із засобів роботи з процесами, призначений для захисту певного об’єкта у потоці від доступу інших потоків. М'ютекс є одним із засобів синхронізації роботи потоків або процесів[5].

Задача м’ютекса – захист об’єкту від доступу до нього інших потоків, відмінних від того, який заволодів м’ютексом. У кожний конкретний момент часу один і тільки один потік може володіти об’єктом, захищеним м’ютексом. Якщо іншому потоку потрібен доступ до змінної, захищеною м’ютексом, то цей потік буде блокуватися до тих пір, поки м’ютекс не буде вільний.

Створюється Mutex за допомогою функції CreateMutex():

HANDLE CreateMutex

(

LPSECURITY\_ATTRIBUTES lpMutexAttributes,// атрибут безпеки

BOOL bInitialOwner, //прапорець початкового володаря

LPCTSTR lpName //ім’я об’єкту

);

Результатом буд дескриптор об’єкту mutex. Для звільнення об’єкту слід викликати функцію ReleaseMutex():

BOOL ReleaseMutex

(

Handle hMutex //дескриптор м’ютекс

);

При успішному виконанні функція повертає нуль. По закінченню виконання дескриптор потрібно закрити, викликавши **CloseHandle().**

BOOL CloseHandle

(

Handle hMutex //дескриптор);

## **1.4 Події**

**Подія або event – це найбільш примітивний об’єкт синхронізації. По суті, це просто прапорець, який може знаходитися або в нейтральному стані, або в сигнальному. Створити event можна за допомогою функції CreateEvent().**

**Опис CreateEvent() :**

**HANDLE WINAPI CreateEvent(**

**LPSECURITY\_ATTRIBUTES lpEventAttributes, // атрибут безпеки**

**BOOL bManualReset,// атрибут автоскидання**

**BOOL bInitialState,// початковий стан**

**LPCTSTR lpName// ім’я**

**);**

Події мають можливість «автоскидання», для цього слід встановити параметр BOOL bManualReset у значення FALSE. В цьому випадку при обробці івента функцією WaitFor\*\*\* об'єкт автоматично перейде в нейтральний стан. Якщо ж bManualReset == TRUE, то скидання прапорця виконується функцією ResetEvent.

Події можуть бути іменованими і неіменованими. В першому випадку до події можна отримати доступ з іншого процесу, відкривши об’єкт за допомогою API OpenEvent[6]. Такі події зручно використовувати при міжпроцесорній синхронізації. Також події можна задати початковий стан за допомогою параметра BOOL bInitialState.

Події в зв'язку з WaitFor\*\* - функціями забезпечує простий і зручний спосіб синхронізації потоків.

## **1.5 Висновки до першого розділу**

1. Виконано загальний огляд бібліотеки Win32 і проаналізовано, ща дана бібліотека має досить гнучку структуру, що дозволяє будувати неоднорідні системи;
2. Розглянуто основні механізми(функції) для роботи з процесами: створення, переривання потоків, завершення потоків і управління взаємодією між ними.
3. Детально розглянуто засоби роботи з процесами, а саме: семафори, події, м’ютекси. Їх основні функції, атрибути та можливості.
4. На основі огляду бібліотек Win32, можна зробити висновок, що дана бібліотека має всі необхідні засоби для створення паралельних програм будь якої складності.

# РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС CП

Математична задача:

Структура ПКС СП:



## **2.1. Розробка паралельного математичного алгоритму**

Згідно з технічним завданням необхідно розробити паралельний алгоритм. Його можна розділити на наступні етапи:

1. MAH =α\*MB\*(MC\*MDH)+E\*MTH

ЗР: α, MB, MC, e

Пояснення до використовуваних констант:

*  – розмірність векторів і матриць;
*  – кількість ядер;
* .

## **2.2 Розробка алгоритмів процесів**

Оскільки розроблюване програмне забезпечення є масштабованим і працює на системі із кількістю процесорів P, то зручним варіантом реалізації є написання єдиного алгоритму для всіх задач.

|  |  |
| --- | --- |
| Задачі T(0) – T(P) | |
| Крок алгоритму | КД і ТС |
| Якщо tid = 1, ввести MD, MT, E, α . |  |
| Якщо tid = 1, **сигнал** всім іншим задачам про завершення вводу MD, MT, E, α. |  |
| Якщо tid = P, ввести MC, MB. |  |
| Якщо tid =P, **сигнал** всім іншим задачам про завершення вводу MC, MB  5.Якщо tid != 1, чекати завершення вводу MD, MT, E, α  6. Якщо tid != Р, чекати завершення вводу MC, MB |  |
| 7. Обчислення 1: |  |
| 8.Обчислення 2:  e= max(e,e1) |  |
| 9.Сигнал закінчення Обчислення 2. |  |
| 10.Чекати закінчення Обчислення 2 |  |
| 11. Копії  α 1 = α  MB1 = MB  MC1 = MC  e1=e |  |
| 12. Обчислення 3: |  |
| 13. Якщо tid != Р, сигнал про завершення розрахунків |  |
| 14. Якщо tid = Р, чекати завершення розрахунків в усіх задачах  15. Якщо tid = Р, вивести результат MA |  |

## **2.3. Розробка схеми взаємодії процесів**

На основі алгоритму для всіх задач, розглянутого в попередньому розділі, було розроблена структурна схема взаємодії задач (рис. Рисунок 2.1). На схемі зображено монітор, за допомогою якого виконується вся взаємодія між процесами (потоками). На схемі позначені всі потрібні для вирішення задачі синхронізації та взаємного виключення, захищені функції та входи.

На структурній схемі взаємодії задач введено такі засоби для вирішення задач взаємного виключення та синхронізації:

* Захищені методи SignalIn та WaitIn для синхронізації із завершенням вводу в Т(1) і Т(Р)
* Захищені методи Input\_α та Input\_MB\_MC, max\_e для керування доступом до спільних ресурсів α, MC, MB та зміни їх значень.
* Захищені методи SignMax та WaitMax для синхронізації задач T(1..P) із завершенням пошуку максимуму в Е
* Захищені методи Copy\_α, Copy\_e, Copy\_MB, Copy\_MС для керування доступом до спільних ресурсів α, e, MB, MС
* Захищені методи SignalOut та WaitOut для синхронізації задачі T(P) із завершенням всіх обчислень в інших задачах.



Рисунок 2.1 – Структурна схема взаємодії задач для ПРГ1

## **2.4. Розробка програми ПРГ1**

Програма ПРГ1 згідно технічного завдання розроблення на мові програмування Java. Для взаємодії процесів використовується концепція моніторів**.** Програма складається з чотирьох класів: основного Major.java та допоміжних Monitor.java , Tasks.java і Function.java.

Основний клас Major.java містить один метод:

* main – точка входу в програму, формує ідентифікатори id потоків, запускає потоки.

В основному класі задаються такі змінні: N – установка розмірності матриць та векторів, Р – налаштування кількості процесорів. Дані параметри задаються користувачем з клавіатури після запуску програми.

Допоміжний клас Monitor.java представляє собою клас, за допомогою якого відбувається взаємодія зі спільними ресурсами програми. Цей клас містить в собі захищені методи, які використовуються для вирішення задач взаємного виключення та синхронізації. Також в класі Monitor.java об’явлені змінні які є спільними ресурсами: α, e, MB, MC.

Допоміжний клас Tasks.java являє собою клас, в якому описані всі дії які виконуються в потоках. В класі описаний один метод - run(). Клас Tasks.java наслідує клас Threads тим самим дає можливість працювати з потоками.

Допоміжний клас Function.java представляє собою клас, в якому описані всі математичні дії, які .

## **2.5. Тестування програми ПРГ 1**

Метою проведення тестування є оцінка коефіцієнтів прискорення і коефіцієнтів ефективності для розроблених програм при їх виконанні на реальній паралельній обчислювальній системі. Для визначення вищевказаних коефіцієнтів проведений ряд експериментів із різними розмінностями операндів (N = 900,1200, 1500, 2400) і різною кількістю працюючих ядер ( P = 1, 2, 3, 4).

Для вимірювання час виконання програми використовувався системний таймер currentTimeMillis().

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:



Таблиця 2.1 – Час виконання програми ПРГ1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Кількість процесорів (P) | | | | |
| N | 1 | 2 | 3 | 4 |
| 900 | 1,23 | 0,745 | 0,648 | 0,443 |
| 1200 | 2,75 | 1,496 | 1,329 | 0,983 |
| 1500 | 4,524 | 2,734 | 2,190 | 2,1 |
| 2400 | 20,329 | 12,207 | 9,927 | 7,90 |

На основі даних із таблиці Таблиця 2.1 виконано розрахунок значень коефіцієнтів прискорення, які наведені в таблиці Таблиця 2.2.

Таблиця 2.2 – Коефіцієнти прискорення для програми ПРГ1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Кількість процесорів (P) | | | | |
| N | 1 | 2 | 3 | 4 |
| 900 | 1,00 | 1,792 | 2,119 | 2,309 |
| 1200 | 1,00 | 1,547 | 2,183 | 2,507 |
| 1500 | 1,00 | 1,669 | 2,141 | 2,173 |
| 2400 | 1,00 | 1,706 | 2,214 | 2,446 |

Коефіцієнти ефективності (таблиця Таблиця 2.3) обчислено за даними таблиці Таблиця 2.2.

Таблиця 2.3 – Коефіцієнти ефективності для програми ПРГ1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Кількість процесорів (P) | | | | |
| N | 1 | 2 | 3 | 4 |
| 900 | 100.00% | 89.6 | 70.3 | 65.22 |
| 1200 | 100.00% | 82.35 | 72.76 | 62.67 |
| 1500 | 100.00% | 83.45 | 71.06 | 54.32 |
| 2400 | 100.00% | 85.3 | 73.8 | 61.16 |

Використовуючи таблиці Таблиця 2.1‑Таблиця 2.3 побудовано графіки зміни коефіцієнтів прискорення і ефективності, па також зміни часу роботи програми в залежності від *N* і .

Рисунок 2.2 – Графік зміни часу виконання програми ПРГ1 в залежності від кількості ядер та значення N

Рисунок 2.3 – Графік зміни коефіцієнту прискорення програми ПРГ1 в залежності від кількості ядер при N = 900

Рисунок 2.4 – Графік зміни коефіцієнту прискорення програми ПРГ1 в залежності від кількості ядер при N = 1200

Рисунок 2.5 – Графік зміни коефіцієнту прискорення програми ПРГ1 в залежності від кількості ядер при N = 1500

Рисунок 2.6 – Графік зміни коефіцієнту прискорення програми ПРГ1 в залежності від кількості ядер при N = 2400

## **2.6. Висновки до розділу 2**

Виконано розробку програми ПРГ1 для ПСК СП з використанням мови Java і засобів синхронізації даної мови. Тестування програми показало наступне:

* Використання багатоядерної ПКС та програми ПРГ1 забезпечує скорочення часу обчислення заданої математичної задачі. Значення К*П* лежить в межах від 1,949 до 3,734
* Максимальне значення К*П* =2,214забезпечує ПКС з Р = 4 та N = 2400.
* Мінімальне значення К*П* = 1,547виявлено у ПКС з Р = 2 та N = 1200.
* Найефективніше програма ПРГ1 використовує ПКС з Р = 2 та N = 900. При цьому Ке = 89.6%
* Зі зростанням Р від 1 до 4 Ке  лінійно спадає від 100,00% до 54.32% при N = 1500.
* При збільшенні кількості потоків (більше ніж процесорів) показники Кп та Ке різко зменшуються.

# РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС ЛП

# ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ ДО РОБОТИ

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Windows Application Interface(WinAPI) [Електронний ресурс].− Режим доступу: https://msdn.microsoft.com/en-us/library/windows/desktop/ff818516− дата звернення 22.02.2015. – Назва з екрану.
2. Жуков І.А., Корочкін О.В. Паралельні та розподілені обчислення: Навч. посібник. – К.: Корнійчук, 2005. – 26 с.
3. Win32 Multithreading[Електронний ресурс]. – Режим доступу: http://www.firststeps.ru/mfc/winapi/r.php?118 – дата звернення 22.02.2015 – назва з екрану
4. Intel® Thread Profiler для Win32[Електроний ресурс]. – Режим доступу: https://software.intel.com/ru-ru/articles/using-intel-thread-profiler-for-win32-threads-philosophy-and-theory - дата звернення 22.02.2015 – назва з екрану
5. Процессы и потоки in-depth [Електронний ресурс]. Режим доступу: – https://habrahabr/post40227 - дата звернення 22.02.15 – назва з екрану
6. Работа с функциями Windows API[Електронний ресурс].- Режим доступу–http://compress.ru/article.aspx?id=11741 – дата звернення: 22.02.2015 – назва з екрану

# ДОДАТКИ

## **Додаток А. Лістинг програми ПРГ1**

public class Task extends Thread {

\* Created by Oleksii on 23.03.2015.

\*/

public class Tasks {

Monitor monitor;

Function func;

int id;

public Tasks(Function func, Monitor monitor, int id) {

this.monitor = monitor;

this.func = func;

this.id = id;

}

public void run(){

long before = System.currentTimeMillis();

int startid = (id \* Function.getH());

int endid;

if(id!=Function.getP()-1){

endid = (id+1)\*Function.getH();

}

else{

endid = Function.getN();

}

int a1 = 0;

int e1 = Integer.MAX\_VALUE;

int[] B1 = new int[Function.getN()];

int[][] MC1 = new int[Function.getN()][Function.getN()];

int[][] MB1 = new int[Function.getN()][Function.getN()];

System.out.println("Thread "+ (id+1) + " started");

if(id == 0){

Function.setMC(Function.FillMatr(Function.getN()));

Function.setE(Function.FillVec(Function.getN()));

MD1 = Function.FillMatr(Function.getN());

monitor.Input\_MT(MT1);

monitor.SignalIn();

}

if(id == Function.getP()-1){

Function.setMA(Function.FillMatr(Function.getN()));

Function.setMC(Function.FillMatr(Function.getN()));

Function.setMB(Function.FillMatr(Function.getN()));

a1 = 1;

MB1 = Function.FillMatr(Function.getN());

monitor.Input\_MB\_MC(MB1, a1);

monitor.SignalIn();

}

monitor.WaitIn();

for (int i = startid; i < endid; i++){

if (e1 < Function.getZ()[i]){

e1 = Function.getZ()[i];

}

}

monitor.max\_e(e1);

monitor.SignMax();

monitor.WaitMax();

if(id == 0){

a1 = monitor.Copy\_a();

e1 = monitor.Copy\_e();

}

if(id == Function.getP()-1){

MB1 = monitor.Copy\_MB();

MC1 = monitor.Copy\_MO();

e1 = monitor.Copy\_e();

}

else{

a1 = monitor.Copy\_a();

e1 = monitor.Copy\_e();

MB1 = monitor.Copy\_MB();

MC1 = monitor.Copy\_MO();

}

for (int i = startid; i < endid; i++) {

for (int j = 0; j < Function.getN(); j++) {

int temp = 0;

int temp1 = 0;

for (int k = 0; k < Function.getN(); k++) {

temp += Function.getMC()[i][j] \* MB1[j][k];

temp1 += Function.getME()[i][j] \* MC1[j][k];

}

temp = temp \* a1 + e1 \*temp1;

Function.getMA()[i][j] = temp;

}

}

if(id == Function.getP()-1){

monitor.WaitOut();

if(Function.getN() < 13){

Function.OutputMatr(Function.MA);

}

System.out.println("Thread "+ (id+1) + " end");

long after = System.currentTimeMillis();

long diff = after - before;

System.out.println("Working time = "+ (diff/1000) +"," +(diff%1000));

}

else{

monitor.SignOut();

System.out.println("Thread "+ (id+1) + " end");

}

}

}

/\*\*

\* Created by Oleksii on 23.03.2015.

\*/

public class Monitor {

private static int a;

private static int e = Integer.MAX\_VALUE;

private static int[][] MB = new int [Function.getN()][Function.getN()];

private static int[][] MO = new int[Function.getN()][Function.getN()];

private int SIn = 0;

private int SOut = 0;

private int SMax = 0;

synchronized void Input\_MB\_a (int[][] MB, int a){

Monitor.MB = MB;

Monitor.a = a;

}

synchronized void Input\_MO(int[][] MO){

Monitor.MO = MO;

}

synchronized int Copy\_a(){

return a;

}

synchronized int Copy\_e(){

return e;

}

synchronized int[][] Copy\_MB(){

return MB;

}

synchronized int[][] Copy\_MO(){

return MO;

}

synchronized void max\_e(int e){

if (e > Monitor.e)

Monitor.e = e;

}

synchronized void SignalIn(){

SIn++;

if(SIn == 2)

notifyAll();

}

synchronized void SignMax(){

SMax++;

if(SMax == Function.getP())

notifyAll();

}

synchronized void SignOut(){

SOut++;

if (SOut == Function.getP() - 1)

notifyAll();

}

synchronized void WaitIn(){

try{

if(SIn != 2)

wait();

}

catch(Exception e){

System.out.println(e.getLocalizedMessage());

}

}

synchronized void WaitOut(){

try{

if(SOut != Function.getP() - 1)

wait();

}

catch(Exception e){

System.out.println(e.getLocalizedMessage());

}

}

synchronized void WaitMax(){

try{

if(SMax != Function.getP())

wait();

}

catch(Exception e){

System.out.println(e.getLocalizedMessage());

}

}

}

/\*\*

\* Created by Oleksii on 23.03.2015.

\*/

public class Function {

static int N = 1000;

static int P = 1000;

static int H ;

static int[] Z = new int[getN()];

static int[][] MA = new int[getN()][getN()];

static int[][] MC = new int[getN()][getN()];

static int[][] ME = new int[getN()][getN()];

public static int[][] FillMatr(int N){

int[][] MM = new int[N][N];

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

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

MM[i][j] = 1;

}

}

return MM;

}

public static int[] FillVec(int N){

int[] V = new int[N];

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

V[i] = 1;

}

return V;

}

public static void OutputMatr(int[][] MM){

System.out.println("MM = [");

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

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

System.out.print(MM[i][j]+", ");

}

System.out.println();

}

System.out.println("].");

}

public static int getN() {

return N;

}

public static void setN(int n) {

N = n;

}

public static int getH() {

return H;

}

public static void setH(int h) {

H = h;

}

public static int getP() {

return P;

}

public static void setP(int p) {

P = p;

}

public static int[] getZ(){

return Z;

}

public static void setZ(int[] z){

Z = z;

}

public static int[][] getMA() {

return MA;

}

public static void setMA(int[][] mA) {

MA = mA;

}

public static int[][] getMC() {

return MC;

}

public static void setMC(int[][] mC) {

MC = mC;

}

public static int[][] getME() {

return ME;

}

public static void setME(int[][] mE) {

ME = mE;

}

}

/\*\*

\* Created by Oleksii on 23.03.2015.

\*/

public class Major {

private static final int N = 4;

private static final int P = 4;

public static void main(String[] args){

Monitor monitor = new Monitor();

Function func = new Function();

int id;

Tasks[] T = new Tasks[Function.getP()];

Function.setP(P);

Function.setN(N);

Function.setH(Function.getN()/Function.getP());

for(int i = 0; i < Function.P; i++){

id = i;

T[i] = new Tasks(func, monitor, id);

}

for(int i = 0; i < Function.getP(); i++){

T[i].start();

}

}

}